Contents

1 Migrating your Database to Cosmos DB

This repository contains the lab documents and source code for the exercises for the workshop **Migrating your Database to Cosmos DB**

1.1 Lab 2: Migrate MongoDB to Cosmos DB

In this lab, you'll take an existing MongoDB database and migrate it to Cosmos DB. You'll use the Azure Database Migration Service. You'll also see how to reconfigure existing applications that use the MongoDB database to connect to the Cosmos DB database instead.

The lab is based around an example system that captures temperature data from a series of IoT devices. The temperatures are logged in a MongoDB database, together with a timestamp. Each device has a unique ID. You will run a MongoDB application that simulates these devices and stores the data in the database. You will also use a second application that enables a user to query statistical information about each device. After migrating the database from MongoDB to Cosmos DB, you'll configure both applications to connect to Cosmos DB, and verify that they still function correctly.

1.2 Lab 3: Migrate Cassandra to Cosmos DB

In this lab, you'll migrate two datasets from Cassandra to Cosmos DB. You'll move the data in two ways. First, you'll export the data from Cassandra and use the CQLSH COPY command to import the database into Cosmos DB. Then, you'll migrate the data using Spark. You'll verify that migration was successful by running an application that queries data held in the original Cassandra database, and then reconfiguring the application to connect to Cosmos DB. The results of running the reconfigured application should remain the same.

The scenario for this lab concerns an ecommerce system. Customers can place orders for goods. The customer and order details are recorded in a Cassandra database. You have an application that generates summaries, such as the list of orders for a customer, the orders for a specific product, and various aggregations, such as the number of orders placed, and so on.

Both labs run using the Azure Cloud shell and the Azure portal.

2 Lab: Migrate MongoDB to Cosmos DB

2.1 Estimated Time

90 minutes

2.2 Scenario

In this lab, you'll take an existing MongoDB database and migrate it to Cosmos DB. You'll use the Azure Database Migration Service. You'll also see how to reconfigure existing applications that use the MongoDB database to connect to the Cosmos DB database instead.

The lab is based around an example system that captures temperature data from a series of IoT devices. The temperatures are logged in a MongoDB database, together with a timestamp. Each device has a unique ID. You will run a MongoDB application that simulates these devices and stores the data in the database. You will also use a second application that enables a user to query statistical information about each device. After migrating the database from MongoDB to Cosmos DB, you'll configure both applications to connect to Cosmos DB, and verify that they still function correctly.

The lab runs using the Azure Cloud shell and the Azure portal.

2.3 Objectives

In this lab, you will:

- Create a migration project
- Define the source and target for your migration
- Perform the migration
- Verify the migration

2.4 Notes

The complete instructions for this lab are available at https://github.com/MicrosoftLearning/DP-060T00A-Migrating-your-Database-to-Cosmos-DB/blob/master/Labs/Lab%202%20-%20Migrate%20MongoDB%20Workloads%20to%

All of the files for this lab are located at https://github.com/MicrosoftLearning/DP-160T00A-Migrating-your-Database-to-Cosmos-DB.

Students should download a copy of the lab files before performing the lab.

2.5 Lab Exercises

- Create a Migration Project
- Define Source and Target
- Perform the Migration
- Verify the Migration

2.6 Lab Review

In this lab, you took an existing MongoDB database and migrated it to Cosmos DB. You used the Azure Database Migration Service. You also saw how to reconfigure existing applications that use the MongoDB database to connect to the Cosmos DB database instead.

3 Lab: Migrate Cassandra to Cosmos DB

3.1 Estimated Time

90 minutes

3.2 Scenario

In this lab, you'll migrate two datasets from Cassandra to Cosmos DB. You'll move the data in two ways. First, you'll export the data from Cassandra and use the CQLSH COPY command to import the database into Cosmos DB. Then, you'll migrate the data using Spark. You'll verify that the migration was successful by running an application that queries data held in the original Cassandra database, and then reconfiguring the application to connect to Cosmos DB. The results of running the reconfigured application should remain the same.

The scenario for this lab concerns an ecommerce system. Customers can place orders for goods. The customer and order details are recorded in a Cassandra database. You have an application that generates summaries, such as the list of orders for a customer, the orders for a specific product, and various aggregations, such as the number of orders placed, and so on.

The lab runs using the Azure Cloud shell and the Azure portal.

3.3 Objectives

In this lab, you will:

- Export the schema
- Move data using CQLSH COPY
- Move data using Spark
- Verify the migration

3.4 Notes

The full instructions for this lab are available at https://github.com/MicrosoftLearning/DP-060T00A-Migrating-your-Database-to-Cosmos-DB/blob/master/Labs/Lab%203%20-%20Migrate%20Cassandra%20Workloads%20to%20Cosmos

All of the files for this lab are located at https://github.com/MicrosoftLearning/DP-160T00A-Migrating-your-Database-to-Cosmos-DB.

Students should download a copy of the lab files before performing the lab.

3.5 Lab Exercises

- Export the Schema
- Move Data Using CQLSH COPY
- Move Data Using Spark
- Verify Migration

3.6 Lab Review

In this lab, you migrated two datasets from Cassandra to Cosmos DB. You moved the data in two ways. First, you exported the data from Cassandra and used the CQLSH COPY command to import the database into Cosmos DB. Then, you migrated the data using Spark. You verified that migration was successful by running an application that queries data held in the original Cassandra database, and then reconfiguring the application to connect to Cosmos DB.

4 Lab 2: Migrate MongoDB Workloads to Cosmos DB

In this lab, you'll take an existing MongoDB database and migrate it to Cosmos DB. You'll use the Azure Database Migration Service. You'll also see how to reconfigure existing applications that use the MongoDB database to connect to the Cosmos DB database instead.

The lab is based around an example system that captures temperature data from a series of IoT devices. The temperatures are logged in a MongoDB database, together with a timestamp. Each device has a unique ID. You will run a MongoDB application that simulates these devices, and stores the data in the database. You will also use a second application that enables a user to query statistical information about each device. After migrating the database from MongoDB to Cosmos DB, you'll configure both applications to connect to Cosmos DB, and verify that they still function correctly.

The lab runs using the Azure Cloud shell and the Azure portal.

4.1 Exercise 1: Setup

In the first exercise, you'll create the MongoDB database for holding the data captured from temperature devices that your company manufactures.

