Contents

1 DP-070-Migrate-Open-Source-Workloads-to-Azure

The following is a summary of the lab objectives for each module:

1.1 Lab 1: Introduction to open source database migration on Azure

There is no lab for this module

1.2 Lab 2: Migrate an on-premises MySQL database to Azure

In this lab, you'll use the information learned in this module to migrate a MySQL database to Azure. To give complete coverage, students will perform two migrations. The first is an offline migration, transferring an on-premises MySQL database to a virtual machine running on Azure. The second is an online migration of the database running on the virtual machine to Azure Database for MySQL.

You'll also reconfigure and run a sample application that uses the database, to verify that the database operates correctly after each migration.

1.3 Lab 3: Migrate an on-premises PostgreSQL database to Azure

In this lab, you'll use the information learned in this module to migrate a PostgreSQL database to Azure. To give complete coverage, students will perform two migrations. The first is an offline migration, transferring an on-premises PostgreSQL database to a virtual machine running on Azure. The second is an online migration of the database running on the virtual machine to Azure Database for PostgreSQL.

You'll also reconfigure and run a sample application that uses the database, to verify that the database operates correctly after each migration.

1.4 Lab 4: Monitoring and tuning a migrated database

In this lab, you'll use the information learned in this module to monitor and tune a database that they previously migrated. You will run a sample application that subjects the database to a read-heavy workload, and monitor the results using the metrics available in Azure. You will use Query Store to examine the performance of queries. You will then configure read replicas, and offload the read processing for some clients to the replicas. You will then examine how this change has affected performance.

2 Lab: Migrate an on-premises MySQL database to Azure

2.1 Overview

In this lab, you'll use the information learned in this module to migrate a MySQL database to Azure. To give complete coverage, students will perform two migrations. The first is an offline migration, transferring an on-premises MySQL database to a virtual machine running on Azure. The second is an online migration of the database running on the virtual machine to Azure Database for MySQL.

You'll also reconfigure and run a sample application that uses the database, to verify that the database operates correctly after each migration.

2.2 Objectives

After completing this lab, you will be able to:

1. Perform an offline migration of an on-premises MySQL database to an Azure virtual machine.
2. Perform an online migration of a MySQL database running on a virtual machine to Azure Database for MySQL.

2.3 Scenario

You work as a database developer for the AdventureWorks organization. AdventureWorks has been selling bicycles and bicycle parts directly to end-consumer and distributors for over a decade. Their systems store information in a database that currently runs using MySQL, located in their on-premises datacenter. As part of a hardware rationalization exercise, AdventureWorks want to move the database to Azure. You have been asked to perform this migration.

Initially, you decide to relocate quickly the data to a MySQL database running on an Azure virtual machine. This is considered to be a low-risk approach as it requires few if any changes to the database. However, this approach does require that you continue to perform most of the day-to-day monitoring and administrative tasks associated with the database. You also need to consider how the customer base of AdventureWorks has changed. Initially AdventureWorks targeted customers in their local region, but now they expanded to be a world-wide operation. Customers can be located anywhere, and ensuring that customers querying the database are subject to minimal latency is a primary concern. You can implement MySQL replication to virtual machines located in other regions, but again this is an administrative overhead.

Instead, once you have got the system running on a virtual machine, you will then consider a more long-term solution; you will migrate the database to Azure Database for MySQL. This PaaS strategy removes much of the work associated with maintaining the system. You can easily scale the system, and add read-only replicas to support customers anywhere in the world. Additionally Microsoft provide a guaranteed SLA for availability.

2.4 Setup

You have an on-premises environment with an existing MySQL database containing the data that you wish to migrate to Azure. Before you start the lab, you need to create an Azure virtual machine running MySQL that will act as the target for the initial migration. We have provided a script that creates this virtual machine and configures it. Use the following steps to download and run this script:

1. Sign in to the **LON-DEV-01** virtual machine running in the classroom environment. The username is **azureuser**, and the password is **Pa55w.rd**.

   This virtual machine simulates your on-premises environment. It is running a MySQL server that is hosting the AdventureWorks database that you need to migrate.

2. Using a browser, sign in to the Azure portal.

3. Open an Azure Cloud Shell window. Make sure that you are running the **Bash** shell.

4. Clone the repository holding the scripts and sample databases if you haven't done this previously.

   ```
   git clone https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure worksh
   ```

5. Move to the *migration_samples/setup* folder.

   ```
   cd ~/workshop/migration_samples/setup
   ```

6. Run the *create_mysql_vm.sh* script as follows. Specify the name of a resource group and a location for holding the virtual machine as parameters. The resource group will be created if it doesn't already exist. Specify a location near to you, such as *eastus* or *uksouth*:

   ```
   bash create_mysql_vm.sh [resource group name] [location]
   ```

   The script will take approximately 10 minutes to run. It will generate plenty of output as it runs, finishing with the IP address of the new virtual machine, and the message **Setup Complete**.

7. Make a note of the IP address.

   [!NOTE] You will need this IP address for the exercise.

2.5 Exercise 1: Migrate the on-premises database to an Azure virtual machine

In this exercise, you'll perform the following tasks:

1. Review the on-premises database.
2. Run a sample application that queries the database.
3. Perform an offline migration of the database to the Azure virtual machine.
4. Verify the database on the Azure virtual machine.
5. Reconfigure and test the sample application against the database in the Azure virtual machine.

2.5.1 Task 1: Review the on-premises database

1. On the **LON-DEV-01** virtual machine running in the classroom environment, in the **Favorites** bar on the left-hand side of the screen, select **MySQLWorkbench**.
2. In the **MySQLWorkbench** window, select **LON-DEV-01**, and select **OK**.
3. Expand **adventureworks**, and then expand **Tables**.
4. Right-click the **contact** table, select **Select Rows - Limit 1000**, and then, in the **contact** query window, select **Limit to 1000 rows** and select **Don't Limit**.
5. Select **Execute** to run the query. It should return 19972 rows.
6. In the list of tables, right-click the **Employee** table, and select **Select rows**.
7. Select **Execute** to run the query. It should return 290 rows.
8. Spend a couple of minutes browsing the data for the other tables in the various tables in the database.

2.5.2 Task 2: Run a sample application that queries the database

1. On the **LON-DEV-01** virtual machine, on the favorites bar, select **Show Applications** and then type **term**.
2. Select **Terminal** to open a terminal window.
3. In the terminal window, download the sample code for the lab. If prompted, enter **Pa55w.rd** for the password:

   ```
   sudo rm -rf ~/workshop
   git clone  https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure ~/wo
   ```

4. Move to the *~/workshop/migration_samples/code/mysql/AdventureWorksQueries* folder:

   ```
   cd ~/workshop/migration_samples/code/mysql/AdventureWorksQueries
   ```

   This folder contains a sample app that runs queries to count the number of rows in several tables in the *adventureworks* database.

5. Run the app:

   ```
   dotnet run
   ```

   The app should generate the following output:

   ```
   Querying AdventureWorks database
   SELECT COUNT(*) FROM product
   504

   SELECT COUNT(*) FROM vendor
   104

   SELECT COUNT(*) FROM specialoffer
   16

   SELECT COUNT(*) FROM salesorderheader
   31465

   SELECT COUNT(*) FROM salesorderdetail
   121317

   SELECT COUNT(*) FROM customer
   19185
   ```

2.5.3 Task 3: Perform an offline migration of the database to the Azure virtual machine

Now that you have an idea of the data in the adventureworks database, you can migrate it to the MySQL server running on the virtual machine in Azure. You'll perform this operation as an offline task, using backup and restore commands.

> [!NOTE] If you wanted to migrate the data online, you could configure replication from the on-premises database to the database running on the Azure virtual machine.

1. From the terminal window, run the following command to take a backup of the *adventureworks* database. Note that the MySQL server on the LON-DEV-01 virtual machine is listening using port 3306:

   ```
   mysqldump -u azureuser -pPa55w.rd adventureworks > aw_mysql_backup.sql
   ```

2. Using the Azure Cloud shell, connect to the virtual machine containing the MySQL server and database. Replace *<nn.nn.nn.nn>* with the IP address of the virtual machine. If you are asked if you want to continue, type **yes** and press Enter.

   ```
   ssh azureuser@nn.nn.nn.nn
   ```

3. Type **Pa55w.rdDemo** and press Enter.

4. Connect to the MySQL server:

   ```
   mysql -u azureuser -pPa55w.rd
   ```

5. Create the target database on the Azure virtual machine:

   ```
   create database adventureworks;
   ```

6. Quit MySQL:

   ```
   quit
   ```

7. Exit the SSH session:

   ```
   exit
   ```

8. Restore the backup into the new database by running this mysql command in the LON-DEV-01 terminal:

   ```
   mysql -h [nn.nn.nn.nn] -u azureuser -pPa55w.rd adventureworks < aw_mysql_backup.sql
   ```

   This command will take a few minutes to run.

2.5.4 Task 4: Verify the database on the Azure virtual machine

1. Run the following command to connect to the database on the Azure virtual machine. The password for the *azureuser* user in the MySQL server running on the virtual machine is **Pa55w.rd**:

   ```
   mysql -h [nn.nn.nn.nn] -u azureuser -pPa55w.rd adventureworks
   ```

2. Run the following query:

   ```
   SELECT COUNT(*) FROM specialoffer;
   ```

   Verify that this query returns 16 rows. This is the same number of rows that is in the on-premises database.

3. Query the number of rows in the *vendor* table.

   ```
   SELECT COUNT(*) FROM vendor;
   ```

   This table should contain 104 rows.

4. Close the *mysql* utility with the **quit** command.

5. Switch to the **MySQL Workbench** tool.

6. On the **Database** menu, select **Manage Connections**, and select **New**.

7. Select the **Connection** tab.

8. In **Connection name** type **MySQL on Azure VM**

9. Enter the following details:

Property	Value
Hostname	*[nn.nn.nn.nn]*
Port	3306
Username	azureuser
Default Schema	adventureworks

10. Select **Test connection**.
11. At the password prompt, type **Pa55w.rd** and select **OK**.
12. Select **OK** and select **Close**.
13. On the **Database** menu, select **Connect to Database**, select **MySQL on Azure VM**, and then select **OK**
14. In **adventureworks** browse the tables in the database. The tables should be the same as those in the on-premises database.

