

Microsoft Official Course



DP-070T00

Migrate Open Source Data Workloads to Azure

DP-070T00

Migrate Open Source Data Workloads to Azure

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks>¹ are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

¹ <http://www.microsoft.com/trademarks>

MICROSOFT LICENSE TERMS

MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

1. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
2. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
3. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facility that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
4. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of an MPN Member, or (iii) a Microsoft full-time employee.
5. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
6. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
7. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.
8. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.
9. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
10. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.
11. "MPN Member" means an active Microsoft Partner Network program member in good standing.
12. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
13. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led

Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.

14. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.

15. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.

2. **USE RIGHTS.** The Licensed Content is licensed not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

- 2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

1. **If you are a Microsoft IT Academy Program Member:**

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content, **provided you comply with the following:**
3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
4. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
5. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
6. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

7. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
 8. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
 9. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.
2. **If you are a Microsoft Learning Competency Member:**
1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
 2. For each license you acquire on behalf of an End User or MCT, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
 2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content, **provided you comply with the following:**
 3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 4. you will ensure that each End User attending a Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
 5. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 6. you will ensure that each MCT teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
 7. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
 8. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
 9. you will only provide access to the Trainer Content to MCTs.

3. If you are a MPN Member:

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content, **provided you comply with the following:**
3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
4. you will ensure that each End User attending a Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
5. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
6. you will ensure that each Trainer teaching a Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,
7. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
8. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
9. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
10. you will only provide access to the Trainer Content to Trainers.

4. If you are an End User:

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft

Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

5. **If you are a Trainer.**

1. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.
 2. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.
 - **2.2 Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.
 - **2.3 Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.
 - **2.4 Third Party Notices.** The Licensed Content may include third party code that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.
 - **2.5 Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.
3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("**Pre-release**"), then in addition to the other provisions in this agreement, these terms also apply:
1. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
 2. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft

to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.

3. **Pre-release Term.** If you are a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("**Pre-release term**"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.
4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
 - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
 - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
 - modify or create a derivative work of any Licensed Content,
 - publicly display, or make the Licensed Content available for others to access or use,
 - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
 - work around any technical limitations in the Licensed Content, or
 - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.
6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.
7. **SUPPORT SERVICES.** Because the Licensed Content is "as is", we may not provide support services for it.
8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or

updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.

11. **APPLICABLE LAW.**

1. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
2. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.

12. **LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.

13. **DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**

14. **LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque: Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection des consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les

garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contre-façon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised November 2014



Contents

■	Module 0 Welcome	1
	Welcome to the Course	1
■	Module 1 Introduction to open source database migration on Azure	5
	Module introduction	5
	MySQL and PostgreSQL database services in Azure	6
	Considerations for migration	9
	Approaches to migration	18
	Module summary	23
■	Module 2 Migrate on-premises MySQL to Azure Database for MySQL	27
	Module Introduction	27
	Introduction to Azure Database for MySQL	28
	Migrate on-premises MySQL database to Azure	40
	Application Migration	61
	Module summary	65
	Module Lab Information	66
■	Module 3 Migrate on-premises PostgreSQL to Azure Database for PostgreSQL	69
	Module introduction	69
	Introduction to Azure Database for PostgreSQL	70
	Migrate on-premises PostgreSQL database to Azure	83
	Application Migration	106
	Module summary	110
	Module Lab Information	111
■	Module 4 Protecting monitoring and tuning	115
	Module Introduction	115
	Protecting a database and server	116
	Monitoring and tuning	123
	Module summary	144
	Module Lab Information	145



Module 0 Welcome

Welcome to the Course

Implementing an Azure Data Solution

This course has been created by the Microsoft World Wide Learning team

Learning objectives

In this module you will gain information:

- About this course
- About the audience
- Course pre-requisites
- Understand the DP 070 certification
- About this course

Course Introduction

This module provides an overview of the following

- About this course
- Course agenda
- Course audience
- Course pre-requisites
- DP070: Migrating Open Source Workloads to Azure

About this course

The Migrating Open Source Workloads to Azure is a specialist course that enables an individual to understand the tools and processes required to migrate on-premises MySQL or Postgres solutions to Azure Database for MySQL and Azure Database for Postgres respectively.

Course Agenda

At the end of this course, the student will learn:

Module 1: Introduction to open source database migration on Azure

In this module, the students will learn what Open Source (OSS) Database are and the common workloads that run on such systems. Students will also learn about the common customer challenges that can occur when migrating OSS systems to the cloud .

Module Objectives:

At the end of this module, the students will be able to:

- Describe Open Source Systems Databases
- Understand the considerations for migration
- Explain the approaches to migration

Module 2: Migrate on-premises MySQL to Azure Database for MySQL

In this module, the student will learn how to configure and manage Azure Database for MySQL. They will then learn how to migrate the systems from an on-premises server to the cloud, and how to deal with application issues during the migration.

Module Objectives:

At the end of this module, the students will be able to:

- Introduction to Azure Database for MySQL
- Migrate on-premises MySQL to Azure
- Handle applicaiton migrations

Module 3: Migrate on-premises MySQL to Azure Database for PostgreSQL

In this module, the student will learn how to configure and manage Azure Database for Postgres. They will then learn how to migrate the systems from an on-premises server to the cloud, and how to deal with application issues during the migration.

Module Objectives:

At the end of this module, the students will be able to:

- Introduction to Azure Database for PostgreSQL
- Migrate on-premises PostgreSQL to Azure
- Handle applicaiton migrations

Module 4: Protecting monitoring and tuning

In this module, the student will learn the various tools that can be used to monitor and tune MySQL and PostgreSQL

Module Objectives:

At the end of this module, the students will be able to:

- Protecting a database and server
- Monitoring and tuning

Course Audience

The course is ideally suited to Database Administrators, Data Engineers, Data Architects or Migration specialist who have an understanding of MySQL or Postgres and Azure Services and are seeking information on how to perform a data migration project to the cloud . This is done as a single one-day course.