4.1.1 Task 1: Create a Resource Group and Virtual Network

1. In your Internet browser, navigate to https://portal.azure.com and sign in.
2. In the Azure portal, select **Resource groups**, and then select **+Add**.
3. On the **Create a resource group page**, enter the following details:

Property	Value
Subscription	*<your-subscription>*
Resource Group	mongodbrg
Region	Select your nearest location

4. Select **Review + Create** and then select **Create**. Wait for the resource group to be created.
5. In the hamburger menu of the Azure portal, select **+ Create a resource**.
6. On the **New** page, in the **Search the Marketplace** box, type **Virtual Network**, and press Enter.
7. On the **Virtual Network** page, select **Create**.
8. On the **Create virtual network** page, enter the following details:

Property	Value
Subscription	*<your-subscription>*
Resource Group	mongodbrg
Name	databasevnet
Region	Select the same location that you specified for the resource group

9. Select **Next: IP Addresses**
10. On the **IP Addresses** page, under **IPv4 address space**, select the **10.0.0.0/16** space, and then to the right select the **Delete** button.
11. In the **IPv4 address space** list, enter **10.0.0.0/24**.
12. Select **+ Add subnet**. In the **Add subnet** pane, set the **Subnet name** to **default**, set the **Subnet address range** to **10.0.0.0/28**, and then select **Add**.
13. On the **IP Addresses** page, select **Next: Security**.
14. On the **Security** page, verify that **DDoS Protection Standard** is set to **Disable**, and **Firewall** is set to **Disable**. Select **Review + create**.
15. On the **Create virtual network** page, select **Create**. Wait for the virtual network to be created before continuing.

4.1.2 Task 2: Create a MongoDB Database Server

1. In the hamburger menu of the Azure portal, select **+ Create a resource**.
2. In the **Search the Marketplace** box, type **Ubuntu**, and then press Enter.
3. On the **Marketplace** page, select **Ubuntu Server 18.04 LTS**.
4. On the **Ubuntu Server 18.04 LTS** page, select **Create**.
5. On the **Create a virtual machine** page, enter the following details:

Property	Value
Subscription	*<your-subscription>*
Resource Group	mongodbrg
Virtual machine name	mongodbserver
Region	Select the same location that you specified for the resource group
Availability options	No infrastructure redundancy required
Image	Ubuntu Server 18.04 LTS - Gen1
Azure Spot instance	Unchecked
Size	Standard A1_v2
Authentication type	Password
Username	azureuser
Password	Pa55w.rdPa55w.rd
Confirm password	Pa55w.rdPa55w.rd
Public inbound ports	Allow selected ports
Select inbound ports	SSH (22)

6. Select **Next: Disks >**.
7. On the **Disks** page, leave the settings at their default, and then select **Next: Networking >**.

8. On the **Networking** page, enter the following details:

Property	Value
Virtual network	databasevnet
Subnet	default (10.0.0.0/28)
Public IP	(new) mongodbserver-ip
NIC network security group	Advanced
Configure network security group	(new) mongodbserver-nsg
Accelerated networking	Unchecked
Load balancing	Unchecked

9. Select **Review + create >**.
10. On the validation page, select **Create**.
11. Wait for the virtual machine to be deployed before continuing
12. In hamburger menu of the Azure portal, select **All resources**.
13. On the **All resources** page, select **mongodbserver-nsg**.
14. On the **mongodbserver-nsg** page, under **Settings**, select **Inbound security rules**.
15. On the **mongodbserver-nsg - Inbound security rules** page, select **+ Add**.
16. In the **Add inbound security rule** pane, enter the following details:

Property	Value
Source	Any
Source port ranges	*
Destination	Any
Destination port ranges	27017
Protocol	Any
Action	Allow
Priority	1030
Name	Mongodb-port
Description	Port that clients use to connect to MongoDB

17. Select **Add**.

4.1.3 Task 3: Install MongoDB

1. In the hamburger menu the Azure portal, select **All resources**.
2. On the **All resources** page, select **mongodbserver-ip**.
3. On the **mongodbserver-ip** page, make a note of the **IP address**.
4. In the toolbar at the top of the Azure portal, select **Cloud Shell**.
5. If the **You have no storage mounted** message box appears, select **Create storage**.
6. When the Cloud Shell starts, in the drop-down list above the Cloud Shell window, select **Bash**.
7. In the Cloud Shell, enter the following command to connect to the mongodbserver virtual machine. Replace *<ip address>* with the value of the **mongodbserver-ip** IP address:

```
ssh azureuser@<ip address>
```

8. At the prompt, type **yes** to continue connecting.
9. Enter the password **Pa55w.rdPa55w.rd**.
10. To import the MongoDB public GPG key, enter this command. The command should return **OK**:

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.0.asc | sudo apt-key add -
```

11. To create a list of packages, enter this command:

    ```
    echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.0 multiver
    ```

    The command should return text similar to `deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.0 multiverse`. This indicates that the package list has been successfully created.

12. To reload the package database, enter this command:

    ```
    sudo apt-get update
    ```

13. To install MongoDB, enter this command:

    ```
    sudo apt-get install -y mongodb-org
    ```

    The installation should proceed with messages about installing, preparing, and unpacking packages. It can take a few minutes for the installation to complete.

4.1.4 Task 4: Configure the MongoDB database

By default, the Mongo DB instance is configured to run without authentication. In this task, you'll configure MongoDB to bind to the local network interface so that it can accept connections from other computers. You'll also enable authentication and create the necessary user account to perform migration. Finally, you'll add an account that a test application can use to query the database.

1. To open the MongoDB configuration file, run this command:

    ```
    sudo nano /etc/mongod.conf
    ```

2. In the file, locate the **bindIp** setting, and set it to **0.0.0.0**.

3. Add the following setting. If there is already a `security` section, replace it with this code. Otherwise add the code on two new lines:

    ```
    security:
        authorization: 'enabled'
    ```

4. To save the configuration file, press Esc and then press CTRL + X. Press y and then Enter to save the modified buffer.

5. To restart the MongoDB service and apply your changes, enter this command:

    ```
    sudo service mongod restart
    ```

6. To connect to the MongoDB service, enter this command:

    ```
    mongo
    ```

7. At the > prompt, to switch to the **admin** database, run this command:

    ```
    use admin;
    ```

8. To create a new user named **administrator**, run the following command. You can enter the command on one line or across multiple lines for better readability. The command is executed when the `mongo` program reaches the semicolon:

    ```
    db.createUser(
        {
            user: "administrator",
            pwd: "Pa55w.rd",
            roles: [
                { role: "userAdminAnyDatabase", db: "admin" },
                { role: "clusterMonitor", db:"admin" },
                "readWriteAnyDatabase"
            ]
        }
    );
    ```

9. To exit the `mongo` program, enter this command;

    ```
    exit;
    ```

10. To connect to MongoDB with the new administrator's account, run this command:

```
mongo -u "administrator" -p "Pa55w.rd"
```

11. To switch to the **DeviceData** database, execute this command:

```
use DeviceData;
```

12. To create a user named **deviceadmin**, which the app will use to connect to the database, run this command:

```
db.createUser(
    {
        user: "deviceadmin",
        pwd: "Pa55w.rd",
        roles: [ { role: "readWrite", db: "DeviceData" } ]
    }
);
```

13. To exit the `mongo` program, enter this command;

```
exit;
```

14. Run the following command restart the mongodb service. Verify that the service restarts without any error messages:

```
sudo service mongod restart
```

15. Run the following command to verify that you can now log in to mongodb as the deviceadmin user:

```
mongo -u "deviceadmin" -p "Pa55w.rd" --authenticationDatabase DeviceData
```

16. At the > prompt, run the following command to quit the mongo shell:

```
exit;
```

17. At the bash prompt, run the following command to disconnect from the MongoDB server and return to the Cloud Shell:

```
exit
```

4.2 Exercise 2: Populate and Query the MongoDB Database

You have now created a MongoDB server and database. The next step is to demonstrate the sample applications that can populate and query the data in this database.

4.2.1 Task 1: Build and Run an App to Populate the MongoDB Database

1. In the Azure Cloud Shell, run the following command to download the sample code for this workshop:

```
git clone https://github.com/MicrosoftLearning/DP-160T00A-Migrating-your-Database-to-Cosmos-DB mig
```

2. Move to the **migration-workshop-apps/MongoDeviceDataCapture/MongoDeviceCapture** folder:

```
cd ~/migration-workshop-apps/MongoDeviceDataCapture/MongoDeviceDataCapture
```

3. Use the **Code** editor to examine the **TemperatureDevice.cs** file:

```
code TemperatureDevice.cs
```

The code in this file contains a class named **TemperatureDevice** that simulates a temperature device capturing data and saving it in a MongoDB database. It uses the MongoDB library for the .NET Framework. The **TemperatureDevice** constructor connects to the database using settings stored in the application configuration file. The **RecordTemperatures** method generates a reading and writes it to the database.

4. To close the code editor press CTRL + Q, and then open the **ThermometerReading.cs** file:

```
code ThermometerReading.cs
```

This file shows the structure of the documents that the application stores in the database. Each document contains the following fields:

- An object ID. The is the "_id" field generated by MongoDB to uniquely identify each document.

- A device ID. Each device has a number with the prefix ”Device”.
- The temperature recorded by the device.
- The date and time when the temperature was recorded.

5. To close the code editor press CTRL + Q, and then open the **App.config** file:

   ```
   code App.config
   ```

   This file contains the settings for connecting to the MongoDB database. Set the value for the **Address** key to the IP address of the MongoDB server that you recorded earlier.

6. To save the file, press CTRL + S, and the to close the code editor press CTRL + Q

7. To open the project file, enter this command:

   ```
   code MongoDeviceDataCapture.csproj
   ```

8. Check that the `<TargetFramework>` tags contain the value `netcoreapp3.1`. Replace this value if necessary.

9. To save the file, press CTRL + S, and the to close the code editor press CTRL + Q

10. Run the following command to rebuild the application:

    ```
    dotnet build
    ```

11. Run the application:

    ```
    dotnet run
    ```

    The application simulates 100 devices running simultaneously. Allow the application to run for a couple of minutes, and then press Enter to stop it.

4.2.2 Task 2: Build and run another app to query the MongoDB database

1. Move to the **migration-workshop-apps/MongoDeviceDataCapture/DeviceDataQuery** folder:

   ```
   cd ~/migration-workshop-apps/MongoDeviceDataCapture/DeviceDataQuery
   ```

   This folder contains another application that you can use to analyze the data captured by each device.

2. Use the **Code** editor to examine the **Program.cs** file:

   ```
   code Program.cs
   ```

   The application connects to the database (using the **ConnectToDatabase** method at the bottom of the file) and then prompts the user for a device number. The application uses the MongoDB library for the .NET Framework to create and run an aggregate pipeline that calculates the following statistics for the specified device:

   - The number of readings recorded.
   - The average temperature recorded.
   - The lowest reading.
   - The highest reading.
   - The latest reading.

3. To close the code editor press CTRL + Q, and then open the **App.config** file:

   ```
   code App.config
   ```

   As before, set the value for the **Address** key to the IP address of the MongoDB server that you recorded earlier.

4. To save the file, press CTRL + S, and the to close the code editor press CTRL + Q

5. To open the project file, enter this command:

   ```
   code DeviceDataQuery.csproj
   ```

6. Check that the `<TargetFramework>` tags contain the value `netcoreapp3.1`. Replace this value if necessary.

7. To save the file, press CTRL + S, and the to close the code editor press CTRL + Q

8. Build and run the application:

```
dotnet build
dotnet run
```

9. At the **Enter Device Number** prompt, enter a value between 0 and 99. The application will query the database, calculate the statistics, and display the results. Press Q + Enter to quit the application.

4.3 Exercise 3: Migrate the MongoDB Database to Cosmos DB

The next step is to take the MongoDB database and transfer it to Cosmos DB.

4.3.1 Task 1: Create a Cosmos Account and Database

1. Return to the Azure portal.
2. In the hamburger menu, select **+ Create a resource**.
3. On the **New** page, in the **Search the Marketplace** box, type **Azure Cosmos DB**, end then press Enter.
4. On the **Azure Cosmos DB** page, select **Create**.
5. On the **Create Azure Cosmos DB Account** page, enter the following settings:

Property	Value
Subscription	Select your subscription
Resource group	mongodbrg
Account Name	mongodb*nnn*, where *nnn* is a random number selected by you
API	Azure Cosmos DB for MongoDB API
Notebooks	Off
Location	Specify the same location that you used for the MongoDB server and virtual network
Capacity mode	Provisioned throughput
Apply Free Tier Discount	Apply
Account Type	Non-Production
Version	3.6
Geo-Redundancy	Disable
Multi-region Writes	Disable
Availability Zones	Disable

6. Select **Review + create**
7. On the validation page, select **Create**, and wait for the Cosmos DB account to be deployed.
8. In the hamburger menu of the Azure portal, select **All resources**, and then select your new Cosmos DB account (**mongodb*nnn***).
9. On the **mongodb*nnn*** page, select **Data Explorer**.
10. In the **Data Explorer** pane, select **New Collection**.
11. In the **Add Collection** pane, specify the following settings:

Property	Value
Database id	Select **Create new**, and then type **DeviceData**
Provision database throughput	Checked
Throughput	1000
Collection id	Temperatures
Storage capacity	Unlimited
Shard key	deviceID
My shard key is larger than 100 bytes	unchecked
Create a Wildcard Index on all fields	unchecked
Analytical store	Off

12. Select **OK**.

4.3.2 Task 2: Create the Database Migration Service

1. In the hamburger menu of the Azure portal, select **All services**.
2. In the **All services** search box, type **Subscriptions**, and then press Enter.
3. On the **Subscriptions** page, select your subscription.
4. On your subscription page, under **Settings**, select **Resource providers**.
5. In the **Filter by name** box, type **DataMigration**, and then select **Microsoft.DataMigration**.
6. If the **Status** is not **Registered**, select **Register**, and wait for the **Status** to change to **Registered**. It might be necessary to select **Refresh** to see the status change.
7. In the hamburger menu of the Azure portal, select **+ Create a resource**.
8. On the **New** page, in the **Search the Marketplace** box, type **Azure Database Migration Service**, and then press Enter.
9. On the **Azure Database Migration Service** page, select **Create**.
10. On the **Create Migration Service** page, enter the following settings:

Property	Value
Subscription	Select your subscription
Resource group	mongodbrg
Service Name	MongoDBMigration
Location	Select the same location that you used previously
Service mode	Azure
Pricing Tier	Standard: 1 vCores

11. Select **Next: Networking**.
12. On the **Networking** page, check the **databasevnet/default** checkbox, and then select **Review + create**
13. Select **Create**, and wait for the service to be deployed before continuing. This operation will take a few minutes.

4.3.3 Task 3: Create and Run a New Migration Project

1. In the hamburger menu of the Azure portal, select **Resource groups**.
2. In the **Resource groups** window, select **mongodbrg**.
3. In the **mongodbrg** window, select **MongoDBMigration**.
4. On the **MongoDBMigration** page, select **+ New Migration Project**.
5. On the **New migration project** page, enter the following settings:

Property	Value
Project name	MigrateTemperatureData
Source server type	MongoDB
Target server type	Cosmos DB (MongoDB API)
Choose type of activity	Offline data migration

6. Select **Create and run activity**.
7. When the **Migration Wizard** starts, on the **Source details** page, enter the following details:

Property	Value
Mode	Standard mode
Source server name	Specify the value of the **mongodbserver-ip** IP address that you recorded earlier
Server port	27017

Property	Value
User Name	administrator
Password	Pa55w.rd
Require SSL	unchecked

8. Select **Next: Select target**.
9. On the **Select target** page, enter the following details:

Property	Value
Mode	Select Cosmos DB target
Subscription	Select your subscription
Select Cosmos DB name	mongodb*nnn*
Connection string	Accept the connection string generated for your Cosmos DB account

10. Select **Next: Database setting**.
11. On the **Database setting** page, enter the following details:

Property	Value
Source Database	DeviceData
Target Database	DeviceData
Throughput (RU/s)	1000
Clean up collections	Unchecked

12. Select **Next: Collection setting**.
13. On the **Collection setting** page, select the dropdown arrow by the DeviceData database, enter the following details:

Property	Value
Name	Temperatures
Target Collection	Temperatures
Throughput (RU/s)	1000
Shard Key	deviceID
Unique	Leave blank

14. Select **Next: Migration summary**
15. On the **Migration summary** page, in the **Activity name** field, enter **mongodb-migration**, and then select **Start migration**.
16. On the **mongodb-migration** page, select **Refresh** every 30 seconds, until the migration has completed. Note the number of documents processed.

4.3.4 Task 4: Verify that Migration was Successful

1. In the hamburger menu of the Azure portal, select **All Resources**.
2. On the **All resources** page, select **mongodb*nnn***.
3. On the **mongodb*nnn** page, select **Data Explorer**.
4. In the **Data Explorer** pane, expand the **DeviceData** database, expand the **Temperatures** collection, and then select **Documents**.
5. In the **Documents** pane, scroll through the list of documents. You should see a document id (**_id**) and the shard key (**/deviceID**) for each document.

6. Select any document. You should see the details of the document displayed. A typical document looks like this:

```
{
    "_id" : ObjectId("5ce8104bf56e8a04a2d0929a"),
    "deviceID" : "Device 83",
    "temperature" : 19.65268837271849,
    "time" : 636943091952553500
}
```

7. In the toolbar in the **Document Explorer** pane, select **New Shell**.

8. In the **Shell 1** pane, at the > prompt, enter the following command, and then press Enter:

   ```
   db.Temperatures.count()
   ```

   This command displays the number of documents in the Temperatures collection. It should match the number reported by the Migration Wizard .

9. Enter the following command, and then press Enter:

   ```
   db.Temperatures.find({deviceID: "Device 99"})
   ```

   This command fetches and displays the documents for Device 99.

4.4 Exercise 4: Reconfigure and Run Existing Applications to Use Cosmos DB

The final step is to reconfigure your existing MongoDB applications to connect to Cosmos DB, and verify that they operate as before. This process requires you to modify the way in which your applications connect to the database, but the logic of your applications should remain unchanged.

1. In the **mongodb*nnn*** pane, under **Settings**, select **Connection String**.

2. On the **mongodb*nnn* Connection String** page, make a note of the following settings:

   - Host
   - Username
   - Primary Password

3. Return to the Cloud Shell window (reconnect if the session has timed out), and move to the **migration-workshop-apps/MongoDeviceDataCapture/DeviceDataQuery** folder:

   ```
   cd ~/migration-workshop-apps/MongoDeviceDataCapture/DeviceDataQuery
   ```

4. Open the App.config file in the Code editor:

   ```
   code App.config
   ```

5. In the **Settings for MongoDB** section of the file, comment out the existing settings.

6. Uncomment the settings in the **Settings for Cosmos DB Mongo API** section, and set the values for these settings as follows:

Setting	Value
Address	The **Host** from the **mongodb*nnn* Connection String** page
Username	The **Username** from the **mongodb*nnn* Connection String** page
Password	The **Primary Password** from the **mongodb*nnn* Connection String** page

   The completed file should look similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <appSettings>
        <add key="Database" value="DeviceData" />
        <add key="Collection" value="Temperatures" />

        <!-- Settings for MongoDB -->
        <!--add key="Address" value="nn.nn.nn.nn" />
```

```
        <add key="Port" value="27017" />
        <add key="Username" value="deviceadmin" />
        <add key="Password" value="Pa55w.rd" /-->
        <!-- End of settings for MongoDB -->

        <!-- Settings for CosmosDB Mongo API -->
        <add key="Address" value="mongodbnnn.documents.azure.com"/>
        <add key="Port" value="10255"/>
        <add key="Username" value="mongodbnnn"/>
        <add key="Password" value="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        <!-- End of settings for CosmosDB Mongo API -->
    </appSettings>
</configuration>
```

7. Save the file, and then close the Code editor.
8. Open the Program.cs file using the Code editor:

   ```
   code Program.cs
   ```

9. Scroll down to the **ConnectToDatabase** method.
10. Comment out the line that sets the credentials for connecting to MongoDB, and uncomment the statements that specify the credentials for connecting to Cosmos DB. The code should look like this:

    ```
    // Connect to the MongoDB database
    MongoClient client = new MongoClient(new MongoClientSettings
    {
        Server = new MongoServerAddress(address, port),
        ServerSelectionTimeout = TimeSpan.FromSeconds(10),

        //
        // Credential settings for MongoDB
        //

        // Credential = MongoCredential.CreateCredential(database, azureLogin.UserName, azureLogin.Sec

        //
        // Credential settings for CosmosDB Mongo API
        //

        UseTls = true,
        Credential = new MongoCredential("SCRAM-SHA-1", new MongoInternalIdentity(database, azureLogin

        // End of Mongo API settings
    });
    ```

    These changes are necessary because the original MongoDB database was not using a TLS connection. Cosmos DB always uses TLS.
11. Save the file, and then close the Code editor.
12. Rebuild and run the application:

    ```
    dotnet build
    dotnet run
    ```

13. At the **Enter Device Number** prompt, enter a device number between 0 and 99. The application should run exactly as before, except this time it is using the data held in the Cosmos DB database.
14. Test the application with other device numbers. Enter **Q** to finish.