2.5.5 Task 5: Reconfigure and test the sample application against the database on the Azure virtual machine

1. Return to the **terminal** window.
2. Open the App.config file for the test application using the *nano* editor:

   ```
   nano App.config
   ```

3. Change the value of the **ConnectionString** setting and replace **127.0.0.1** with the IP address of the Azure virtual machine. The file should look like this:

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <configuration>
   <appSettings>
       <add key="ConnectionString" value="Server=nn.nn.nn.nn;Database=adventureworks;Uid=azureuser;Pw
   </appSettings>
   </configuration>
   ```

   The application should now connect to the database running on the Azure virtual machine.

4. To save the file and close the editor, press ESC, then press CTRL X. Save your changes when you are prompted, by pressing Y and then Enter.
5. Build and run the application:

   ```
   dotnet run
   ```

   Verify that the application runs successfully, and returns the same number of rows for each table as before.

   You have now migrated your on-premises database to an Azure virtual machine, and reconfigured your application to use the new database.

2.6 Exercise 2: Perform an online migration to Azure Database for MySQL

In this exercise, you'll perform the following tasks:

1. Configure the MySQL server running on the Azure virtual machine and export the schema.
2. Create the Azure Database for MySQL server and database.
3. Import the schema into the target database.
4. Perform an online migration using the Database Migration Service.
5. Modify data, and cutover to the new database.
6. Verify the database in Azure Database for MySQL.
7. Reconfigure and test the sample application against the database in the Azure Database for MySQL.

2.6.1 Task 1: Configure the MySQL server running on the Azure virtual machine and export the schema

1. Using a web browser, return to the Azure portal.

2. Open an Azure Cloud Shell window. Make sure that you are running the **Bash** shell.

3. Connect to the Azure virtual machine running the MySQL server. In the following command, replace *nn.nn.nn.nn* with the IP address of the virtual machine. Enter the password **Pa55w.rdDemo** when prompted:

   ```
   ssh azureuser@nn.nn.nn.nn
   ```

4. Verify that MySQL has started correctly:

   ```
   service mysql status
   ```

   If the service is running, you should see messages similar to the following:

   ```
   mysql.service - MySQL Community Server
     Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2019-09-02 14:45:42 UTC; 21h ago
    Process: 15329 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid (cod
    Process: 15306 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/
   Main PID: 15331 (mysqld)
      Tasks: 30 (limit: 4070)
     CGroup: /system.slice/mysql.service
             15331 /usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid

   Sep 02 14:45:41 mysqlvm systemd[1]: Starting MySQL Community Server...
   Sep 02 14:45:42 mysqlvm systemd[1]: Started MySQL Community Server.
   ```

5. Export the schema for the source database using the mysqldump utility:

   ```
   mysqldump -u azureuser -pPa55w.rd adventureworks --no-data > adventureworks_mysql_schema.sql
   ```

6. At the bash prompt, run the following command to export the **adventureworks** database to a file named **adventureworks_mysql.sql**

   ```
   mysqldump -u azureuser -pPa55w.rd adventureworks > adventureworks_mysql.sql
   ```

7. Disconnect from the virtual machine and return to the Cloud Shell prompt:

   ```
   exit
   ```

8. In the Cloud Shell, copy the schema file from the virtual machine. Replace *nn.nn.nn.nn* with the IP address of the virtual machine. Enter the password **Pa55w.rdDemo** when prompted::

   ```
   scp azureuser@nn.nn.nn.nn:~/adventureworks_mysql_schema.sql adventureworks_mysql_schema.sql
   ```

2.6.2 Task 2. Create the Azure Database for MySQL server and database

1. Switch to the Azure portal.
2. Select **+ Create a resource**.
3. In the **Search the Marketplace** box, type **Azure Database for MySQL**, and press enter.
4. On the **Azure Database for MySQL** page, select **Create**.
5. On the **Select Azure Database for MySQL deployment option** page, under **Single server** select **Create**.
6. On the **Create MySQL server** page, enter the following details:

Property	Value
Subscription	Your subscription
Resource group	Use the same resource group that you specified when you created the Azure virtual machine earlier i
Server name	**adventureworks*nnn***, where *nnn* is a suffix of your choice to make the server name unique
Data source	None
Location	Select your nearest location
Version	5.7
Compute + storage	Select **Configure server**, select the **Basic** pricing tier, and then select **OK**
Admin username	awadmin
Password	Pa55w.rdDemo

Property	Value
Confirm password	Pa55w.rdDemo

7. Select **Review + create**.
8. On the **Review + create** page, select **Create**. Wait for the service to be created before continuing.
9. When the service has been created, select **Go to resource**, and then under **Settings**, and select **Connection security**.
10. On the **Connection security page**, set **Allow access to Azure services** to **Yes**.
11. In the list of firewall rules, add a rule named **VM**, and set the **START IP ADDRESS** and **END IP ADDRESS** to the IP address of the virtual machine running the MySQL server.
12. Select **Add current client IP address**, to enable the **LON-DEV-01** virtual machine acting as the on-premises server to connect to Azure Database for MySQL. You will need this access later, when running the reconfigured client application.
13. Under **SSL settings**, next to **Enforce SSL connection**, select **DISABLED**
14. **Save**, and wait for the firewall rules to be updated.
15. At the Cloud Shell prompt, run the following command to create a new database in your Azure Database for MySQL service. Replace *[nnn]* with the suffix you used when you created the Azure Database for MySQL service. Replace *[resource group]* with the name of the resource group you specified for the service:

    ```
    az MySQL db create \
    --name azureadventureworks \
    --server-name adventureworks[nnn] \
    --resource-group [resource group]
    ```

    If the database is created successfully, you should see a message similar to the following:

    ```
    {
          "charset": "latin1",
          "collation": "latin1_swedish_ci",
          "id": "/subscriptions/nnnnnnnnnnnnnnnnnnnnnnnnnnnnnn/resourceGroups/nnnnnn/providers/Microsof
          "name": "azureadventureworks",
          "resourceGroup": "nnnnn",
          "type": "Microsoft.DBforMySQL/servers/databases"
    }
    ```

2.6.3 Task 3: Import the schema into the target database

1. In the Cloud Shell, run the following command to connect to the azureadventureworks[nnn] server. Replace the two instances of *[nnn]* with the suffix for your service. Note that the username has the *@adventureworks[nnn]* suffix:

    ```
    mysql -h adventureworks[nnn].MySQL.database.azure.com -u awadmin@adventureworks[nnn] -p
    ```

2. At the **Enter password** prompt, enter **Pa55w.rdDemo**.
3. Run the following commands to create a user named *azureuser* and set the password for this user to *Pa55w.rd*. The second statement gives the *azureuser* user the necessary privileges to create objects in the *azureadventureworks* database.

    ```
    GRANT SELECT ON *.* TO 'azureuser'@'localhost' IDENTIFIED BY 'Pa55w.rd';
    GRANT CREATE ON *.* TO 'azureuser'@'localhost';
    ```

4. Run the following commands to create an *adventureworks* database.

    ```
    CREATE DATABASE adventureworks;
    ```

5. Close the *mysql* utility with the **quit** command.
6. Import the **adventureworks** schema to your Azure Database for MySQL service. You are performing the import as *azureuser*, so enter the password **Pa55w.rd** when prompted.