Course Prerequisites

In addition to their professional experience, students who take this training should have technical knowledge equivalent to the following courses:

- **Azure fundamentals**¹

DP-070 Certification Details

There is no certification exam associated with this course

¹ <https://docs.microsoft.com/en-us/learn/paths/azure-fundamentals/>



Module 1 Introduction to open source database migration on Azure

Module introduction

Module 1: Introduction to open source database migration on Azure

Module introduction

In this module, you'll learn about the issues and considerations for migrating on-premises open source databases to Azure. You'll be introduced to the services that Azure provides to help you migrate and host your databases. You'll look at what you need to consider when planning a migration project, and you'll learn about different approaches to migrating databases.

Learning objectives

By the end of this module, you'll be able to:

- Describe the features and services available in Azure for hosting an open source database.
- Explain the key considerations for implementing a migration project.
- Describe different approaches that you can take for migrating databases.

MySQL and PostgreSQL database services in Azure

Lesson 1: MySQL and PostgreSQL database services in Azure

If you want to run a MySQL database or a PostgreSQL database in Azure, there's plenty of choice. For example, if you have a set of servers on-premises—and you want to replicate that infrastructure as closely as possible in the cloud—you can create a matching set of virtual machines and install the database server software on them. Alternatively, if you want to dispense with as many administrative tasks as possible, you could use **Azure Database for MySQL** or **Azure Database for PostgreSQL**. In that case, you don't have to worry about virtual servers, operating systems, or database software installations. Instead, you can start to create the database and let Azure manage all the software.

In this lesson, you'll see how Azure hosts database software.

Learning objectives

By the end of this lesson, you'll understand how to:

- Decide whether to use an infrastructure as a service (IaaS) approach or a platform as a service (PaaS) approach to host a database server in the cloud.
- Choose whether to run a database workload on Azure Database for MySQL or Azure Database for PostgreSQL.
- Choose whether to run a MySQL or PostgreSQL database on Azure virtual machines or in containers.

Overview of Azure as a database platform

If you have a functional on-premises database, why should you consider moving it into a cloud service, such as Azure? Similarly, why consider implementing a new database system in Azure? Many organizations perform this migration because it's easy and relatively cheap to create a highly available and scalable service in the cloud, compared to using an on-premises datacenter.

Availability

The availability guaranteed by Azure service-level agreements (SLAs) depends on the precise details of your implementation—but they're always high. For example, for Azure Database for PostgreSQL, Microsoft guarantees 99.99% availability with no additional cost.

If you wanted to achieve this availability in an on-premises network, you would have to architect a system that's resilient to hardware failures. You would need:

- Multiple physical servers.
- A load balancer that could reroute queries if there's a server failure.
- A storage area network that shares data between servers or a way to replicate data between servers.
- Resilient network hardware.
- An environmentally controlled datacenter.
- Uninterruptible power supplies and backup generators.

All these components are expensive and require skill to implement and run. They would also consume significant administrative resources. With Azure, these requirements are already solved; you just create the database, and high availability is built-in.

Scalability

If your user base grows, system demand grows with it. Every on-premises system has a maximum capacity—if you approach that limit, you must increase it by adding more hardware. You can't add capacity instantaneously. Instead, you must purchase the hardware, install the necessary operating system and software, apply updates, and then add the data to the new database instance. All these things take time.

Also remember that high demand is often temporary. For example, if you run a successful marketing campaign, you might see a peak in traffic, followed by a return to lower demand. In an on-premises set-up, you have to design the system for these peaks. This means that the system is under-utilized most of the time but still runs up bills and requires maintenance.

These challenges are much easier to overcome in the cloud. If your system reaches its capacity, you can respond very quickly—for example, by moving to a larger tier or adding virtual machines. If demand drops, you'll quickly save money by removing capacity. In Azure, you only pay for the capacity you use.

IaaS and PaaS

You can choose from at least two approaches when you implement a database on Azure, depending on the level of control you need:

- **Infrastructure as a service (IaaS).** If you choose the IaaS approach, Azure runs the physical infrastructure for you. You create virtual machines and virtual networks to connect them, and then install the necessary software and data. Running a virtual machine is like running a physical server. You maintain the operating system and software but you don't have to worry about the datacenter, environmental control, or connections to the internet.
- **Platform as a service (PaaS).** If you choose the PaaS approach, Azure runs the physical infrastructure, the necessary virtual servers, and the database software. You don't have to perform configuration or maintenance tasks on these components. For example, Azure applies services packs automatically. You can concentrate on database administration. PaaS offerings for running open source databases on Azure include **Azure Database for MySQL** and **Azure Database for PostgreSQL**.

Lesson 1: Summary

In this lesson, you saw how to:

- Decide whether to use an infrastructure as a service (IaaS) approach or a platform as a service (PaaS) approach to host a database server in the cloud.
- Choose whether to run a database workload on Azure Database for MySQL or Azure Database for PostgreSQL.
- Choose whether to run a MySQL or PostgreSQL database on Azure virtual machines or in containers.

Knowledge Check

Multiple choice

You want to create a highly available database system that requires the minimum of administrative time to run.

Should you use an IaaS or a PaaS approach?

- Use a IaaS approach. Create a set of virtual machines to run your system, and connect them using a virtual network. You have full control over the databases and server running on these virtual machines.
- Use a PaaS approach. For example, run a PostgreSQL database using Azure Database for PostgreSQL.

Multiple choice

You want to run a MySQL database in an IaaS environment in which you'll create new instances as quickly as possible.

Should you use virtual machines or containers?

- Use containers.
- Use virtual machines.

Considerations for migration

Lesson 2: Considerations for migration

There are many issues you should consider carefully before embarking on a database migration project. These issues concern items such as:

- The impact on any dependent systems of moving a database from one environment to another.
- The migration strategy to adopt.
- How to keep your business functions operational during a migration.
- How to maintain security in a new environment.
- How to manage and maintain databases running in Azure.

In this lesson, you'll learn about many of these considerations.