You have successfully migrated a MongoDB database to Cosmos DB, and reconfigured an existing MongoDB application to connect to the Cosmos DB database.

4.5 Exercise 5: Clean Up

1. Return to the Azure portal.
2. In the hamburger menu, select **Resource groups**.
3. In the **Resource groups** window, select **mongodbrg**.
4. Select **Delete resource group**.
5. On the **Are you sure you want to delete "mongodbrg"** page, in the **Type the resource group name** box, enter **mongodbrg**, and then select **Delete**.

---

5 Lab 3: Migrate Cassandra Workloads to Cosmos DB

In this lab, you'll migrate two datasets from Cassandra to Cosmos DB. You'll move the data in two ways. First, you'll export the data from Cassandra and use the **CQLSH COPY** command to import the database into Cosmos DB. Then, you'll migrate the data using Spark. You'll verify that migration was successful by running queries against the data held in the Cosmos DB database.

The scenario for this lab concerns an ecommerce system. Customers can place orders for goods. The customer and order details are recorded in a Cassandra database.

The lab runs using the Azure Cloud shell and the Azure portal.

5.1 Exercise 1: Setup

In the first exercise, you'll create the Cassandra database for holding the customer and order data.

5.1.1 Task 1: Create a Resource Group and Virtual Network

1. In your Internet browser, navigate to https://portal.azure.com and sign in.
2. In the Azure portal, select **Resource groups**, and then select **+Add**.

3. On the **Create a resource group page**, enter the following details:

Property	Value
Subscription	*<your-subscription>*
Resource Group	cassandradbrg
Region	Select your nearest location

4. Select **Review + create**.
5. Select **Create**, and wait for the resource group to be created.
6. In the left-hand pane of the Azure portal, select **+ Create a resource**.
7. On the **New** page, in the **Search the Marketplace** box, type **Virtual Network**, and press Enter.
8. On the **Virtual Network** page, select **Create**.
9. On the **Basics** page, enter the following details:

Property	Value
Subscription	*<your-subscription>*
Resource Group	cassandradbrg
Name	databasevnet
Region	Select the same location that you specified for the resource group

10. Select **Next:IP Addresses**.
11. On the **IP Addresses** page, set the **IPv4 address space** to **10.0.0.0/24**.
12. Select the default subnet, and then select **Remove subnet**.
13. Select **+ Add subnet**. In the **Add subnet** pane, set the **Subnet name** to **default**, set the **Subnet address range** to **10.0.0.0/28**, and then select **Add**.
14. On the **IP Addresses** page, select **Next: Security**.
15. On the **Security** page, enter the following details:

Property	Value
BastionHost	Disabled
DDoS Protection Standard	Disabled
Firewall	Disabled

16. Select **Review + create**
17. On the **Review + create** page, select **Create** and then wait for the virtual network to be created before continuing.

5.1.2 Task 2: Create a Cassandra Database Server

1. In the left-hand pane of the Azure portal, select **+ Create a resource**.
2. In the **Search the Marketplace** box, type **Cassandra Certified by Bitnami**, and then press Enter.
3. On the **Cassandra Certified by Bitnami** page, select **Create**.
4. On the **Create a virtual machine** page, enter the following details.

Property	Value
Subscription	*<your-subscription>*
Resource Group	cassandradbrg
Virtual machine name	cassandraserver
Region	Select the same location that you specified for the resource group

Property	Value
Availability options	No infrastructure redundancy required
Image	Cassandra Certified by Bitnami - Gen1
Size	Standard D2 v2
Authentication type	Password
Username	azureuser
Password	Pa55w.rdPa55w.rd
Confirm password	Pa55w.rdPa55w.rd

5. Select **Next: Disks >**.
6. On the **Disks** page, leave the settings at their default, and then select **Next: Networking >**.
7. On the **Networking** page, enter the following details:

Property	Value
Virtual network	databasevnet
Subnet	default (10.0.0.0/28)
Public IP	(new) cassandraserver-ip
NIC network security group	Advanced
Configure network security group	(new) cassandraserver-nsg
Accelerated networking	Off
Load balancing	No

8. Select **Next: Management >**.
9. On the **Management** page, leave the settings at their default, and then select **Next: Advanced >**.
10. On the **Advanced** page, leave the settings at their default, and then select **Next: Tags >**.
11. On the **Tags** page, leave the settings at their default, and then select **Next: Review + create >**.
12. On the validation page, select **Create**.
13. Wait for the virtual machine to be deployed before continuing.
14. In the left-hand pane of the Azure portal, select **All resources**.
15. On the **All resources** page, select **cassandraserver-nsg**.
16. On the **cassandraserver-nsg** page, under **Settings**, select **Inbound security rules**.
17. On the **cassandraserver-nsg - Inbound security rules** page, select **+ Add**.
18. In the **Add inbound security rule** pane, enter the following details:

Property	Value
Source	Any
Source port ranges	*
Destination	Any
Destination port ranges	9042
Protocol	Any
Action	Allow
Priority	1020
Name	Cassandra-port
Description	Port that clients use to connect to Cassandra

19. Select **Add**.

5.1.3 Task 3: Populate the Cassandra Database

1. In the left-hand pane of the Azure portal, select **All resources**.

2. On the **All resources** page, select **cassandraserver-ip**.
3. On the **cassandraserver-ip** page, make a note of the **IP address**.
4. In the toolbar at the top of the Azure portal, select **Cloud Shell**.
5. If the **You have no storage mounted** message box appears, select **Create storage**.
6. When the Cloud Shell starts, in the drop-down list above the Cloud Shell window, select **Bash**.
7. In the Cloud Shell, if you haven't performed Lab 2, run the following command to download the sample code and data for this workshop:

   ```
   git clone https://github.com/MicrosoftLearning/DP-160T00A-Migrating-your-Database-to-Cosmos-DB mig
   ```

8. Move to the **migration-workshop-apps/Cassandra** folder:

   ```
   cd ~/migration-workshop-apps/Cassandra
   ```

9. Enter the following commands to copy the setup scripts and data to the **cassandraserver** virtual machine. Replace *<ip address>* with the value of the **cassandraserver-ip** IP address:

   ```
   scp *.* azureuser@<ip address>:~
   ```

10. At the prompt, type **yes** to continue connecting.
11. At the **Password** prompt, enter the password **Pa55w.rdPa55w.rd**
12. Type the following command to connect to the **cassandraserver** virtual machine. Specify the IP address of the **cassandraserver** virtual machine:

    ```
    ssh azureuser@<ip address>
    ```

13. At the **Password** prompt, enter the password **Pa55w.rdPa55w.rd**
14. Run the following command to connect to the Cassandra database, create the tables required by this lab, and populate them.