```
mysql -h adventureworks[nnnn].MySQL.database.azure.com -u awadmin@adventureworks[nnn] -pPa55w.rdDe
```

2.6.4 Task 4. Perform an online migration using the Database Migration Service

1. Switch back to the Azure portal.
2. In the menu on the left, select **Subscriptions**, and then select your subscription.
3. On your subscription page, under **Settings**, select **Resource providers**.
4. In the **Filter by name** box, type **DataMigration**, and then select **Microsoft.DataMigration**.
5. If the **Microsoft.DataMigration** isn't registered, select **Register**, and wait for the **Status** to change to **Registered**. It might be necessary to select **Refresh** to see the status change.
6. Select **Create a resource**, in the **Search the Marketplace** box type **Azure Database Migration Service**, and then press Enter.
7. On the **Azure Database Migration Service** page, select **Create**.
8. On the **Create Migration Service** page, enter the following details:

Property	Value
Subscription	Select your own subscription
Select a resource group	Specify the same resource group that you used for the Azure Database for MySQL service and th
Migration service name	adventureworks_migration_service
Location	Select your nearest location
Service mode	Azure
Pricing tier	Premium, with 4 vCores

9. Select **Next: Networking >>**.
10. On the **Networking** page, select the **MySQLvnet/mysqlvmSubnet** virtual network. This network was created as part of the setup.
11. Select **Review + create** and then select **Create**. Wait while the Database Migration Service is created. This will take a few minutes.
12. In the Azure portal, go to the page for your Database Migration Service.
13. Select **New Migration Project**.
14. On the **New migration project** page, enter the following details:

Property	Value
Project name	adventureworks_migration_project
Source server type	MySQL
Target Database for MySQL	Azure Database for MySQL
Choose type of activity	Online data migration

15. Select **Create and run activity**.
16. When the **Migration Wizard** starts, on the **Select source** page, enter the following details:

Property	Value
Source server name	nn.nn.nn.nn *(The IP address of the Azure virtual machine running MySQL)*
Server port	3306
User Name	azureuser
Password	Pa55w.rd

17. Select **Next: Select target>>**.
18. On the **Select target** page, enter the following details:

Property	Value
Target server name	adventureworks[nnn].MySQL.database.azure.com
User Name	awadmin@adventureworks[nnn]
Password	Pa55w.rdDemo

19. Select **Next: Select databases>>**.
20. On the **Select databases** page, ensure that both the **Source Database** and the **Target Database** are set to **adventureworks** and then select **Next: Configure migration settings**.
21. On the **Configure migration settings** page, select **Next: Summary>>**.
22. On the **Migration summary** page, in the **Activity name** box type **AdventureWorks_Migration_Activity**, and then select **Start migration**.
23. On the **AdventureWorks_Migration_Activity** page, select **Refresh** at 15 second intervals. You will see the status of the migration operation as it progresses. Wait until the **MIGRATION DETAILS** column changes to **Ready to cutover**.

2.6.5 Task 5. Modify data, and cutover to the new database

1. Return to the **AdventureWorks_Migration_Activity** page in the Azure portal.
2. Select the **adventureworks** database.
3. On the **adventureworks** page, verify that the status for all tables is marked as **COMPLETED**.
4. Select **Incremental data sync**. Verify that the status for every table is marked as **Syncing**.
5. Switch back to the Cloud Shell.
6. Run the following command to connect to the **adventureworks** database running using MySQL on the virtual machine:

   ```
   mysql -h nn.nn.nn.nn -u azureuser -pPa55w.rd adventureworks
   ```

7. Execute the following SQL statements to display, and then remove orders 43659, 43660, and 43661 from the database. Note that the database implements a cascading delete on the *salesorderheader* table, which automatically deletes the corresponding rows from the *salesorderdetail* table.

   ```
   SELECT * FROM salesorderheader WHERE salesorderid IN (43659, 43660, 43661);
   SELECT * FROM salesorderdetail WHERE salesorderid IN (43659, 43660, 43661);
   DELETE FROM salesorderheader WHERE salesorderid IN (43659, 43660, 43661);
   ```

8. Close the *mysql* utility with the **quit** command.
9. Return to the **adventureworks** page in the Azure portal, and then select **Refresh**. Scroll to the page for the *salesorderheader* and *salesorderdetail* tables. Verify that the *salesorderheader* table indicates that 3 rows have been deleted, and 29 rows have been removed from the **sales.salesorderdetail** table. If there are no updates applied, check that there are **Pending changes** for the database.
10. Select **Start cutover**.
11. On the **Complete cutover** page, select **Confirm**, and then select **Apply**. Wait until the status changes to **Completed**.
12. Return to the Cloud Shell.
13. Run the following command to connect to the **azureadventureworks** database running using your Azure Database for MySQL service:

    ```
    mysql -h adventureworks[nnn].MySQL.database.azure.com -u awadmin@adventureworks[nnn] -pPa55w.rdDem
    ```

14. Run the following SQL statements to display the orders and details for orders 43659, 43660, and 43661. The purpose of these queries is to show that the data has been transferred:

    ```
    SELECT * FROM salesorderheader WHERE salesorderid IN (43659, 43660, 43661);
    SELECT * FROM salesorderdetail WHERE salesorderid IN (43659, 43660, 43661);
    ```

    The first query should return 3 rows. The second query should return 29 rows.

15. Close the *mysql* utility with the **quit** command.

2.6.6 Task 6: Verify the database in Azure Database for MySQL

1. Return to the virtual machine acting as your on-premises computer
2. Switch to the **MySQL Workbench** tool.
3. On the **Database** menu, select **Connect to database**
4. Enter the following details:

Property	Value
Hostname	adventureworks*[nnn]*.MySQL.database.azure.com
Port	3306
Username	awadmin@adventureworks*[nnn]*
Password	Pa55w.rdDemo

5. Select **OK**.
6. Expand **Databases**, expand **adventureworks**, and then browse the tables in the database. The tables should be the same as those in the on-premises database.

2.6.7 Task 7: Reconfigure and test the sample application against the database in the Azure Database for MySQL

1. Return to the **terminal** window on the **LON-DEV-01** virtual machine.
2. Move to the *workshop/migration_samples/code/mysql/AdventureWorksQueries* folder:

   ```
   cd ~/workshop/migration_samples/code/mysql/AdventureWorksQueries
   ```

3. Open the App.config file using the nano editor:

   ```
   nano App.config
   ```

4. Change the value of the **ConnectionString** setting and replace the IP address of the Azure virtual machine to **adventureworks[nnn].MySQL.database.azure.com**. Change the **User Id** to **awadmin@adventureworks[nnn]**. Change the **Password** to **Pa55w.rdDemo**. The file should look like this:

   ```
   <?xml version="1.0" encoding="utf-8" ?>
       <configuration>
         <appSettings>
           <add key="ConnectionString" value="Server=adventureworks[nnn].MySQL.database.azure.com;data
         </appSettings>
       </configuration>
   ```

   The application should now connect to the database running on Azure Database for MySQL.

5. Save the file and close the editor.
6. Build and run the application:

   ```
   dotnet run
   ```

   The app should display the same results as before, except that it is now retrieving the data from the database running in Azure.

   You have now migrated your database to Azure Database for MySQL, and reconfigured your application to use the new database.

3 Lab: Migrate an on-premises PostgreSQL database to Azure

3.1 Overview

In this lab, you'll use the information learned in this module to migrate a PostgreSQL database to Azure. To give complete coverage, students will perform two migrations. The first is an offline migration, transferring an

on-premises PostgreSQL database to a virtual machine running on Azure. The second is an online migration of the database running on the virtual machine to Azure Database for PostgreSQL.

You'll also reconfigure and run a sample application that uses the database, to verify that the database operates correctly after each migration.

3.2 Objectives

After completing this lab, you will be able to:

1. Perform an offline migration of an on-premises PostgreSQL database to an Azure virtual machine.
2. Perform an online migration of a PostgreSQL database running on a virtual machine to Azure Database for PostgreSQL.

3.3 Scenario

You work as a database developer for the AdventureWorks organization. AdventureWorks has been selling bicycles and bicycle parts directly to end-consumer and distributors for over a decade. Their systems store information in a database that currently runs using PostgreSQL, located in their on-premises datacenter. As part of a hardware rationalization exercise, AdventureWorks want to move the database to Azure. You have been asked to perform this migration.

Initially, you decide to relocate the data quickly to a PostgreSQL database running on an Azure virtual machine. This is considered to be a low-risk approach as it requires few if any changes to the database. However, this approach does require that you continue to perform most of the day-to-day monitoring and administrative tasks associated with the database. You also need to consider how the customer base of AdventureWorks has changed. Initially AdventureWorks targeted customers in their local region, but now they expanded to be a world-wide operation. Customers can be located anywhere, and ensuring that customers querying the database are subject to minimal latency is a primary concern. You can implement PostgreSQL replication to virtual machines located in other regions, but again this is an administrative overhead.

Instead, once you have got the system running on a virtual machine, you will then consider a more long-term solution; you will migrate the database to Azure Database for PostgreSQL. This PaaS strategy removes much of the work associated with maintaining the system. You can easily scale the system, and add read-only replicas to support customers anywhere in the world. Additionally Microsoft provide a guaranteed SLA for availability.

3.4 Setup

You have an on-premises environment with an existing PostgreSQL database containing the data that you wish to migrate to Azure. Before you start the lab, you need to create an Azure virtual machine running PostgreSQL that will act as the target for the initial migration. We have provided a script that creates this virtual machine and configures it. Use the following steps to download and run this script:

1. Sign in to the **LON-DEV-01** virtual machine running in the classroom environment. The username is **azureuser**, and the password is **Pa55w.rd**.
2. Using a browser, sign in to the Azure portal.
3. Open an Azure Cloud Shell window. Make sure that you are running the **Bash** shell.
4. Clone the repository holding the scripts and sample databases if you haven't done this previously.

   ```
   git clone https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure ~/wor
   ```

5. Move to the *workshop/migration_samples/setup* folder.

   ```
   cd ~/workshop/migration_samples/setup
   ```

6. Run the *create_postgresql_vm.sh* script as follows. Specify the name of a resource group and a location for holding the virtual machine as parameters. The resource group will be created if it doesn't already exist. Specify a location near to you, such as *eastus* or *uksouth*:

   ```
   bash create_postgresql_vm.sh [resource group name] [location]
   ```

   The script will take approximately 10 minutes to run. It will generate plenty of output as it runs, finishing with the IP address of the new virtual machine, and the message **Setup Complete**.

7. Make a note of the IP address.

[!NOTE] You will need this IP address for the exercise.

3.5 Exercise 1: Migrate the on-premises database to an Azure virtual machine

In this exercise, you'll perform the following tasks:

1. Review the on-premises database.
2. Run a sample application that queries the database.
3. Perform an offline migration of the database to the Azure virtual machine.
4. Verify the database on the Azure virtual machine.
5. Reconfigure and test the sample application against the database in the Azure virtual machine.

3.5.1 Task 1: Review the on-premises database

1. Sign in to the **LON-DEV-01** virtual machine running in the classroom environment. The username is **azureuser**, and the password is **Pa55w.rd**.

   This virtual machine simulates your on-premises environment. It is running a PostgreSQL server that is hosting the AdventureWorks database that you need to migrate.

2. On the **LON-DEV-01** virtual machine running in the classroom environment, in the **Favorites** bar on the left-hand side of the screen, select the **pgAdmin4** utility.

3. In the **Unlock Saved Password** dialog box, enter the password **Pa55w.rd**, and then select **OK**.

4. In the **pgAdmin4** window, expand **Servers**, expand **LON-DEV-01**, expand **Databases**, expand **adventureworks**, and then expand **Schemas**.

5. In the **sales** schema, expand **Tables**.

6. Right-click the **salesorderheader** table, select **Scripts**, and then select **SELECT Script**.

7. Press **F5** to run the query. It should return 31465 rows.

8. In the list of tables, right-click the **salesorderdetail** table, select **Scripts**, and then select **Select Script**.

9. Press **F5** to run the query. It should return 121317 rows.

10. Spend a couple of minutes browsing the data for the other tables in the various schemas in the database.

3.5.2 Task 2: Run a sample application that queries the database

1. On the **LON-DEV-01** virtual machine, on the favorites bar, select **Show Applications** and then type **term**.

2. Select **Terminal** to open a terminal window.

3. In the terminal window, download the sample code for the demonstration. If prompted, enter **Pa55w.rd** for the password:

   ```
   sudo rm -rf ~/workshop
   git clone https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure ~/wor
   ```

4. Move to the *~/workshop/migration_samples/code/postgresql/AdventureWorksQueries* folder:

   ```
   cd ~/workshop/migration_samples/code/postgresql/AdventureWorksQueries
   ```

   This folder contains a sample app that runs queries to count the number of rows in several tables in the *adventureworks* database.

5. Run the app:

   ```
   dotnet run
   ```

   The app should generate the following output:

   ```
   Querying AdventureWorks database
   SELECT COUNT(*) FROM production.vproductanddescription
   1764