Learning objectives

By the end of this lesson, you'll understand how to:

- Assess dependencies and fall-back requirements.
- Select the migration approach—online or offline.
- Determine the most suitable way to migrate applications that use your databases.
- Maintain parallel systems.
- Consider whether to migrate a database system in one unit, or piecemeal by workload.
- Assess security concerns.
- Determine the best way to monitor and manage your databases in Azure.

Assess dependencies and fall-back requirements

A business system running on-premises can have an architecture that's coupled to other services operating within the same environment. It's important to understand the relationships between a system you wish to migrate, and the other applications and services your organization is currently using.

Investigate dependencies

In a complex system, a component rarely functions entirely independently. Instead, it makes calls to other processes and components. Databases, for example, might depend on directory services that host user accounts. When you move a database into the cloud, can you access that directory service? If not, how will users sign in? When you overlook a dependency like this, there might be a total failure of the database.

To mitigate risks, check whether your database depends on services such as the following:

- Directory services, such as Active Directory.
- Other stores of security principals.
- Other databases.
- Web APIs or other network services.

Also remember that other components might depend on your database. If you move the database without reconfiguring the dependent components, what are the consequences? For example, if you move a product catalog database, and the public-facing website depends on it to determine what products to present to users, will the move cause an interruption in service?

Check to see if any of these types of component depend on your database:

- Websites and web APIs.
- Client apps, such as mobile apps and desktop software.
- Other databases.
- Reports.
- Data warehouses.

To make these checks, talk to the stakeholders, administrators, and developers involved with each database and system component. Consult the documentation then, if you're not confident that you understand the behavior of the systems, consider running tests, such as network captures to observe behavior.

Prepare to fall back

In any migration project, you should always be prepared for a failure. In a database migration project to the cloud, the worst eventuality is that the new database becomes unavailable and users can't do their jobs. It's common to mitigate this risk, which might have a large impact on your company's profitability, by planning to roll back to the original, unmodified database on-premises.

For the fall-back plan, consider:

- How to ensure you can fall back to a database that's untouched by the failed migration. For example, it's highly recommended to take a full database backup, just before you begin the migration.
- How long is it acceptable for the database to be offline, if you need to fall back?
- How much budget do you have for the fall-back plan? For example, can you afford to replicate the database to a dedicated fall-back server? If so, you can switch this server off just before the migration. To fall back, you boot up this server. You would immediately have a database that's unaffected by the migration, without having to restore it from backup.

Offline versus online migration

To migrate a database, you have at least two options:

- Halt the system, transfer the database, then reconfigure and restart the system to use the new database. This is an offline migration.
- Keep the system running while you move the database to its new location, roll forward transactions being performed against the original database to the new database while the migration is proceeding, and then switch your applications to connect to the new database. This is an online migration.

It's simpler to perform an offline migration, where users can't change the data while the migration takes place. However, if your database is busy or critical to the success of your organization, that might not be possible.

Offline migrations

Suppose that your database supports a team of analysts who all work in a single time zone during normal business hours. The team usually doesn't work at weekends. Between 6:00 PM on Friday and 9:00 AM on Sunday, the database isn't often used.

In this situation, you could do an offline migration over the weekend, by following these steps:

1. Take the database offline, after all transactions have completed on Friday evening.
2. Take a full backup or export of the database.
3. Shut down the on-premises server and keep it in case you need to fall back.
4. Restore the database on the target cloud system.
5. Bring the target system online.
6. Reconfigure clients to connect to the cloud database.

In this case, an offline migration is possible because there's a long period when an interruption in service has little effect on users. During this time, you can do a complete backup and restore of the database, knowing that you won't miss any changes.

Online migrations

Now consider another database that supports a sales app. Sales staff are distributed around the world and also work at weekends. There isn't a period of low activity, the database is always busy and, if you take the database offline for a significant period, it will impact users. Sales activity is business-critical, so an interruption in service will have a direct affect on the organization's bottom line.

In cases like this, consider performing an online migration. In an online migration, downtime is limited to the time it takes to switch to the new database. Use a tool such as the Azure Data Migration Service to execute an online migration. Online migrations have the following differences to offline migrations:

- You don't move the original database offline before taking a backup or export.
- While the migration is in progress, changes apply to the old database.
- The migration tool ensures that these changes are copied to the new database before the switch over. This is often achieved by reconfiguring the old database to replicate changes to the new one.

Application migration

After you've migrated a database, how (and when) should you cut over to the new system and update applications to use the new database? You might:

- Move applications one-by-one to the new database.
- Move subsets of users.
- Adopt a combination of both approaches.

The intention is that you perform application migration in small stages that can be easily rolled back if something goes awry. Regardless of whether you've followed an offline or online approach to database migration, you should still have a working configuration located at the original source. In theory, you'll be able to switch back to the original source quickly. But if the data is constantly changing, you need to consider where these changes have been made.

- In an offline migration, the source and destinations are independent of each other. Users and applications might no longer see a consistent view of the data. In a transactional system, this situation is

likely to be unacceptable. In this case, you would need to maintain some form of bidirectional replication between databases while both systems remain live. Alternatively, if the purpose of an application is to generate monthly or weekly reports, generate sales projections, or perform other statistical operations, this lack of consistency might not be so worrying. Such applications take a “long view” of the data, rather than being dependent on up-to-date data. In this latter case, transactional applications use the new database, whereas reporting applications are moved more slowly.

- In an online migration, the new database is kept synchronized with the old, usually by some form of replication. The replication process might be asynchronous, so there could be a lag. However, changes made to data in the new database won't be propagated back to the old, resulting in possible conflicts. An application running against the old database might make a conflicting change to data that's been modified in the new database. Replication will blindly overwrite the change in the new database, resulting in a “lost update”.

Approaches to testing

If the database plays a critical role in your business, the consequences of a failure might be extensive. To increase your confidence that this won't happen, consider running performance tests against the migrated database to ensure that it copes with the load users might place upon it and respond quickly. Remember that there could be periods of peak activity, when demand is much higher than normal. You must be sure that your migrated system handles the expected workload.

Always perform some type of regression testing against the new database before allowing access to users. These tests will verify that the behavior and functionality of the system are correct.