    ```
    bash upload.sh
    ```

    The script creates two keyspaces named **customerinfo** and **orderinfo**. The script creates a table named **customerdetails** in the **customerinfo** keyspace, and two tables named **orderdetails** and **orderline** in the**orderinfo** keyspace.

15. Run the following command, and make a note of the default password in this file:

    ```
    cat bitnami_credentials
    ```

16. Start the Cassandra Query Shell as the user **cassandra** (this is the name of the default Cassandra user created when the virtual machine was set up). Replace *<password>* with the default password from the previous step:

    ```
    cqlsh -u cassandra -p <password>
    ```

17. At the **cassandra@cqlsh** prompt, run the following command. This command displays the first 100 rows from the **customerinfo.customerdetails** table:

    ```
    select *
    from customerinfo.customerdetails
    limit 100;
    ```

    Note that the data is clustered by the **stateprovince** column, and then ordered by **customerid**. This grouping enables applications to quickly find all customers located in the same region.

18. Run the following command. This command displays the first 100 rows from the **orderinfo.orderdetails** table:

    ```
    select *
    from orderinfo.orderdetails
    limit 100;
    ```

    The **orderinfo.orderdetails** table contains a list of orders placed by each customer. The data recorded includes the date the order was placed, and the value of the order. The data is clustered by the **customerid** column, so that applications can quickly find all orders for a specified customer.

19. Run the following command. This command displays the first 100 rows from the **orderinfo.orderline** table:

    ```
    select *
    from orderinfo.orderline
    limit 100;
    ```

    This table contains the items for each order. The data is clustered by the **orderid** column, and sorted by **orderline**.

20. Quit the Cassandra Query Shell:

    ```
    exit;
    ```

21. At the **bitnami@cassandraserver** prompt, type the following command to disconnect from the Cassandra server and return to the Cloud Shell:

    ```
    exit
    ```

5.2 Exercise 2: Migrate Data from Cassandra to Cosmos DB Using the CQLSH COPY Command

You have now created and populated a Cassandra database. In this exercise, you will create a Cosmos DB account using the Cassandra API, and then migrate the data from your Cassandra database to a database in the Cosmos DB account.

5.2.1 Task 1: Create a Cosmos Account and Database

1. Return to the Azure portal.
2. In the left pane, select **+ Create a resource**.
3. On the **New** page, in the **Search the Marketplace** box, type **Azure Cosmos DB**, end then press Enter.
4. On the **Azure Cosmos DB** page, select **Create**.
5. On the **Create Azure Cosmos DB Account** page, enter the following setting:

Property	Value
Subscription	Select your subscription
Resource group	cassandradbrg
Account Name	cassandra*nnn*, where *nnn* is a random number selected by you
API	Cassandra
Notebooks	Off
Location	Specify the same location that you used for the Cassandra server and virtual network
Capacity mode	Provisioned throughput
Apply Free Tier Discount	Apply
Account Type	Non-Production
Geo-Redundancy	Disable
Multi-region Writes	Disable
Availability Zones	Disable

6. Select **Review + create**.
7. On the validation page, select **Create**, and wait for the Cosmos DB account to be deployed.
8. In the left-hand pane, select **All resources**.
9. On the **All resources** page, select your Cosmos DB account (**cassandra*nnn***).
10. On the **cassandra*nnn*** page, select **Data Explorer**.
11. In the **Data Explorer** pane, select **New Table**.
12. In the **Add Table** pane, specify the following settings, and then select **OK**:

Property	Value
Keyspace name	Select **Create new**, and then type **customerinfo**
Provision keyspace throughput	de-selected
Enter tableId	customerdetails
CREATE TABLE box	(customerid int, firstname text, lastname text, email text, stateprovince text, PRIMARY
Throughput	10000

13. In the **Data Explorer** pane, select **New Table**.
14. In the **Add Table** pane, specify the following settings:

Property	Value
Keyspace name	Select **Create new**, and then type **orderinfo**
Provision keyspace throughput	de-selected
Enter tableId	orderdetails
CREATE TABLE box	(orderid int, customerid int, orderdate date, ordervalue decimal, PRIMARY KEY ((custo
Throughput	10000

15. Select **OK**.
16. In the **Data Explorer** pane, select **New Table**.
17. In the **Add Table** pane, specify the following settings:

Property	Value
Keyspace name	Select **Use existing**, and then select **orderinfo**
Enter tableId	orderline
CREATE TABLE box	(orderid int, orderline int, productname text, quantity smallint, orderlinecost decimal, PRIMARY
Throughput	10000

18. Select **OK**.

5.2.2 Task 2: Export the Data from the Cassandra Database

1. Return to the Cloud Shell.
2. Run the following command to connect to the cassandra server. Replace *<ip address>* with the IP address of the virtual machine. Enter the password **Pa55w.rdPa55w.rd** when prompted:

```
ssh azureuser@<ip address>
```

3. Start the Cassandra Query Shell. Specify the password from the **bitnami_credentials** file:

```
cqlsh -u cassandra -p <password>
```

4. At the **cassandra@cqlsh** prompt, run the following command. This command downloads the data in the **customerinfo.customerdetails** table and writes it to a file named **customerdata**. The command should export 19119 rows:

```
copy customerinfo.customerdetails
to 'customerdata';
```

5. Run the following command to export the data in the **orderinfo.orderdetails** table to a file named **orderdata**. This command should export 31465 rows:

```
copy orderinfo.orderdetails
to 'orderdata';
```

6. Run the following command to export the data in the **orderinfo.orderline** table to a file named **orderline**. This command should export 121317 rows:

```
copy orderinfo.orderline
to 'orderline';
```

7. Close the Cassandra Query Shell:

   ```
   exit;
   ```

5.2.3 Task 3: Import the Data to Cosmos DB

1. Switch back to your Cosmos DB account in the Azure portal.
2. Under **Settings**, select **Connection String**, and make a note of the following items:
   - Contact Point
   - Port
   - Username
   - Primary Password
3. Return to the Cassandra server, and start the Cassandra Query Shell. This time, connect to your Cosmos DB account. Replace the arguments to the **cqlsh** command with the values you just noted:

   ```
   export SSL_VERSION=TLSv1_2
   export SSL_VALIDATE=false

   cqlsh <contact point> <port> -u <username> -p <primary password> --ssl
   ```

   Note that Cosmos DB requires an SSL connection.
4. At the **cqlsh** prompt, run the following commands to import the data that you previously exported from the Cassandra database:

   ```
   copy customerinfo.customerdetails from 'customerdata' with chunksize = 200;
   copy orderinfo.orderdetails from 'orderdata' with chunksize = 200;
   copy orderinfo.orderline from 'orderline' with chunksize = 100;
   ```

   These commands should import 19119 **customerdetails** rows, 31465 **orderdetails** rows, and 121317 **orderline** rows. If these commands fail with a timeout error, you can resolve the problem either by:
   - Increasing the throughput for the corresponding table in the Azure portal (and reducing it again afterwards, to avoid excessive throughput charges), or
   - Lowering the chunk size of each copy operation. Decreasing the chunk size slows the ingestion rate, whereas increasing the throughput raises the costs.