   SELECT COUNT(*) FROM purchasing.vendor
   104
   ```

```
SELECT COUNT(*) FROM sales.specialoffer
16

SELECT COUNT(*) FROM sales.salesorderheader
31465

SELECT COUNT(*) FROM sales.salesorderdetail
121317

SELECT COUNT(*) FROM person.person
19972
```

3.5.3 Task 3: Perform an offline migration of the database to the Azure virtual machine

Now that you have an idea of the data in the adventureworks database, you can migrate it to the PostgreSQL server running on the virtual machine in Azure. You'll perform this operation as an offline task, using backup and restore commands.

> [!NOTE] If you wanted to migrate the data online, you could configure replication from the on-premises database to the database running on the Azure virtual machine.

1. From the terminal window, run the following command to take a backup of the *adventureworks* database:

   ```
   pg_dump adventureworks -U azureuser -Fc > adventureworks_backup.bak
   ```

2. Create the target database on the Azure virtual machine. Replace [nn.nn.nn.nn] with the IP address of the virtual machine that was created during the setup stage of this lab. Enter the password **Pa55w.rd** when prompted for the password for **azureuser**:

   ```
   createdb -h [nn.nn.nn.nn] -U azureuser --password adventureworks
   ```

3. Restore the backup into the new database with the pg_restore command:

   ```
   pg_restore -d adventureworks -h [nn.nn.nn.nn] -Fc -U azureuser --password adventureworks_backup.ba
   ```

   This command will take a couple of minutes to run.

3.5.4 Task 4: Verify the database on the Azure virtual machine

1. Run the following command to connect to the database on the Azure virtual machine. The password for the *azureuser* user in the PostgreSQL server running on the virtual machine is **Pa55w.rd**:

   ```
   psql -h [nn.nn.nn.nn] -U azureuser adventureworks
   ```

2. Run the following query:

   ```
   SELECT COUNT(*) FROM sales.salesorderheader;
   ```

   Verify that this query returns 31465 rows. This is the same number of rows that is in the on-premises database.

3. Query the number of rows in the *sales.salesorderdetail* table.

   ```
   SELECT COUNT(*) FROM sales.salesorderdetail;
   ```

   This table should contain 121317 rows.

4. Close the *psql* utility with the **\q** command.
5. Switch to the **pgAdmin4** tool.
6. In the left-pane, right-click **Servers**, select **Create**, and then select **Server**.
7. In the **Create - Server** dialog box, on the **General** tab, in the **Name** box, enter **Virtual Machine**, and then select the **Connection** tab.
8. Enter the following details:

Property	Value
Host name/address	*[nn.nn.nn.nn]*
Port	5432
Maintenance database	postgres
Username	azureuser
Password	Pa55w.rd
Save password	Selected
Role	*leave blank*
Service	*Leave blank*

9. Select **Save**.
10. In the left pane of the **pgAdmin4** window, under **Servers**, expand **Virtual Machine**.
11. Expand **Databases**, expand **adventureworks**, and then browse the schemas and tables in the database. The tables should be the same as those in the on-premises database.

3.5.5 Task 5: Reconfigure and test the sample application against the database on the Azure virtual machine

1. Return to the **terminal** window.
2. Open the App.config file for the test application using the *nano* editor.

   ```
   nano App.config
   ```

3. Change the value of the **ConnectionString** setting and replace **localhost** with the IP address of the Azure virtual machine. The file should look like this:

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <configuration>
     <appSettings>
       <add key="ConnectionString" value="Server=nn.nn.nn.nn;Database=adventureworks;Port=5432;User I
     </appSettings>
   </configuration>
   ```

   The application should now connect to the database running on the Azure virtual machine.

4. To save the file and close the editor, press ESC, then press CTRL X. Save your changes when you are prompted, by pressing Y and then Enter.
5. Build and run the application:

   ```
   dotnet run
   ```

   Verify that the application runs successfully, and returns the same number of rows for each table as before.

   You have now migrated your on-premises database to an Azure virtual machine, and reconfigured your application to use the new database.

3.6 Exercise 2: Perform an online migration to Azure Database for PostgreSQL

In this exercise, you'll perform the following tasks:

1. Configure the PostgreSQL server running on the Azure virtual machine and export the schema.
2. Create the Azure Database for PostgreSQL server and database.
3. Import the schema into the target database.
4. Perform an online migration using the Database Migration Service
5. Modify data, and cut over to the new database
6. Verify the database in Azure Database for PostgreSQL
7. Reconfigure and test the sample application against the database in the Azure Database for PostgreSQL.

3.6.1 Task 1: Configure the PostgreSQL server running on the Azure virtual machine and export the schema

1. Using a web browser, return to the Azure portal.

2. Open an Azure Cloud Shell window. Make sure that you are running the **Bash** shell.
3. Clone the repository holding the scripts and sample databases if you haven't done this previously.

   ```
   git clone https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure ~/work
   ```

4. Move to the *~/workshop/migration_samples/setup/postgresql/adventureworks* folder.

   ```
   cd ~/workshop/migration_samples/setup/postgresql/adventureworks
   ```

5. Connect to the Azure virtual machine running the PostgreSQL server. In the following command, replace *nn.nn.nn.nn* with the IP address of the virtual machine. Enter the password **Pa55w.rdDemo** when prompted:

   ```
   ssh azureuser@nn.nn.nn.nn
   ```

6. On the virtual machine, switch to the *root* account. Enter the password for the *azureuser* user if prompted (**Pa55w.rdDemo**).

   ```
   sudo bash
   ```

7. Move to the directory */etc/postgresql/10/main*:

   ```
   cd /etc/postgresql/10/main
   ```

8. Using the *nano* editor, open the *postgresql.conf* file.

   ```
   nano postgresql.conf
   ```

9. Scroll to the bottom of the file, and verify that the following parameters have been configured:

   ```
   listen_addresses = '*'
   wal_level = logical
   max_replication_slots = 5
   max_wal_senders = 10
   ```

10. To save the file and close the editor, press ESC, then press CTRL X. Save your changes, if you are prompted to, by pressing Enter.
11. Restart the PostgreSQL service:

    ```
    service postgresql restart
    ```

12. Verify that PostgreSQL has started correctly:

    ```
    service postgresql status
    ```

    If the service is running, you should see messages similar to the following:

    ```
     postgresql.service - PostgreSQL RDBMS
      Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
       Active: active (exited) since Fri 2019-08-23 12:47:02 UTC; 2min 3s ago
       Process: 115562 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
      Main PID: 115562 (code=exited, status=0/SUCCESS)

    Aug 23 12:47:02 postgreSQLVM systemd[1]: Starting PostgreSQL RDBMS...
    Aug 23 12:47:02 postgreSQLVM systemd[1]: Started PostgreSQL RDBMS.
    ```

13. Leave the *root* account and return to the *azureuser* account:

    ```
    exit
    ```

14. Run the following command to connect to the database on the Azure virtual machine. The password for the *azureuser* user in the PostgreSQL server running on the virtual machine is **Pa55w.rd**:

    ```
    psql -h [nn.nn.nn.nn] -U azureuser adventureworks
    ```

15. Grant replication permission to azureuser:

    ```
    ALTER ROLE azureuser REPLICATION;
    ```

16. At the bash prompt, run the following command to export the schema for the **adventureworks** database to a file named **adventureworks_schema.sql**

```
pg_dump -o  -d adventureworks -s > adventureworks_schema.sql
```

17. Disconnect from the virtual machine and return to the Cloud Shell prompt:

```
exit
```

18. In the Cloud Shell, copy the schema file from the virtual machine. Replace *nn.nn.nn.nn* with the IP address of the virtual machine. Enter the password **Pa55w.rdDemo** when prompted::

```
scp azureuser@nn.nn.nn.nn:~/adventureworks_schema.sql adventureworks_schema.sql
```

3.7 Task 2. Create the Azure Database for PostgreSQL server and database

1. Switch to the Azure portal.
2. Select **+ Create a resource**.
3. In the **Search the Marketplace** box, type **Azure Database for PostgreSQL**, and press enter.
4. On the **Azure Database for PostgreSQL** page, select **Create**.
5. On the **Select Azure Database for PostgreSQL deployment option** page, in the **Single server** box, select **Create**.
6. On the **Single server** page, enter the following details:

Property	Value
Subscription	Select your subscription
Resource group	Use the same resource group that you specified when you created the Azure virtual machine earlier i
Server name	**adventureworks*nnn***, where *nnn* is a suffix of your choice to make the server name unique
Data source	None
Location	Select your nearest location
Version	10
Compute + storage	Select **Configure server**, select the **Basic** pricing tier, and then select **OK**
Admin username	awadmin
Password	Pa55w.rdDemo
Confirm password	Pa55w.rdDemo

7. Select **Review + create**.
8. On the **Review + create** page, select **Create**. Wait for the service to be created before continuing.
9. When the service has been created, go to the page for the service in the portal, and select **Connection security**.
10. On the **Connection security page**, set **Allow access to Azure services** to **Yes**.
11. In the list of firewall rules, add a rule named **VM**, and set the **START IP ADDRESS** and **END IP ADDRESS** to the IP address of the virtual machine running the PostgreSQL server you created earlier.
12. Select **Add current client IP address**, to enable the **LON-DEV-01** virtual machine acting as the on-premises server to connect to Azure Database for PostgreSQL. You will need this access later, when running the reconfigured client application.
13. **Save**, and wait for the firewall rules to be updated.
14. At the Cloud Shell prompt, run the following command to create a new database in your Azure Database for PostgreSQL service. Replace *[nnn]* with the suffix you used when you created the Azure Database for PostgreSQL service. Replace *[resource group]* with the name of the resource group you specified for the service:

```
az postgres db create \
  --name azureadventureworks \
  --server-name adventureworks[nnn] \
  --resource-group [resource group]
```

If the database is created successfully, you should see a message similar to the following:

```
{
  "charset": "UTF8",
  "collation": "English_United States.1252",
  "id": "/subscriptions/nnnnnnnnnnnnnnnnnnnnnn/resourceGroups/nnnnnnnn/providers/Microsoft.DBforPo
  "name": "azureadventureworks",
  "resourceGroup": "nnnnnnnn",
  "type": "Microsoft.DBforPostgreSQL/servers/databases"
}
```

3.7.1 Task 3: Import the schema into the target database

1. In the Cloud Shell, run the following command to connect to the azureadventureworks[nnn] server. Replace the two instances of *[nnn]* with the suffix for your service. Note that the username has the *@adventureworks[nnn]* suffix. At the password prompt, enter **Pa55w.rdDemo**.

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U awadmin@adventureworks[nnn] -d postgres
   ```

2. Run the following commands to create a user named *azureuser* and set the password for this user to *Pa55w.rd*. The third statement gives the *azureuser* user the necessary privileges to create and manage objects in the *azureadventureworks* database. The *azure_pg_admin* role enables the *azureuser* user to install and use extensions in the database.

   ```
   CREATE ROLE azureuser WITH LOGIN;
   ALTER ROLE azureuser PASSWORD 'Pa55w.rd';
   GRANT ALL PRIVILEGES ON DATABASE azureadventureworks TO azureuser;
   GRANT azure_pg_admin TO azureuser;
   ```

3. Close the *psql* utility with the **\q** command.

4. Import the schema for the **adventureworks** database to the **azureadventureworks** database running on your Azure Database for PostgreSQL service. You are performing the import as *azureuser*, so enter the password **Pa55w.rd** when prompted.

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
   ```

   You will see a series of messages as each item is created. The script should complete without any errors.

5. Run the following command. The *findkeys.sql* script generates another SQL script named *dropkeys.sql* that will remove all the foreign keys from the tables in the **azureadventureworks** database. You will run the *dropkeys.sql* script shortly:

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
   ```

   You can examine the *dropkeys.sql* script using a text editor if you have time.

6. Run the following command. The *createkeys.sql* script generates another SQL script named *addkeys.sql* that will recreate all the foreign keys. You will run the *addkeys.sql* script after you have migrated the database:

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
   ```

7. Run the *dropkeys.sql* script:

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
   ```

   You will see a series **ALTER TABLE** messages displayed, as the foreign keys are dropped.

8. Stat the psql utility again and connect to the *azureadventureworks* database.

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
   ```

9. Run the following query to find the details of any remaining foreign keys:

   ```
   SELECT constraint_type, table_schema, table_name, constraint_name
   FROM information_schema.table_constraints
   WHERE constraint_type = 'FOREIGN KEY';
   ```

   This query should return an empty result set. However, if any foreign keys still exist, for each foreign key, run the following command:

   ```
   ALTER TABLE [table_schema].[table_name] DROP CONSTRAINT [constraint_name];
   ```

10. After you have removed any remaining foreign keys, execute the following SQL statement to display the triggers in the database:

    ```
    SELECT trigger_name
    FROM information_schema.triggers;
    ```

    This query should also return an empty result set, indicating that the database contains no triggers. If the database did contain triggers, you would have to disable them before migrating the data, and re-enable them afterwards.