Additionally, you should consider running a “soak test”. A soak test is a load test designed to see how the system as a whole operates under pressure. A soak test stresses the new database and determines whether it's stable under high demand. A soak test runs over a significant time period to see what happens when high demand persists.

When you've established that the new system is stable, you can start to migrate users. However, you might need additional assurance that users will find the new system acceptable. At this point, you might consider “canary testing”. A canary test transparently directs a small subset of users to the new system, but they aren't aware that they're accessing it. It's a form of blind test, intended to enable you to find any problems or issues with the new system. Monitor the responses from these users, and make any adjustments required.

Maintaining parallel systems

There are several reasons why you might choose to run the old on-premises database in parallel with the new cloud database:

- **Testing periods.** As you saw in the previous topic, it's a good idea to run canary tests against the migrated database to assess its functionality, stability, and capacity before using it to support people. Maintaining the on-premises system in parallel gives you a quick way to revert users to the old system if there are issues with the new system.
- **Phased migrations.** One way to mitigate the impact of unexpected failures on production is to move a small number of users to the new system first, and monitor the results. If all runs smoothly, you could move other groups of users as you gain confidence in the new database. You can move users alphabetically, by department, by location, or by role, until they're all on the new database.
- **Piecemeal migrations.** Another approach is to segment the migration not by user, but by workload. For example, you could migrate the database tables that support human resources, before those that support sales.

In all these cases, there's a period when the old on-premises database runs in parallel with the new cloud database. You must ensure that changes made to the old database are also applied to the new database and that they flow in the opposite direction. You could script this synchronization, or use a tool like Azure Data Migration Service.

If you decide to maintain parallel databases and synchronize changes, ask yourself these questions:

- **Conflict resolution.** Is it likely that a change to a row on-premises happens at a similar time to a different change to the same row in the cloud? This would create a conflict, where it's unclear which change should be retained. How would you resolve such conflicts?
- **Network traffic.** How much network traffic will be generated while changes are synchronized between databases? Do you have enough network capacity for this traffic?
- **Latency.** When there's a change in one of the databases, what lag (if any) is acceptable before that change reaches the other database? For example, in a product catalog, you might be able to wait for up to a day before new products are visible to all users. However, if the database includes critical transactional information, such as currency exchange rates, that data should be synchronized much more frequently, if not instantaneously.

Piecemeal migration

A piecemeal migration is where you divide a complete system into workloads and migrate one workload at a time.

Multiple databases

A complex system might include multiple databases that support several workloads. For example, human resources might use the *StaffDB* database, while the sales team could have mobile apps that query both the *ProductCatalogDB* database and the *OrdersDB* database.

Of course, you can choose to migrate the entire database system to the cloud in one go. This might be simpler, because you don't have to run databases both on-premises and in the cloud. You don't need to consider how those databases communicate during the migration. However, the risks of failure are higher. A single problem might affect both the human resources team and the sales team.

Consider mitigating these risks for medium and large database systems by performing a piecemeal migration, where you move one workload at a time. In this example, you might consider migrating the *StaffDB* database first, because the problems associated with a failure would be limited to the human resources team and wouldn't prevent you from taking orders. By solving any problems that arise with the *StaffDB* migration, you'll learn lessons that apply to other workload migrations.

Next, you could think about migrating the *Product Catalog* workload to the cloud. If you do, consider questions such as:

- How do you ensure that products newly added to the catalog, are available to order?
- Do you need to synchronize any data with the *OrdersDB* database, which remains on-premises?
- What latency is acceptable for new products to reach the *OrdersDB* database from the product catalog?

Single database piecemeal migrations

Even if you only have a single database that supports all the workloads, you can still consider a piecemeal migration. For example, you could divide the database into pieces like this:

- Tables that support HR operations.
- Tables that support sales.
- Tables that support analysis and reporting.

If you migrate the HR operations tables first, any problem that arises only affects HR personnel. Sales and data analysts continue to work on the old database without interruption.

Before you perform a piecemeal migration, you must fully understand the databases and how they're used by applications. For example, some tables in your database might support both sales and reporting. That means you can't cleanly divide workloads. You must synchronize these tables between on-premises and cloud databases, until all the workloads have been migrated.

Security concerns

Your databases might contain sensitive data, such as product details, personal staff information, and payment details—so security is always a high priority. You must decide how to protect this data during the migration and in the new database.

Firewall protection

In an internet-connected application, database servers are usually protected by at least two firewalls. The first firewall separates the internet from the front-end servers—if these servers host websites or web APIs, for example. Only TCP port 80 should be open on the outer firewall, but this port must be open for all source IP addresses, except blacklisted addresses.

The second firewall separates the front-end servers from the database servers. It's recommended to publish the database service on a private port number that's not known to the outside world. On the second firewall, open this port number only for the IP addresses of the front-end servers. This arrangement prevents any direct communication from a malicious internet user to the database servers.

If you plan to migrate database servers to Azure virtual machines, use a virtual network with Network Security Groups (NSGs) to replicate firewall rules. If you use **Azure Database for MySQL** or **Azure Database for PostgreSQL**, you can create firewall rules to protect the database using the **Connection security** page for the server in the Azure portal.

Authentication and authorization

In most databases, you need to closely control who accesses and modifies which data. This control requires that users are positively identified when they connect to the database. This process is called **authentication** and is usually done with a username and password. Database systems such as MySQL and PostgreSQL provide their own authentication mechanisms. You must ensure that you continue to authenticate users securely when you migrate your systems to the cloud.

[NOTE]

The **Azure Database for MySQL** and **Azure Database for PostgreSQL** services emulate traditional MySQL and PostgreSQL authentication.

When you know who the user is, you must assign them permissions to complete the tasks that are part of their job. This process is called **authorization**.

For a database migration project, you have to make sure that users are authorized correctly in the cloud database:

1. Find out where user accounts are stored in the on-premises system. In MySQL, user accounts are usually stored in the `user` table of the `mysql` database but it's possible, for example, to integrate with user accounts stored in Active Directory.