5.2.4 Task 4: Verify that Data Migration was Successful

1. Return to your Cosmos DB account in Azure portal, and then select **Data Explorer**.
2. In the **Data Explorer** pane, expand the **customerinfo** keyspace, expand the **customerdetails** table, and then select **Rows**. Verify that a set of customers appears.
3. Select **Add new clause**.
4. In the **Field** box, select **stateprovince**, and in the **Value** box, type **Tasmania**.
5. In the toolbar, select **Run Query**. Verify that the query returns 106 rows.
6. In the **Data Explorer** pane, expand the **orderinfo** keyspace, expand the **orderdetails** table, and then select **Rows**.
7. Select **Add new clause**.
8. In the **Field** box, select **customerid**, and in the **Value** box, type **13999**.
9. In the toolbar, select **Run Query**. Verify that the query returns 2 rows. Note the **orderid** for the first row (it should be 46899).
10. In the **Data Explorer** pane, expand the **orderinfo** keyspace, expand the **orderline** table, and then select **Rows**.
11. Select **Add new clause**.
12. In the **Field** box, select **orderid**, and in the **Value** box, type **46899**.
13. In the toolbar, select **Run Query**. Verify that the query returns 1 row, listing the product being ordered as **Road-550-W Yellow, 38**.

You have successfully migrated a Cassandra database to Cosmos DB by using the CQLSH COPY command.

5.2.5 Task 5: Clean Up

1. Switch back to the Cassandra Query Shell running on your Cassandra server. The shell should still be connected to your Cosmos DB account. If you closed the shell earlier, then reopen it, as follows:

```
export SSL_VERSION=TLSv1_2
export SSL_VALIDATE=false

cqlsh <contact point> <port> -u <username> -p <primary password> --ssl
```

2. In the Cassandra Query Shell, run the following commands to remove the keyspaces (and tables):

```
drop keyspace customerinfo;
drop keyspace orderinfo;
exit;
```

5.3 Exercise 3: Migrate Data from Cassandra to Cosmos DB Using Spark

In this exercise, you'll migrate the same data used previously, but this time you'll use Spark from an Azure Databricks notebook.

5.3.1 Task 1: Create a Spark Cluster

1. In the Azure portal, in the left-hand pane, select **+ Create a resource**.
2. In the **New** pane, in the **Search the Marketplace** box, type **Azure Databricks**, and then press Enter.
3. On the **Azure Databricks** page, select **Create**.
4. On the **Azure Databricks Service** page, enter the following details:

Property	Value
Subscription	*<your-subscription>*
Resource Group	Use existing, cassandradbrg
Workspace name	CassandraMigration
Location	Select the same location that you specified for the resource group
Pricing Tier	Standard

5. Select **Review + create**.
6. On the **Review + create** page, select **Create** and then wait for the Databricks Service to be deployed.
7. In the left-hand pane, select **Resource groups**, select **cassandradbrg**, and then select the **CassandraMigration** Databricks Service.
8. On the **CassandraMigration** page, select **Launch Workspace**.
9. On the **Azure Databricks** page, under **Common Tasks**, select **New Cluster**.
10. On the **New Cluster** page, enter the following settings:

Property	Value
Cluster Name	MigrationCluster
Cluster Mode	Standard
Pool	None
Databrick Runtime Version	Runtime: 5.5 LTS (Scala 2.11, Spark 2.4.3)
Python Version	3
Enable autoscaling	Selected
Terminate after	60
Worker Type	Leave the settings at their default
Driver Type	Same as worker

[!NOTE] When you select the runtime version, ensure you select the version without GPU support, otherwise you will receive an incompatibility error message.

11. Select **Create Cluster**.
12. Wait for the cluster to be created; the state of the **MigrationCluster** is reported as **Running** when the cluster is ready. This process will take several minutes.

5.3.2 Task 2: Create a Notebook for Migrating Data

1. In the pane to the left, select **Clusters**, select the **MigrationCluster**, select the **Libraries** tab, and then select **Install New**.
2. In the **Install Library** dialog, enter the following settings:

Property	Value
Library Source	Maven
Coordinates	com.datastax.spark:spark-cassandra-connector_2.11:2.4.3
Repository	Leave blank
Exclusions	Leave blank

3. Select **Install**. This library contains the classes for connecting to Cassandra from Spark.
4. When the connector library is installed, select **Install New** to install another library.
5. In the **Install Library** dialog, enter the following settings:

Property	Value
Library Source	Maven
Coordinates	com.microsoft.azure.cosmosdb:azure-cosmos-cassandra-spark-helper:1.2.0
Repository	Leave blank
Exclusions	Leave blank

6. Select **Install**. This library contains the classes for connecting to Cosmos DB from Spark.
7. In the pane to the left, select **Azure Databricks**.
8. On the **Azure Databricks** page, under **Common Tasks**, select **New Notebook**.
9. In the **Create Notebook** dialog box, enter the following settings:

Property	Value
Name	MigrateData
Language	Scala
Cluster	MigrationCluster

10. Select **Create**.

5.3.3 Task 3: Connect to Cosmos DB and Create Tables

1. In the first cell of the notebook, enter the following code:

```
// Import libraries

import org.apache.spark.sql.cassandra._
import org.apache.spark.sql._
import org.apache.spark._
import com.datastax.spark.connector._
import com.datastax.spark.connector.cql.CassandraConnector
import com.microsoft.azure.cosmosdb.cassandra
```

   This code imports the types required to connect to Cosmos DB and Cassandra from Spark.