11. Close the *psql* utility with the **\q** command.

3.8 Task 4. Perform an online migration using the Database Migration Service

1. Switch back to the Azure portal.
2. Select **All services**, select **Subscriptions**, and then select your subscription.
3. On your subscription page, under **Settings**, select **Resource providers**.
4. In the **Filter by name** box, type **DataMigration**, and then select **Microsoft.DataMigration**.
5. If the **Microsoft.DataMigration** isn't registered, select **Register**, and wait for the **Status** to change to **Registered**. It might be necessary to select **Refresh** to see the status change.
6. Select **Create a resource**, in the **Search the Marketplace** box type **Azure Database Migration Service**, and then press Enter.
7. On the **Azure Database Migration Service** page, select **Create**.
8. On the **Create Migration Service** page, enter the following details:

Property	Value
Subscription	Select your own subscription
Select a resource group	Specify the same resource group that you used for the Azure Database for PostgreSQL service an
Service name	adventureworks_migration_service
Location	Select your nearest location
Service mode	Azure
Pricing tier	Premium, with 4 vCores

9. Select **Next: Networking>>**.
10. On the **Networking** page, select the **postgresqlvnet/posgresqlvmSubnet** virtual network. This network was created as part of the setup.
11. Select **Review + create** and then select **Create**. Wait while the Database Migration Service is created. This will take a few minutes.
12. In the Azure portal, go to the page for your Database Migration Service.
13. Select **New Migration Project**.
14. On the **New migration project** page, enter the following details:

Property	Value
Project name	adventureworks_migration_project
Source server type	PostgreSQL
Target Database for PostgreSQL	Azure Database for PostgreSQL
Choose type of activity	Online data migration

15. Select **Create and run activity**.
16. When the **Migration Wizard** starts, on the **Select source** page, enter the following details:

Property	Value
Source server name	nn.nn.nn.nn *(The IP address of the Azure virtual machine running PostgreSQL)*
Server port	5432
Database	adventureworks
User Name	azureuser
Password	Pa55w.rd
Trust server certificate	Selected
Encrypt connection	Selected

17. Select **Next: Select target>>**.
18. On the **Select target** page, enter the following details:

Property	Value
Subscription	Select your subscription
Azure PostgreSQL	adventureworks[nnn]
Database	azureadventureworks
User Name	azureuser@adventureworks[nnn]
Password	Pa55w.rd

19. Select **Next: Select databases>>**.
20. On the **Select databases** page, select the **adventureworks** database and map it to **azureadventureworks**. Deselect the **postgres** database. Select **Next: Select tables>>**.
21. On the **Select tables** page, select **Next: Configure migration settings>>**.
22. On the **Configure migration settings** page, expand the **adventureworks** dropdown, expand the **Advanced online migration settings dropdown**, verify that **Maximum number of instances to load in parallel** is set to 5, and then select **Next: Summary>>**.
23. On the **Summary** page, in the **Activity name** box type **AdventureWorks_Migration_Activity**, and then select **Start migration**.
24. On the **AdventureWorks_Migration_Activity** page, select **Refresh** at 15 second intervals. You will see the status of the migration operation as it progresses. Wait until the **MIGRATION DETAILS** column changes to **Ready to cutover**.
25. Switch back to the Cloud Shell.
26. Run the following command to recreate the foreign keys in the **azureadventureworks** database. You generated the **addkeys.sql** script earlier:

    ```
    psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
    ```

    You will see a series of **ALTER TABLE** statements as the foreign keys are added. You may see an error concerning the *SpecialOfferProduct* table, which you can ignore for now. This is due to a UNIQUE constraint that doesn't get transferred correctly. In the real world, you should retrieve the details of this constraint from the source database using the following query:

    ```
    SELECT constraint_type, table_schema, table_name, constraint_name
    FROM information_schema.table_constraints
    WHERE constraint_type = 'UNIQUE';
    ```

    You could then manually reinstate this constraint in the target database in Azure Database for PostgreSQL.

    There should be no other errors.

3.9 Task 5. Modify data, and cut over to the new database

1. Return to the **AdventureWorks_Migration_Activity** page in the Azure portal.
2. Select the **adventureworks** database.

3. On the **adventureworks** page, verify that the **Full load completed** value is **66** and that all other values are **0**.
4. Switch back to the Cloud Shell.
5. Run the following command to connect to the **adventureworks** database running using PostgreSQL on the virtual machine:

   ```
   psql -h nn.nn.nn.nn -U azureuser -d adventureworks
   ```

6. Execute the following SQL statements to display, and then remove orders 43659, 43660, and 43661 from the database. Note that the database implements a cascading delete on the *salesorderheader* table, which automatically deletes the corresponding rows from the *salesorderdetail* table.

   ```
   SELECT * FROM sales.salesorderheader WHERE salesorderid IN (43659, 43660, 43661);
   SELECT * FROM sales.salesorderdetail WHERE salesorderid IN (43659, 43660, 43661);
   DELETE FROM sales.salesorderheader WHERE salesorderid IN (43659, 43660, 43661);
   ```

7. Close the *psql* utility with the **\q** command.
8. Return to the **adventureworks** page in the Azure portal, and select **Refresh**. Verify that 32 changes have been applied.
9. Select **Start Cutover**.
10. On the **Complete cutover** page, select **Confirm**, and then select **Apply**. Wait until the status changes to **Completed**.
11. Return to the Cloud Shell.
12. Run the following command to connect to the **azureadventureworks** database running using your Azure Database for PostgreSQL service:

    ```
    psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
    ```

13. Execute the following SQL statements to display the orders and order details in the database. Quit after the first page of each table. The purpose of these queries is to show that the data has been transferred:

    ```
    SELECT * FROM sales.salesorderheader;
    SELECT * FROM sales.salesorderdetail;
    ```

14. Run the following SQL statements to display the orders and details for orders 43659, 43660, and 43661.

    ```
    SELECT * FROM sales.salesorderheader WHERE salesorderid IN (43659, 43660, 43661);
    SELECT * FROM sales.salesorderdetail WHERE salesorderid IN (43659, 43660, 43661);
    ```

    Both queries should return 0 rows.
15. Close the *psql* utility with the **\q** command.

3.9.1 Task 6: Verify the database in Azure Database for PostgreSQL

1. Return to the virtual machine acting as your on-premises computer
2. Switch to the **pgAdmin4** tool.
3. In the left-pane, right-click **Servers**, select **Create**, and then select **Server**.
4. In the **Create - Server** dialog box, on the **General** tab, in the **Name** box, enter **Azure Database for PostgreSQL**, and then select the **Connection** tab.
5. Enter the following details:

Property	Value
Host name/address	adventureworks*[nnn]*.postgres.database.azure.com
Port	5432
Maintenance database	postgres
Username	azureuser@adventureworks*[nnn]*
Password	Pa55w.rd
Save password	Selected
Role	*leave blank*

Property	Value
Service	*Leave blank*

6. Select the **SSL** tab.
7. On the **SSL** tab, set **SSL mode** to **Require**, and then select **Save**.
8. In the left pane of the **pgAdmin4** window, under **Servers**, expand **Azure Database for PostgreSQL**.
9. Expand **Databases**, expand **azureadventureworks**, and then browse the schemas and tables in the database. The tables should be the same as those in the on-premises database.

3.9.2 Task 7: Reconfigure and test the sample application against the database in the Azure Database for PostgreSQL

1. Return to the **terminal** window.
2. Open the App.config file for the test application using the *nano* editor.

   ```
   nano App.config
   ```

3. Change the value of the **ConnectionString** setting and replace the IP address of the Azure virtual machine to **adventureworks[nnn].postgres.database.azure.com**. Change the **Database** to **azureadventureworks**. Change the **User Id** to **azureuser@adventureworks[nnn]**. Append the text **Ssl Mode=Require;** to the end of the connection string. The file should look like this:

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <configuration>
     <appSettings>
       <add key="ConnectionString" value="Server=adventureworks[nnn].postgres.database.azure.com;Datal
     </appSettings>
   </configuration>
   ```

   The application should now connect to the database running on the Azure virtual machine.

4. To save the file and close the editor, press ESC, then press CTRL X. Save your changes when you are prompted, by pressing Y and then Enter.
5. Build and run the application:

   ```
   dotnet run
   ```

   The application will connect to the database, but will fail with the message **materialized view "vproductanddescription" has not been populated**. You need to refresh the materialized views in the database.

6. In the terminal window, use the *psql* utility to connect to the azureadventureworks database:

   ```
   psql -h adventureworks[nnn].postgres.database.azure.com -U azureuser@adventureworks[nnn] -d azurea
   ```

7. Run the following query to list all the materialized views in the database:

   ```
   SELECT schemaname, matviewname, matviewowner, ispopulated
   FROM pg_matviews;
   ```

   This query should return two rows, for *person.vstateprovincecountryregion*, and *production.vproductanddescription*. The *ispopulated* column for both views is *f*, indicating that they haven't yet been populated.

8. Run the following statements to populate both views:

   ```
   REFRESH MATERIALIZED VIEW person.vstateprovincecountryregion;
   REFRESH MATERIALIZED VIEW production.vproductanddescription;
   ```

9. Close the *psql* utility with the **\q** command.
10. Run the application again:

    ```
    dotnet run
    ```

Verify that the application now runs successfully.

You have now migrated your database to Azure Database for PostgreSQL, and reconfigured your application to use the new database.

4 Lab: Monitoring and tuning a migrated database

4.1 Overview

In this lab, you'll use the information learned in this module to monitor and tune a database that you previously migrated. You will run a sample application that subjects the database to a read-heavy workload, and monitor the results using the metrics available in Azure. You will use Query Store to examine the performance of queries. You will then configure read replicas, and offload the read processing for some clients to the replicas. You will then examine how this change has affected performance.

4.2 Objectives

After completing this lab, you will be able to:

1. Use Azure metrics to monitor performance.
2. Use Query Store to examine the performance of queries.
3. Offload read operations for some users to use a read replica.
4. Monitor performance again and assess the results.

4.3 Scenario

You work as a database developer for the AdventureWorks organization. AdventureWorks has been selling bicycles and bicycle parts directly to end-consumers and distributors for over a decade. Their systems store information in a database that you have previously migrated to Azure Database for PostgreSQL.

After having performed the migration, you want assurance that the system is performing well. You decide to use the Azure tools available to monitor the server. To alleviate the possibility of slow response times caused by contention and latency, you decide to implement read replication. You need to monitor the resulting system and compare the results with the single server architecture.

4.4 Setup

This lab assumes that you have completed the lab from module 3; it uses the Azure Database for PostgreSQL server and database created by that lab. If you haven't completed that lab, perform the setup steps below. If you have a working Azure Database for PostgreSQL server, and have successfully migrated the AdventureWorks database to this server, proceed directly to set Exercise 1.

1. Sign in to the **LON-DEV-01** virtual machine running in the classroom environment. The username is **azureuser**, and the password is **Pa55w.rd**.
2. Using a browser, sign in to the Azure portal.
3. Open an Azure Cloud Shell window. Make sure that you are running the **Bash** shell.
4. In the Cloud Shell clone the repository holding the scripts and sample databases, if you haven't done this previously.

   ```
   git clone https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure ~/wor
   ```

5. Move to the *workshop/migration_samples/setup/postgresql* folder.

   ```
   cd ~/workshop/migration_samples/setup/postgresql
   ```

6. Run the following command. Replace *[nnn]* with a number that ensures that the Azure Database for PostgreSQL server created by the script has a unique name. The server is created in the Azure resource group that you specify as the second parameter to the script. This resource group will be created if it doesn't already exist. Optionally, specify the location for the resource group as the third parameter. This location defaults to "westus" if the parameter is omitted:

   ```
   bash copy_adventureworks.sh adventureworks[nnn] [resource_group_name] [location]
   ```

7. When prompted for the *awadmin* password, type **Pa55w.rdDemo**.

8. When prompted for the *azureuser* password, type **Pa55w.rd**.
9. Wait for the script to complete before continuing. It will take 5-10 minutes, and finish with the message **Setup Complete**.

4.5 Exercise 1: Use Azure metrics to monitor performance

In this exercise, you'll perform the following tasks:

1. Configure Azure metrics for your Azure Database for PostgreSQL service.
2. Run a sample application that simulates multiple users querying the database.
3. View the metrics.

4.5.1 Task 1: Configure Azure metrics for your Azure Database for PostgreSQL service

1. Sign in the **LON-DEV-01** virtual machine running in the classroom environment as the **azureuser** user, with password **Pa55w.rd**.
2. Using a web browser, sign in to the Azure portal.
3. In the Azure portal, go to the page for your Azure Database for PostgreSQL service.
4. Under **Monitoring**, select **Metrics**.
5. On the chart page, add the following metric:

Property	Value
Scope	adventureworks[nnn]
Metric Namespace	PostgreSQL server standard metrics
Metric	Active Connections
Aggregation	Avg

   This metric displays the average number of connections made to the server each minute.

6. Select **Add metric**, and add the following metric:

Property	Value
Scope	adventureworks[nnn]
Metric Namespace	PostgreSQL server standard metrics
Metric	CPU percent
Aggregation	Avg

7. Select **Add metric**, and add the following metric:

Property	Value
Scope	adventureworks[nnn]
Metric Namespace	PostgreSQL server standard metrics
Metric	Memory percent
Aggregation	Avg

8. Select **Add metric**, and add the following metric:

Property	Value
Scope	adventureworks[nnn]
Metric Namespace	PostgreSQL server standard metrics
Metric	IO percent
Aggregation	Avg

These final three metrics show how resources are being consumed by the test application.

9. Set the time range for the chart to **Last 30 minutes**.
10. Select **Pin to Dashboard**, and then select **Pin**.

4.5.2 Task 2: Run a sample application that simulates multiple users querying the database

1. In the Azure portal, on the page for your Azure Database for PostgreSQL server, under **Settings**, select **Connection Strings**. Copy the ADO.NET connection string to the clipboard.
2. Switch to the Azure Cloud shell. Clone the repository holding the scripts and sample databases if you haven't done this previously.

   ```
   git clone https://github.com/MicrosoftLearning/DP-070-Migrate-Open-Source-Workloads-to-Azure ~/wor
   ```

3. Move to the *~/workshop/migration_samples/code/postgresql/AdventureWorksSoakTest* folder.

   ```
   cd ~/workshop/migration_samples/code/postgresql/AdventureWorksSoakTest
   ```

4. Open the App.config file using the code editor:

   ```
   code App.config
   ```

5. Replace the value of **Database** with **azureadventureworks**, and replace **ConectionString0** with the connection string from the clipboard. Change the **User Id** to **azureuser@adventureworks[nnn]**, and set the **Password** to **Pa55w.rd**. The completed file should look similar to the example below:

   ```
   <?xml version="1.0" encoding="utf-8" ?>
   <configuration>
       <appSettings>
           <add key="ConnectionString0" value="Server=adventureworks101.postgres.database.azure.com;D
           <add key="ConnectionString1" value="INSERT CONNECTION STRING HERE" />
           <add key="ConnectionString2" value="INSERT CONNECTION STRING HERE" />
           <add key="NumClients" value="100" />
           <add key="NumReplicas" value="1"/>
       </appSettings>
   </configuration>
   ```

   [!NOTE] Ignore the **ConnectionString1** and **ConnectionString2** settings for now. You will update these items later in the lab.

6. Save the changes and close the editor.
7. At the Cloud Shell prompt, run the following command to build and run the app:

   ```
   dotnet run
   ```

   When the app starts, it will spawn a number of threads, each thread simulating a user. The threads perform a loop, running a series of queries. You will see messages such as those shown below starting to appear:

   ```
   Client 48 : SELECT * FROM purchasing.vendor
   Response time: 630 ms

   Client 48 : SELECT * FROM sales.specialoffer
   Response time: 702 ms

   Client 43 : SELECT * FROM purchasing.vendor
   Response time: 190 ms

   Client 57 : SELECT * FROM sales.salesorderdetail
   Client 68 : SELECT * FROM production.vproductanddescription
   Response time: 51960 ms

   Client 55 : SELECT * FROM production.vproductanddescription
   Response time: 160212 ms

   Client 59 : SELECT * FROM person.person
   ```

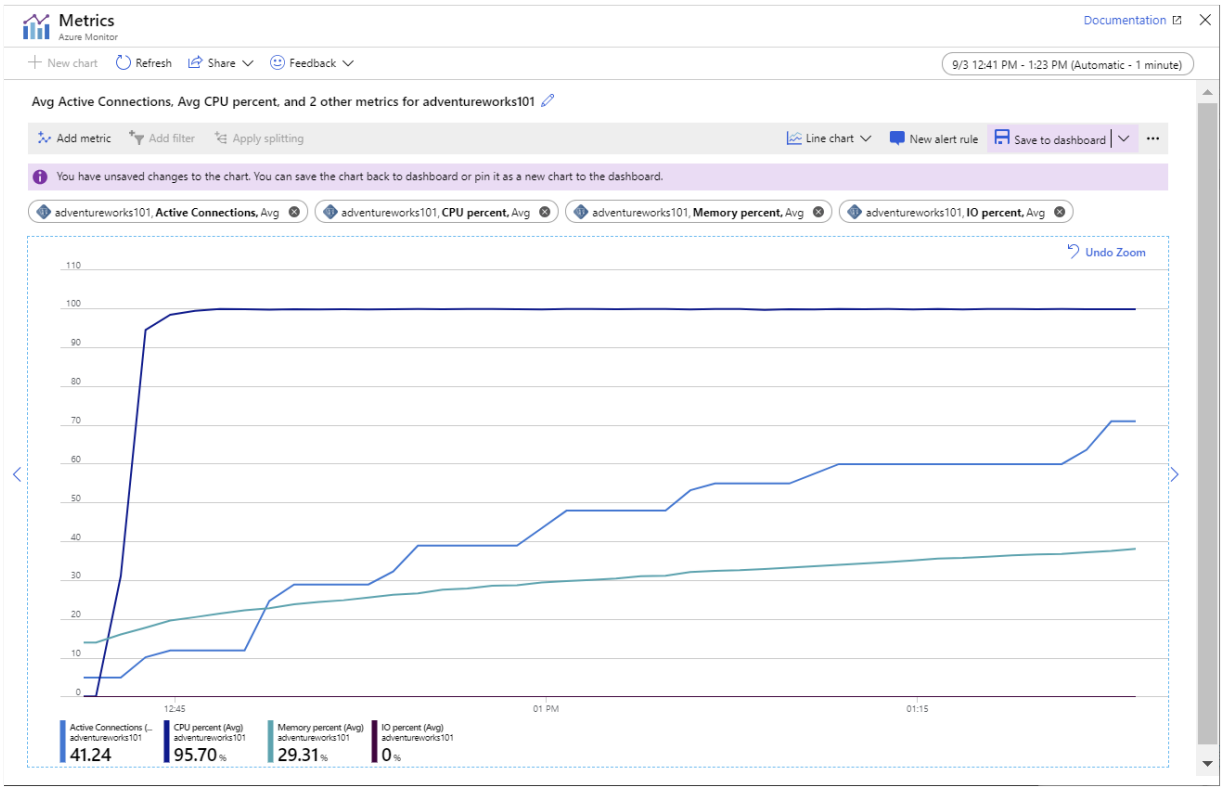
```
Response time: 186026 ms

Response time: 2191 ms

Client 37 : SELECT * FROM person.person
Response time: 168710 ms
```

Leave the app running while you perform the next tasks, and exercise 2.

4.5.3 Task 3: View the metrics

1. Return to the Azure portal.
2. In the left-hand pane, select **Dashboard**.

   You should see the chart displaying the metrics for your Azure Database for PostgreSQL service.
3. Select the chart to open it in the **Metrics** pane.
4. Allow the app to run for several minutes (the longer the better). As time passes, the metrics in the chart should resemble the pattern illustrated in the following image:



This chart highlights the following points:

- The CPU is running at full capacity; utilization reaches 100% very quickly.
- The number of connections slowly rises. The sample application is designed to start 101 clients in quick succession, but the server can only cope with opening a few connections at a time. The number of connections added at each ”step” in the chart is getting smaller, and the time between ”steps” is increasing. After approximately 45 minutes, the system was only able to establish 70 client connections.
- Memory utilization is increasing consistently over time.
- IO utilization is close to zero. All the data required by the client applications is currently cached in memory.

If you leave the application running long enough, you will see connections starting to fail, with the error messages shown in the following image.