2. Get a list of all the user accounts. In MySQL, for example, you could use this command:

```
SELECT host, user FROM mysql.user;
```

3. For each user account, find out what access they have to the database. In MySQL, for example, you could use this command for the `dbadmin@on-premises-host` user account:

```
SHOW GRANTS FOR 'dbadmin'@'on-premises-host';
```

4. Recreate each user account in the cloud database. In MySQL, for example, you could use this command to create a new account:

```
CREATE USER 'dbadmin'@'cloud-host'
```

5. Authorize each user account to the correct level of access in the cloud database. In MySQL, for example, you could use this command to permit a user to access the complete database:

```
GRANT USAGE ON *.* TO 'dbadmin'@'cloud-host'
```

[!IMPORTANT]

Neither Azure Database for MySQL nor Azure Database for PostgreSQL currently support user accounts in Azure Active Directory. If you're using Active Directory to store user accounts on-premises, you must migrate those user accounts to another store in the cloud.

Encryption

As data is sent across the network, it might be intercepted by a so-called "man-in-the-middle" attack. To prevent this, both **Azure Database for MySQL** and **Azure Database for PostgreSQL** support Secure Sockets Layer (SSL) to encrypt communications. SSL is enforced by default, and it's highly recommended that you don't change this setting.

You might need to amend your client applications' connection settings to use SSL encryption. Discuss this topic with your developers to determine the changes, if any, that are necessary.

Monitoring and management

Part of planning to migrate a database is to consider how database administrators will continue to perform their tasks after migration.

Monitoring

On-premises database administrators are used to monitoring regularly to spot problems such as hardware bottlenecks, or contention for network access. They monitor to ensure they can fix any problems before productivity is affected. You can expect any database that's not regularly monitored to begin causing problems sooner or later.

You should take exactly the same approach to cloud databases. It might be easier to solve problems in a PaaS system like Azure, because you can add resources without buying, setting up, and configuring any hardware. However, you still need to spot developing problems, so monitoring is a high priority among your daily tasks.

Before you migrate databases into the cloud, find out what monitoring tools your administrators currently use. If those tools are compatible with your proposed cloud-based solution, you might only need to reconnect them to the migrated databases. If not, you must investigate alternatives.

Bear in mind that Azure includes a set of performance monitoring tools, and collects a wide variety of performance counters and log data. You display this data using dashboards and charts in the Azure portal, by configuring Azure Monitor. You create custom graphs, tables, and reports, specifically designed for the needs of your administrators.

Management

Your database administrators use preferred tools to change the schema and content of the database on-premises. If they use the same tools after migration, you can continue to benefit from their expertise. Start by assessing whether the existing set of tools is compatible with the proposed cloud-hosted database. Many tools will be compatible because they're based on widely adopted standards such as SQL—but it's important to verify that compatibility. If the current management tools won't work after migration, try to identify alternatives with your administrators.

Azure includes several tools that you could use to administer MySQL and PostgreSQL databases:

- **The Azure portal.** This website has powerful facilities that you use to configure, monitor, and manage databases—and all other resources that you might create in the Azure cloud.
- **Azure PowerShell.** This is a scripting execution environment and language that has a wide set of commands. Use PowerShell, which is available for Windows and Linux environments, to automate complex database administrative tasks.
- **Azure CLI.** This is a command-line interface to Azure. Use it to manage services and resources in Azure. You can use the CLI from the Windows and Linux shell environments, and from within Bash scripts.

Lesson 2: Summary

In this lesson, you saw how to:

- Assess dependencies and fall-back requirements.
- Select the migration approach—online or offline.
- Determine the most suitable way to migrate applications that use your databases.
- Maintain parallel systems.
- Consider whether to migrate a database system in one unit, or piecemeal by workload.
- Assess security concerns.
- Determine the best way to monitor, and manage your databases in Azure.

Knowledge Check

Multiple choice

Offline migration enables users to continue accessing your existing on-premises database while data is copied to a new database in the cloud. Changes made to the existing database during migration will be automatically applied to the new database.

True or false?

- True.
- False.

Multiple choice

You want to migrate individual applications and users running specific workloads to the cloud as an initial step in migrating your system. If successful, you'll migrate further workloads until the entire system has been moved.

How would you do this?

- Perform canary testing.
- Perform an online migration of the database.
- Follow a piecemeal approach to migration.
- Perform an offline migration of the database.
- Replicate your database to a server running on a virtual machine in the cloud.

Approaches to migration

Lesson 3: Approaches to migration

In this lesson, you'll see several different ways to approach a database migration project. For offline migrations, you use export and import, or backup and restore techniques to move data, schema, and database objects from an on-premises database into the cloud. For an online migration, you use the Azure Database Migration Service to ensure that databases remain synchronized while the migration takes place. If you have complex, custom requirements and plenty of time for development and testing, you could write your own custom scripts and applications to perform an offline or online migration.

Learning objectives

By the end of this lesson, you'll understand how to:

- Export data from an on-premises system and import it into a cloud database.
- Use a backup and restore technique to migrate data into a cloud database.
- Migrate database schema, content, and security information by developing custom code.
- Use the Azure Database Migration Service to move data.
- Move MySQL data into Azure SQL Database as an alternative to running MySQL in the cloud.

Export and import

A common approach to data migration is to selectively export data from one or more source databases into an intermediate file. You then import the data in that file into the destination database.

When to use export and import

Export and import techniques give you control over the data and schema that's moved in the migration. Use export and import tools if you want to select which data is migrated to the new database, and perhaps clean or modify the data during migration.

Consider using export and import techniques:

- When you want to choose a subset of the tables in the on-premises database to migrate to the cloud database.
- When you want to migrate database objects such as constraints, views, functions, procedures, and triggers, and control how these objects are set up in the cloud database.
- When you want to import data from external sources other than MySQL or PostgreSQL.