2. In the toolbar on the right of the cell, select the drop-down arrow, and then select **Add Cell Below**.
3. In the new cell, enter the following code. Specify the Contact Point, Username, and Primary Password with the values for your Cosmos DB account (you recorded these values in the previous exercise):

```
// Configure connection parameters for Cosmos DB
```

```
val cosmosDBConf = new SparkConf()
    .set("spark.cassandra.connection.host", "<contact point>")
    .set("spark.cassandra.connection.port", "10350")
    .set("spark.cassandra.connection.ssl.enabled", "true")
    .set("spark.cassandra.auth.username", "<username>")
    .set("spark.cassandra.auth.password", "<primary password>")
    .set("spark.cassandra.connection.factory",
        "com.microsoft.azure.cosmosdb.cassandra.CosmosDbConnectionFactory")
    .set("spark.cassandra.output.batch.size.rows", "1")
    .set("spark.cassandra.connection.connections_per_executor_max", "1")
    .set("spark.cassandra.output.concurrent.writes", "1")
    .set("spark.cassandra.concurrent.reads", "1")
    .set("spark.cassandra.output.batch.grouping.buffer.size", "1")
    .set("spark.cassandra.connection.keep_alive_ms", "600000000")
```

This code sets the Spark session parameters to connect to your Cosmos DB account

4. Add another cell below the current one, and enter the following code:

```
// Create keyspaces and tables

val cosmosDBConnector = CassandraConnector(cosmosDBConf)

cosmosDBConnector.withSessionDo(session => session.execute("CREATE KEYSPACE customerinfo WITH repli
cosmosDBConnector.withSessionDo(session => session.execute("CREATE TABLE customerinfo.customerdeta

cosmosDBConnector.withSessionDo(session => session.execute("CREATE KEYSPACE orderinfo WITH replica
cosmosDBConnector.withSessionDo(session => session.execute("CREATE TABLE orderinfo.orderdetails (o

cosmosDBConnector.withSessionDo(session => session.execute("CREATE TABLE orderinfo.orderline (orde
```

These statements rebuild the orderinfo and customerinfo keyspaces, together with the tables. Each table is provisioned with 10000 RU/s of throughput.

5.3.4 Task 4: Connect to the Cassandra Database and Retrieve Data

1. In the notebook, add another cell, and enter the following code. Replace *<ip address>* with the IP address of the virtual machine, and specify the password you retrieved earlier from the **bitnami_credentials** file:

```
// Configure connection parameters for the source Cassandra database

val cassandraDBConf = new SparkConf()
    .set("spark.cassandra.connection.host", "<ip address>")
    .set("spark.cassandra.connection.port", "9042")
    .set("spark.cassandra.connection.ssl.enabled", "false")
    .set("spark.cassandra.auth.username", "cassandra")
    .set("spark.cassandra.auth.password", "<password>")
    .set("spark.cassandra.connection.connections_per_executor_max", "10")
    .set("spark.cassandra.concurrent.reads", "512")
    .set("spark.cassandra.connection.keep_alive_ms", "600000000")
```

2. Add another cell, and enter the following code:

```
// Retrieve the customer and order data from the source database

val cassandraDBConnector = CassandraConnector(cassandraDBConf)
var cassandraSparkSession = SparkSession
    .builder()
    .config(cassandraDBConf)
    .getOrCreate()

val customerDataframe = cassandraSparkSession
    .read
```

```
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "customerdetails", "keyspace" -> "customerinfo"))
    .load

println("Read " + customerDataframe.count + " rows")

val orderDetailsDataframe = cassandraSparkSession
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "orderdetails", "keyspace" -> "orderinfo"))
    .load

println("Read " + orderDetailsDataframe.count + " rows")

val orderLineDataframe = cassandraSparkSession
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "orderline", "keyspace" -> "orderinfo"))
    .load

println("Read " + orderLineDataframe.count + " rows")
```

This block of code retrieves the data from the tables in the Cassandra database into Spark DataFrame objects. The code displays the number of rows read from each table.

5.3.5 Task 5: Insert Data into Cosmos DB Tables and Run the Notebook

1. Add a final cell, and enter the following code:

```
// Write the customer data to Cosmos DB

val cosmosDBSparkSession = SparkSession
    .builder()
    .config(cosmosDBConf)
    .getOrCreate()

// Connect to the existing table from Cosmos DB
var customerCopyDataframe = cosmosDBSparkSession
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "customerdetails", "keyspace" -> "customerinfo"))
    .load

// Merge the results from the Cassandra database into the DataFrame
customerCopyDataframe = customerCopyDataframe.union(customerDataframe)

// Write the results back to Cosmos DB
customerCopyDataframe.write
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "customerdetails", "keyspace" -> "customerinfo"))
    .mode(org.apache.spark.sql.SaveMode.Append)
    .save()

// Write the order data to Cosmos DB, using the same strategy
var orderDetailsCopyDataframe = cosmosDBSparkSession
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "orderdetails", "keyspace" -> "orderinfo"))
    .load

orderDetailsCopyDataframe = orderDetailsCopyDataframe.union(orderDetailsDataframe)
```

```
orderDetailsCopyDataframe.write
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "orderdetails", "keyspace" -> "orderinfo"))
    .mode(org.apache.spark.sql.SaveMode.Append)
    .save()

var orderLineCopyDataframe = cosmosDBSparkSession
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "orderline", "keyspace" -> "orderinfo"))
    .load

orderLineCopyDataframe = orderLineCopyDataframe.union(orderLineDataframe)

orderLineCopyDataframe.write
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> "orderline", "keyspace" -> "orderinfo"))
    .mode(org.apache.spark.sql.SaveMode.Append)
    .save()
```

This code creates another DataFrame for each of the tables in the Cosmos DB database. Each DataFrame will be empty, initially. The code then uses the **union** function to append the data from the corresponding DataFrame for each of the Cassandra tables. Finally, the code writes the appended DataFrame back to the Cosmos DB table.

The DataFrame API is a very powerful abstraction provided by Spark, and is a highly efficient structure for transporting large volumes of data very quickly.

2. In the toolbar at the top of the notebook, select **Run All**. You will see messages indicating that the cluster is starting up. When the cluster is ready, the notebook runs the code in each cell in turn. You will see further messages appearing below each cell. The data transfer operations that read and write DataFrames are executed as Spark jobs. You can expand the job to view the progress. The code in each cell should complete successfully, without displaying any error messages.

5.3.6 Task 6: Verify that Data Migration was Successful

1. Return to your Cosmos DB account in the Azure portal.
2. Select **Data Explorer**,
3. In the **Data Explorer** pane, expand the **customerinfo** keyspace, expand the **customerdetails** table, and then select **Rows**. The first 100 rows should be displayed. If the keyspace does not appear in the **Data Explorer** pane, select **Refresh** to update the display.
4. Expand the **orderinfo** keyspace, expand the **orderdetails** table, and then select **Rows**. The first 100 rows should be displayed for this table as well.
5. Finally, expand the **orderline** table, and then select **Rows**. Verify that the first 100 rows for this table appear.

You have successfully migrated a Cassandra database to Cosmos DB by using Spark from a Databricks notebook.

5.3.7 Task 7: Clean Up

1. In the Azure portal, in the left-hand pane, select **Resource groups**.
2. In the **Resource groups** window, select **cassandradbrg**.
3. Select **Delete resource group**.
4. On the **Are you sure you want to delete "cassandradbrg"** page, in the **Type the resource group name** box, enter **cassandradbrg**, and then select **Delete**.