```
Azure Cloud Shell
Bash
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Client 54 : SELECT * FROM production.vproductanddescription
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Response time: 303679 ms

Client 20 : SELECT * FROM person.person
Response time: 1093 ms

Client 33 : SELECT * FROM sales.salesorderheader
Client Client 6 failed with error: Connection is not open
Client 6 : SELECT * FROM production.vproductanddescription
Response time: 1469 ms

Response time: 275521 ms

Client Client 6 failed with error: Connection is not open
```

5. In the Cloud Shell, press Enter to stop the application.

4.6 Exercise 2: Use Query Store to examine the performance of queries

In this exercise, you'll perform the following tasks:

1. Configure the server to collect query performance data.
2. Examine the queries run by the application using Query Store.
3. Examine any waits that occur using Query Store.

4.6.1 Task 1: Configure the server to collect query performance data

1. In the Azure portal, on the page for your Azure Database for PostgreSQL server, under **Settings**, select **Server parameters**.
2. On the **Server parameters** page, set the following parameters to the values specified in the table below.

Parameter	Value
pg_qs.max_query_text_length	6000
pg_qs.query_capture_mode	ALL
pg_qs.replace_parameter_placeholders	ON
pg_qs.retention_period_in_days	7
pg_qs.track_utility	ON
pg_stat_statements.track	ALL
pgms_wait_sampling.history_period	100
pgms_wait_sampling.query_capture_mode	ALL

3. Select **Save**.

4.6.2 Task 2: Examine the queries run by the application using Query Store

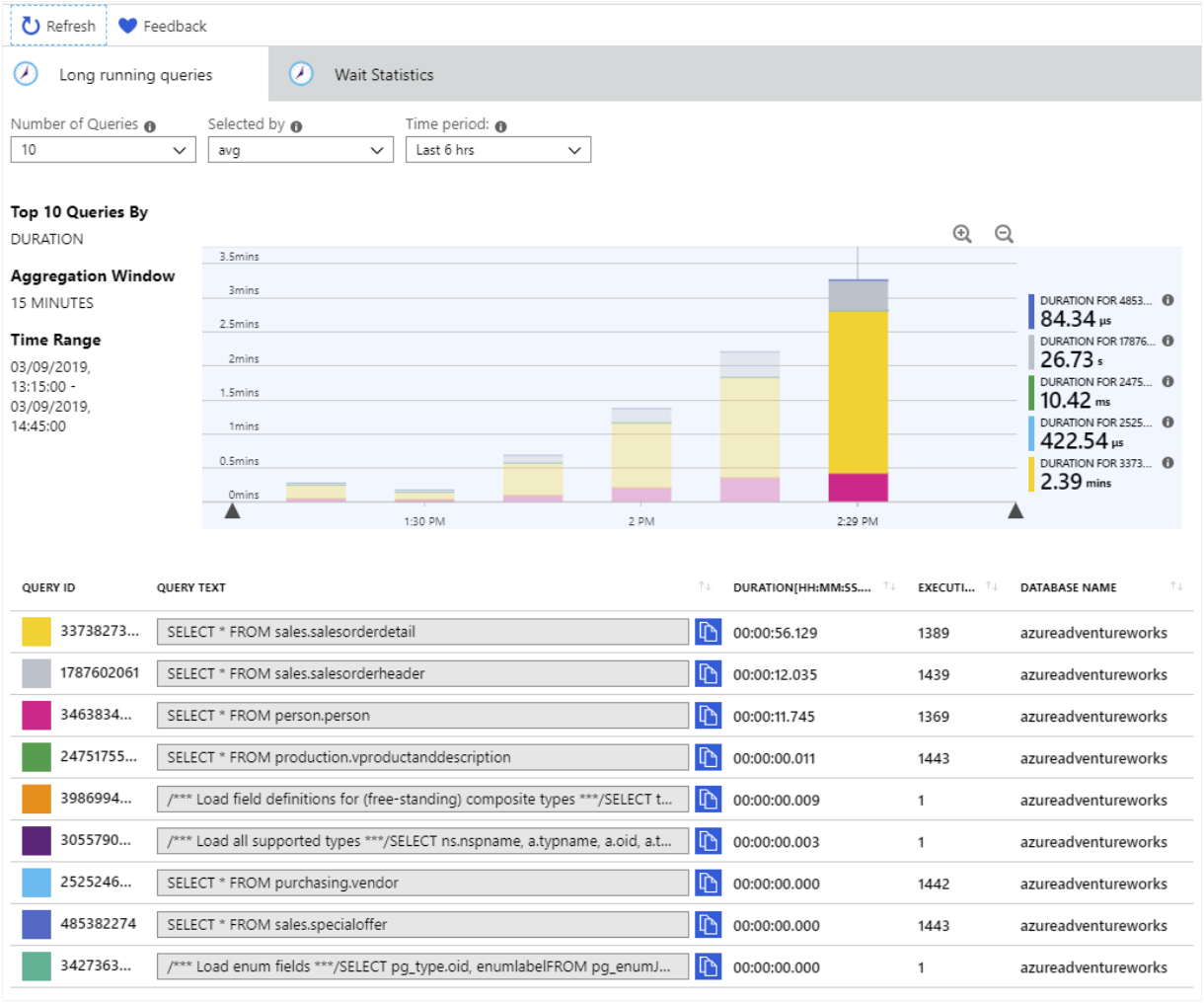
1. Return to the Cloud Shell, and restart the sample app:

   ```
   dotnet run
   ```

   Allow the app to run for 5 minutes or so before continuing.

2. Leave the app running and switch to the Azure portal
3. On the page for your Azure Database for PostgreSQL server, under **Intelligent performance**, select **Query Performance Insight**.
4. On the **Query Performance Insight** page, on the **Long running queries** tab, set **Number of Queries** to 10, set **Selected by** to **avg**, and set the **Time period** to **Last 6 hrs**.
5. Above the chart, select **Zoom in** (the magnifying glass icon with the "+" sign) a couple of times, to home in on the latest data.

   Depending on how long you have let the application run, you will see a chart similar to that shown below. Query Store aggregates the statistics for queries every 15 minutes, so each bar shows the relative time consumed by each query in each 15 minute period:



6. Hover the mouse over each bar in turn to view the statistics for the queries in that time period. The three queries that the system is spending most of its time performing are:

```
SELECT * FROM sales.salesorderdetail
SELECT * FROM sales.salesorderheader
SELECT * FROM person.person
```

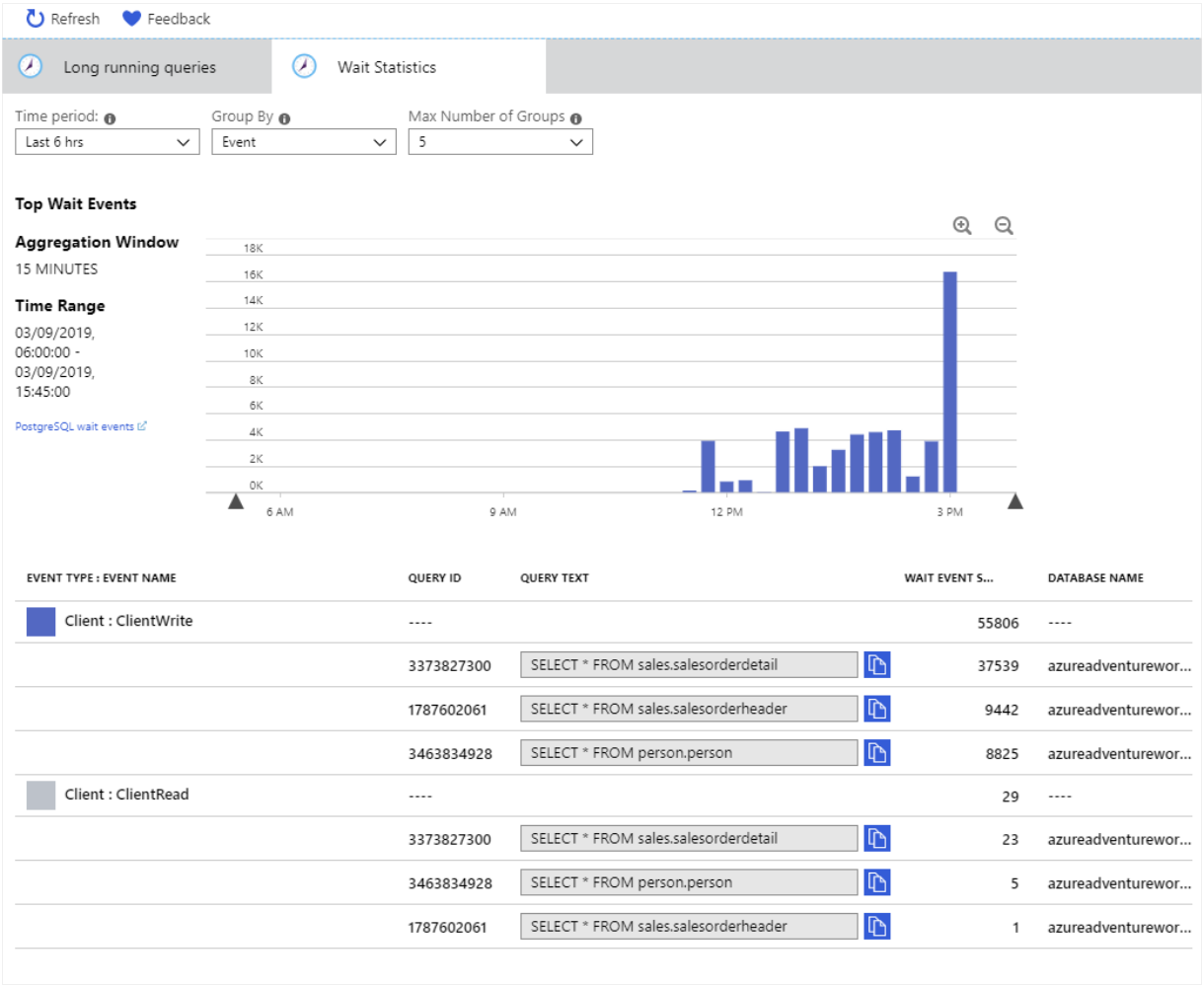
This information is useful for an administrator monitoring a system. Having an insight into the queries being run by users and apps enables you to understand the workloads being performed, and possibly make recommendations to application developers on how they can improve their code. For example, is it really necessary for an application to retrieve all 121,000+ rows from the **sales.salesorderdetail** table?

4.6.3 Task 3: Examine any waits that occur using Query Store

1. Select the **Wait Statistics** tab.
2. Set the **Time period** to **Last 6 hrs**, set **Group By** to **Event**, and set the **Max Number of Groups** to **5**.

   As with the **Long running queries** tab, the data is aggregated every 15 minutes. The table below the chart shows that the system has been the subject of two types of wait event:

   - **Client: ClientWrite**. This wait event occurs when the server is writing data (results) back to the client. It does **not** indicate waits incurred while writing to the database.
   - **Client: ClientRead**. This wait event occurs when the server is waiting to read data (query requests or other commands) from a client. It is **not** associated with time spent reading from the database.



[!NOTE] Read and writes to the database are indicated by **IO** events rather than **Client** events. The sample application does not incur any IO waits as all the data it requires is cached in memory after the first read. If the metrics showed that memory was running low, you would likely see IO wait events start to occur.

3. Return to the Cloud Shell, and press Enter to stop the sample application.

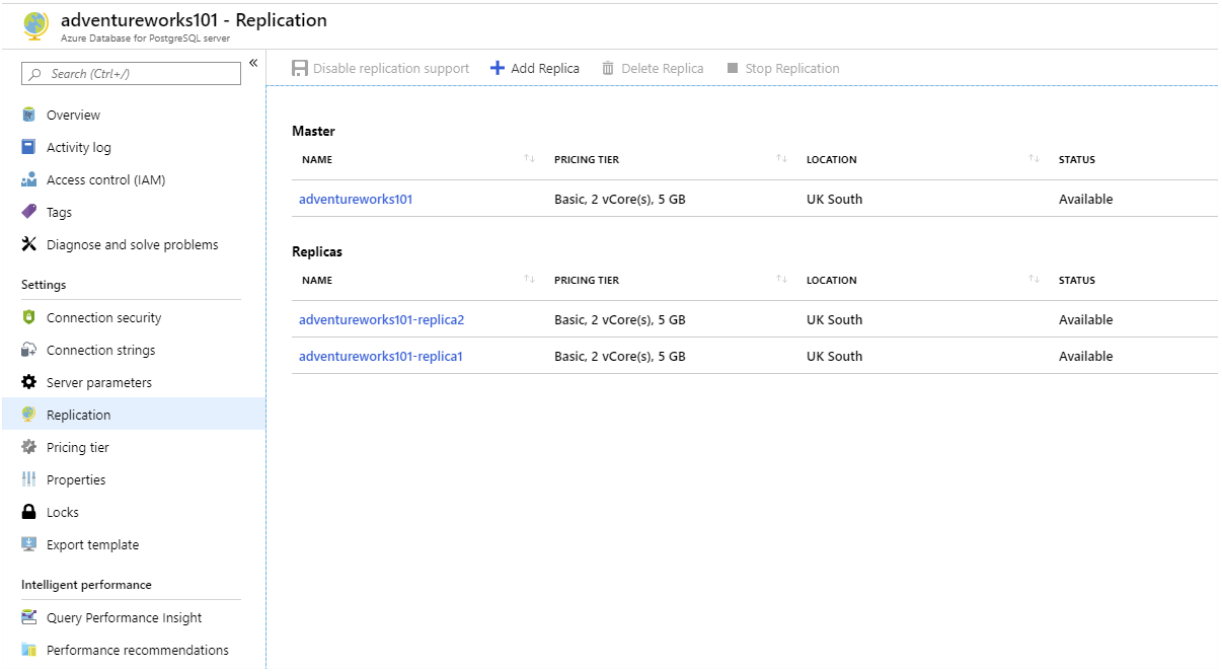
4.7 Exercise 3: Offload read operations for some users to use a read replica

In this exercise, you'll perform the following tasks:

1. Add replicas to the Azure Database for PostgreSQL service.
2. Configure the replicas to enable client access.
3. Restart each server.

4.7.1 Task 1: Add replicas to the Azure Database for PostgreSQL service

1. In the Azure portal, on the page for your Azure Database for PostgreSQL server, under **Settings**, select **Replication**.
2. On the **Replication** page, select **+ Add Replica**.
3. On the **PostgreSQL server** page, in the **Server name** box, type **adventureworks[nnn]-replica1**, and then select **OK**.
4. When the first replica has been created (it will take several minutes), repeat the previous step and add another replica named **adventureworks[nnn]-replica2**.
5. Wait until the status of both replicas changes from **Deploying** to **Available** before continuing.



4.7.2 Task 2: Configure the replicas to enable client access

1. Select the name of the **adventureworks[nnn]-replica1** replica. You will be taken to the page for the Azure Database for PostgreSQL page for this replica.
2. Under **Settings**, select **Connection security**.
3. On the **Connection security** page, set **Allow access to Azure services** to **ON**, and then select **Save**. This setting enables applications that you run using the Cloud Shell to access the server.
4. When the setting has been saved, repeat the previous steps and allow Azure services to access the **adventureworks[nnn]-replica2** replica.

4.7.3 Task 3: Restart each server

[!NOTE] Configuring replication does not require you to restart a server. The purpose of this task is to clear memory and any extraneous connections from each server, so that the metrics gathered when running the application again are *clean*.

1. Go to the page for the **adventureworks[nnn]** server.
2. On the **Overview** page, select **Restart**.
3. In the **Restart server** dialog box, select **Yes**.
4. Wait for the server to be restarted before continuing.
5. Following the same procedure, restart the the **adventureworks[nnn]-replica1** and **adventureworks[nnn]-replica2** servers.

4.8 Exercise 4: Monitor performance again and assess the results

In this exercise, you'll perform the following tasks:

1. Reconfigure the sample application to use the replicas.
2. Monitor the app and observe the differences in the performance metrics.

4.8.1 Task 1: Reconfigure the sample application to use the replicas

1. In the Cloud Shell, edit the App.config file.
2. Add the connections strings for the **ConnectionString1** and **ConnectionString2** settings. These values should be the same as that of **ConnectionString0**, but with the text **adventureworks[nnn]** replaced with **adventureworks[nnn]-replica1** and **adventureworks[nnn]-replica2** in the **Server** and **User Id** elements.
3. Set the **NumReplicas** setting to **3**.

   The App.config file should now look similar this:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="ConnectionString0" value="Server=adventureworks101.postgres.database.azure.com;D
        <add key="ConnectionString1" value="Server=adventureworks101-replica1.postgres.database.az
        <add key="ConnectionString2" value="Server=adventureworks101-replica2.postgres.database.az
        <add key="NumClients" value="100" />
        <add key="NumReplicas" value="3"/>
    </appSettings>
</configuration>
```

4. Save the file and close the editor.
5. Start the app running again:

   ```
   dotnet run
   ```

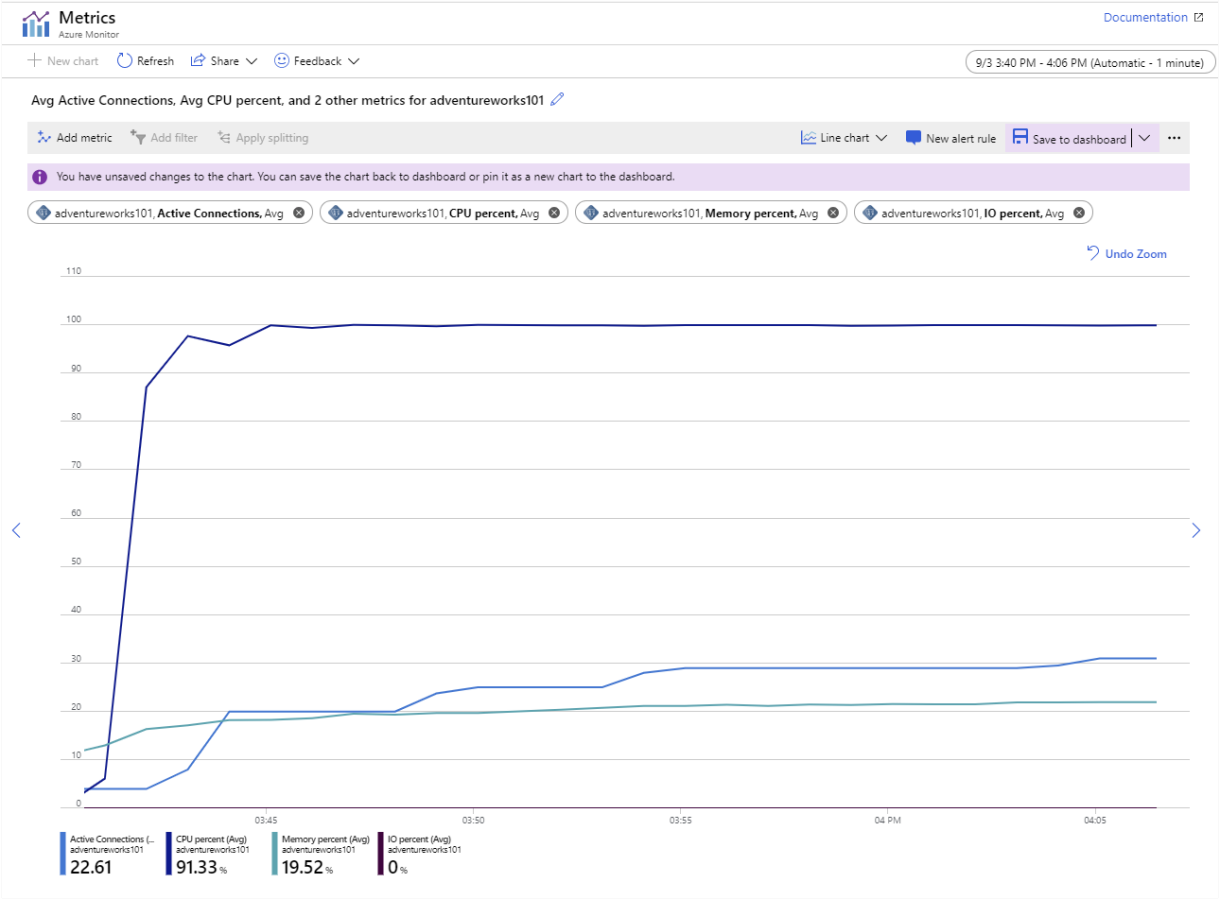
   The application will run as before. However, this time, the requests are distributed across the three servers.
6. Allow the app to run for a few minutes before continuing.

4.8.2 Task 2: Monitor the app and observe the differences in the performance metrics

1. Leave the app running and return to the Azure portal.
2. In the left-hand pane, select **Dashboard**.
3. Select the chart to open it in the **Metrics** pane.

   Remember that this chart displays the metrics for the adventureworks*[nnn]* server, but not the replicas. The load for each replica should be much the same.

   The example chart illustrates the metrics gathered for the application over a 30 minute period, from startup. The chart shows that CPU utilization was still high, but memory utilization was lower. Additionally, after approximately 25 minutes, the system had established connections for over 30 connections. This might not seem a favorable comparison to the previous configuration, which supported 70 connections after 45 minutes. However, the workload was now spread across three servers, which were all performing at the same level, and all 101 connections had been established. Furthermore, the system was able to carrying on running without reporting any connection failures.



You can address the issue of CPU utilization by scaling up to a higher pricing tier with more CPU cores. The example system used in this lab runs using the **Basic** pricing tier with 2 cores. Changing to the **General purpose** pricing tier will give you up to 64 cores.

4. Return to the Cloud Shell and press enter, to stop the app.

You have now seen how to monitor server activity using the tools available in the Azure portal. You have also learned how to configure replication, and seen how creating read-only replicas can distribute the workload in read-intensive scenarios.