For example, you might consider export and import in these scenarios:

- You want to perform a piecemeal migration where the marketing workload is migrated to the cloud and tested before the sales support workload. Both workloads use tables from the *SalesDB* database in your on-premises system. You want to migrate the marketing tables only in the first phase of the project and the sales tables only in the second phase.
- Your on-premises data is old and contains a mix of data that's relevant and irrelevant to the current business. You want to take the opportunity to remove old data and consider a more streamlined database schema.

- You have a large spreadsheet that contains data about products. You want to migrate this data into the cloud database.

Plan an export and import migration

The advantage of using export and import is the extra level of control you have over the data being migrated. However, a disadvantage is that you must plan more carefully to ensure that all the objects you need are included.

Make sure you understand how the following objects will be migrated:

- The database schema.
- Constraints including primary keys, foreign keys, and indexes.
- Views, functions, procedures, and triggers.
- User accounts and permissions.

MySQL export and import

You can use SQL scripts to perform selective export and import from one database to another. However, if your on-premises database is in MySQL, there are several tools available to help, including:

- **MySQL Workbench.** This is a popular database design tool with a Graphical User Interface (GUI) developed by Oracle Corporation. It includes a **Data Export** tool with flexible data selection options.
- **Navicat.** This database administration GUI tool is also compatible with MariaDB databases.
- **mysqlimport.** This command-line tool can import data from text files.

[IMPORTANT]

Azure Database for MySQL only supports the InnoDB storage engine. If you have any tables that use the MyISAM engine, you must convert them to InnoDB before migrating to Azure.

PostgreSQL export and import

PostgreSQL provides the following tools that you use to export and import data:

- **pgAdmin.** This is a GUI utility for PostgreSQL administrators. It provides an interface for exporting and importing data.
- **pg_dump.** This is a command-line tool you use to export a database in various formats, including text. You can edit the resulting .sql files before you import them using the **psql** utility.

Backup and restore

Backup and restore operations are usually done to protect a database against disasters. An exact copy of the database is taken and saved. If a disaster destroys the working copy, the backed-up copy is restored and normal business can resume.

By restoring to a different location, you use a backup to migrate the complete database to another location, such as a database in the cloud.

When to use backup and restore

Backup tools make a simple and precise copy of the database. When you restore in the cloud database, you get exactly the same data and schema that you had in the on-premises system. Use backup and restore to migrate a database:

- When you want to migrate an entire database or set of databases in one operation.
- When you don't need to make any modifications to the data, schema, or other database objects during migration.

You could consider using backup and restore to perform a migration in cases like these:

- You have a single database system that you want to lift-and-shift into the cloud with as little modification as possible.
- You want to perform a piecemeal migration on a system that has multiple databases. Each workload is supported by a complete database.

When you restore a database from a backup file to a cloud location, consider the quantity of data that must be sent across the network to the cloud database. To optimize this data transfer, copy the backed-up database to a virtual machine in the same region as the destination database, and restore from there. This restore is quicker than using an on-premises backup file and is less likely to cause contention for network bandwidth.

Plan a backup and restore on MySQL

To back up a database on an on-premises server, you use the `mysqldump` tool at the command line. That creates a `.sql` file that you restore to the cloud database by passing it to the `mysql` command as a script. If you prefer a GUI tool, choose the **PHPMysqlAdmin** application, or **MySQL Workbench**. These GUI tools can both back up and restore the data.

Remember that Azure Database for MySQL only supports the InnoDB engine. Make sure you convert any MyISAM tables to InnoDB before you execute the backup.

To avoid any compatibility problems, check that the version number of MySQL used in the cloud matches the version number of the on-premises database server. Azure Database for MySQL supports versions 5.6, 5.7, and 8.0. If your on-premises server uses an earlier version, consider upgrading to one of these versions first and troubleshooting any issues on-premises, before you migrate to the cloud.

Plan a backup and restore on PostgreSQL

The equivalent command-line backup and restore tools on PostgreSQL are `pg_dump` and `pg_restore`. There are third party alternatives for a GUI backup and restore tool.

Custom application code

If you have extensive data transformation requirements or want to perform an unusual migration, consider writing your own custom code to move data from an on-premises MySQL or PostgreSQL database into the cloud.

Your custom code could take many forms. The language and framework you choose depends mostly on your development team's expertise:

- SQL Scripts generated from the database and modified or developed from scratch.
- Compiled code from a development framework such as .NET or Java.

- Scripts in PHP or Node.js.
- Shell scripts for Bash or PowerShell.

The custom code approach enables you to be extremely flexible. You customize how data is filtered, aggregated, and transformed, and you can migrate to multiple destinations or merge data from multiple sources. Use this approach if you have requirements that can't be satisfied with an out-of-the-box backup or export tool.

The drawback to this approach is that it requires more investment in development time. For custom code to migrate all the data correctly, it must be extensively tested before it's run on real data. This task requires a team of skilled developers and testers, and often increases the project budget. If you're considering writing custom migration code, don't be tempted to underestimate the time and effort required to create reliable code.

Azure Database Migration Service

Azure includes a flexible service called **Azure Database Migration Service (DMS)**, which you use to do seamless online migrations from multiple data sources into Azure data platforms. These platforms include **Azure Database for MySQL** and **Azure Database for PostgreSQL**.

Consider using Azure DMS whenever you want to perform an online database migration into Azure.

Initial migration

To perform a migration with DMS, you complete these tasks:

1. Create a new target database within Azure on the platform of your choice.
2. Create a new DMS data migration project.
3. Generate the schema from the on-premises source databases. If you're using MySQL, you can generate a schema with `sql_dump`. If the source database is PostgreSQL, use `pg_dump`.
4. Create an empty database to act as the migration destination.
5. Apply the schema to the destination database.
6. Configure connection details for the source and destination databases in a DMS migration project.
7. Run the DMS migration project. The project transfers the data and generates a report.
8. Review the report and correct any issues that it identifies.

Online migrations

Azure DMS is a good tool to use for online migrations, in which the original database remains available while the migration executes. Users continue to make changes to data in the source database. Azure DMS uses replication to synchronize these changes with the migrated database. Once migration is complete, you reconfigure the user applications to connect to the migrated database.

Migrate MySQL to Azure SQL Database

If you want to move a database that's hosted on-premises on a MySQL database server into the Azure cloud—and you don't need the cloud database to run MySQL—consider migrating to Azure SQL Database. Azure SQL Database is a PaaS implementation of Microsoft's industry-leading SQL Server database engine. It includes enterprise-level availability, scalability, and security, and is easy to monitor and manage.

Azure SQL Database is more fully-featured than Azure Database for MySQL.

[NOTE]

You might need to modify any applications that connect to your migrated database—because Azure SQL Database uses different data types, different database objects, and a different API from MySQL. Consult your developers to determine how much work is required to port a client application from an on-premises MySQL database to a cloud Azure SQL database.

SQL Server Migration Assistant for MySQL

If you decide to migrate to Azure SQL Database, you can use a specialized tool: **SQL Server Migration Assistant for MySQL**. This GUI tool connects to a source MySQL database and a SQL Server database, which can be a database in the Azure SQL Database service.

When it's connected, the assistant copies the complete schema to Azure SQL Database, and converts any data types to their SQL Server equivalents. It also migrates views, procedures, triggers, and other objects. You can then start to migrate the data from MySQL to Azure SQL Database.

Lesson 3: Summary

In this lesson, you saw how to:

- Export data from a on-premises system and import it into a cloud database.
- Use a backup and restore technique to migrate data into a cloud database.
- Migrate database schema, content, and security information by developing custom code.
- Use the Azure Database Migration Service to move data.
- Move MySQL data into Azure SQL Database as an alternative to running MySQL in the cloud.

Knowledge Check

Multiple choice

The Azure Database Migration Service automatically transfers a database schema and data to a target database running on a service like Azure Database for MySQL or Azure Database for PostgreSQL. True or false?

- True.
- False.

Multiple choice

How might you migrate an on-premises MySQL database to Azure SQL Database?

- Back up the MySQL database, and restore it to Azure SQL Database.
- Use the SQL Server Migration Assistant.
- Do an online migration using the Azure Database Migration Service.
- Copy the database files for your MySQL database to Azure SQL Database.
- Upload the data to Azure storage, and configure Azure SQL Database to read the data from the storage account.

Module summary

Module Summary

In this module, you learned about the issues and considerations for migrating on-premises open source databases to Azure. You saw the services that Azure provides to help you migrate and host your databases. You looked at what you need to consider when planning a migration project, and you learned about different approaches to migrating databases.

Takeaways

In this module, you learned how to:

- Describe the features and services available in Azure for hosting an open source database.
- Explain the key considerations for implementing a migration project.
- Describe different approaches that you can take for migrating databases.

Answers

Multiple choice

You want to create a highly available database system that requires the minimum of administrative time to run.

Should you use an IaaS or a PaaS approach?

- Use a IaaS approach. Create a set of virtual machines to run your system, and connect them using a virtual network. You have full control over the databases and server running on these virtual machines.
- Use a PaaS approach. For example, run a PostgreSQL database using Azure Database for PostgreSQL.

Explanation

The PaaS approach minimizes the administrative effort required. Microsoft assume responsibility for tasks such as taking backups, and ensuring the availability of your server and databases.

Multiple choice

You want to run a MySQL database in an IaaS environment in which you'll create new instances as quickly as possible.

Should you use virtual machines or containers?

- Use containers.
- Use virtual machines.

Explanation

Containers share an operating system with the host, so they're smaller and start more quickly than virtual machines.

Multiple choice

Offline migration enables users to continue accessing your existing on-premises database while data is copied to a new database in the cloud. Changes made to the existing database during migration will be automatically applied to the new database.

True or false?

- True.
- False.

Explanation

Online migration transfers data while the existing system remains online. Some form of roll-forward replication is often used to transfer updates to the new database during migration.

Multiple choice

You want to migrate individual applications and users running specific workloads to the cloud as an initial step in migrating your system. If successful, you'll migrate further workloads until the entire system has been moved.

How would you do this?

- Perform canary testing.
- Perform an online migration of the database.
- Follow a piecemeal approach to migration.
- Perform an offline migration of the database.
- Replicate your database to a server running on a virtual machine in the cloud.

Explanation

A piecemeal migration divides your complete system into workloads. You migrate one workload at a time.

Multiple choice

The Azure Database Migration Service automatically transfers a database schema and data to a target database running on a service like Azure Database for MySQL or Azure Database for PostgreSQL.

True or false?

- True.
- False.

Explanation

The Azure Database Migration Service transfers data, but you need to have manually set up the schema in the target database beforehand.

Multiple choice

How might you migrate an on-premises MySQL database to Azure SQL Database?

- Back up the MySQL database, and restore it to Azure SQL Database.
- Use the SQL Server Migration Assistant.
- Do an online migration using the Azure Database Migration Service.
- Copy the database files for your MySQL database to Azure SQL Database.
- Upload the data to Azure storage, and configure Azure SQL Database to read the data from the storage account.

Explanation

The SQL Server Migration Assistant provides a step-by-step process for transferring a MySQL schema and database to Azure SQL Database.



Module 2 Migrate on-premises MySQL to Azure Database for MySQL

Module Introduction

Module 2: Migrate on-premises MySQL databases to Azure Database for MySQL

Module introduction

In this module, you'll learn about the benefits of migrating MySQL workloads to Azure, and the features that Azure provides to help manage and optimize MySQL systems. You'll see how to create an instance of the Azure Database for MySQL service, and learn how to migrate on-premises MySQL databases to Azure. You'll also see how to reconfigure an application that uses the database to connect to Azure instead.

Learning objectives

By the end of this module, you'll be able to:

- Describe the features and limitations of Azure Database for MySQL.
- Migrate an on-premises MySQL database to Azure Database for MySQL.
- Reconfigure existing applications that use your on-premises MySQL databases to connect to Azure Database for MySQL.

Introduction to Azure Database for MySQL

Lesson 1: Introduction to Azure Database for MySQL

In this lesson, you'll take a detailed look at Azure Database for MySQL. You'll learn about the features that Azure provides to help protect, manage, and monitor databases running in Azure Database for MySQL.

Learning objectives

By the end of this lesson, you'll be able to:

- Describe the features of Azure Database for MySQL, Single Server version.
- Describe how to create inbound and read-only replicas for Azure Database for MySQL.
- List the tools that Azure Database for MySQL provides to manage and monitor databases.
- Explain the connectivity options for client applications that use databases running in Azure Database for MySQL.
- Summarize the MySQL features that aren't supported by Azure Database for MySQL.

Azure Database for MySQL

Azure Database for MySQL is provisioned as an Azure Database for MySQL server. The Azure Database for MySQL server is equivalent to an on-premises MySQL server and provides a central point to administer multiple MySQL databases.

To create an Azure Database for MySQL database, you must first provision an Azure Database for MySQL Server. An Azure Database for MySQL server is the parent of one or many databases and provides the namespace for the databases. If you delete the server, you'll delete all databases that it contains.

What does Azure Database for MySQL server provide?

The Azure Database for MySQL service includes high availability at no additional cost and scalability as required. You only pay for what you use. Automatic backups are provided, with point-in-time restore.

The server provides connection security to enforce firewall rules and, optionally, require SSL connections. Numerous server parameters enable you to configure server settings such as lock modes, maximum number of connections, and timeouts. Changes to parameters that are marked as *Dynamic* take effect immediately. Static parameters require a server restart. You restart the server using the **Restart** button on the **Overview** page in the portal.

Azure Database for MySQL servers include monitoring functionality to add alerts, and to view metrics and logs.

Pricing tiers

Pricing tiers enable a wide range of performance and capacity from one to 64 vCores and from 5 GB to 4 TB of storage. The basic pricing tier is designed for light compute workloads and supports up to two vCores with 2 GB of memory per core. The general purpose pricing tier will suit most business workloads and supports from two to 64 vCores with 5 GB of memory per core. The memory-optimized pricing tier supports two to 32 vCores, has 10 GB of memory per vCore, and is intended for high performance

workloads, including real-time data analysis. Although you can switch between general purpose and memory-optimized pricing tiers, and change the number of vCores or storage within seconds, you can't move to or from the basic pricing tier.

Pricing tier

Basic	General Purpose	Memory Optimized
Up to 2 vCores with Variable IO performance (1-2 vCores)	Up to 64 vCores with predictable IO performance (2-64 vCores)	Up to 32 memory optimized vCores with predictable IO performance (2-32 vCores)

Info: Please note that changing to and from the Basic pricing tier or changing the backup redundancy options after server creation is not supported.

Compute Generation - [Learn more about compute generation](#)

Gen 5

vCore - [What is a vCore?](#)

4 vCores

Would you like to configure larger storage (Public Preview)? - [Learn more](#)

Storage (type: [General Purpose Storage](#))

599 GB

CAN USE UP TO **1797** available IOPS

Auto-growth - [Learn More](#)

Yes No

Backup Retention Period

28 Days

Backup Redundancy Options - [Learn more details](#)

Locally Redundant <input checked="" type="checkbox"/> Recover from data loss within region	Geo-Redundant Recover from regional outage or disaster
---	---

There are connection limits based on pricing tiers and the number of vCores. See **Limitations in Azure Database for MySQL**¹ for more information.

Versioning and upgrades

Azure Database for MySQL supports version 5.6 (with bug fix release 5.6.42), 5.7 (with bug fix release 5.7.24), and 8.0 (with bug fix release 8.0.15).

[NOTE]

A gateway redirects connections to server instances. MySQL clients will display the version of the gateway rather than the version of the server instance.

To view the version of the server instance, use the `SELECT VERSION();` command.

¹ <https://docs.microsoft.com/en-us/azure/mysql/concepts-limits>

Bug fix versions are applied automatically, but version upgrades are not supported. To upgrade from one version to another, you should perform a dump and restore.

Scalability

As mentioned, you can't change to or from the basic pricing tier. However, you can alter the number of vCores, the hardware generation, the storage volume, and the backup retention period. You can also switch between the general purpose and memory-optimized pricing tiers.

Note that storage is only increased, not decreased, and can be set to auto-grow. If auto-grow is enabled, storage grows by 5 GB when available storage is less than 1 GB or 10% of storage volume (whichever is greater) for servers with less than 100 GB of storage. For servers with more than 100 GB, storage increases by 5% when available storage is less than 5%.

High availability

Azure Database for MySQL includes a financially-backed service level agreement (SLA) for availability of 99.99%. If there's a hardware failure or service deployment, a new node is automatically created, and storage is attached to this node. Failover will be complete within tens of seconds.

If an Azure Database for MySQL server instance is scaled up or scaled down, a similar process occurs with the data storage being attached to the new instance. If a failover takes place, a scale up or scale down occurs, or there's any interruption in internet traffic between the client and Azure, a transient connectivity error might happen at the client. It's important to have retry logic in applications. In the case of a failover, a gateway will direct traffic to the new node with no configuration required at the client.

For information on handling transient errors, see **Handling of transient connectivity errors for Azure Database for MySQL**².

Replicate data in Azure Database for MySQL

Data-in Replication

Data-in Replication uses the native replication functionality of MySQL to replicate data from an external MySQL Server into Azure Database for MySQL. This is useful if you want to provision a hybrid environment with an existing on-premises MySQL instance and an Azure-based replica. This scenario provides local data to users in a globally distributed system. You could also use Data-in Replication to replicate data from a virtual machine or MySQL database service hosted by another cloud provider.

Considerations for Data-in Replication

Here are some factors to consider for Data-in Replication:

- The master and replica servers must be the same version—and at least version 5.6.
- The master and replica should use the InnoDB engine.
- Every table must have a primary key.
- The Azure Database for MySQL server must have a **General Purpose** or **Memory Optimized** pricing tier.
- You should have the rights to create users and configure binary logging on the master server.

² <https://docs.microsoft.com/en-us/azure/mysql/concepts-connectivity>

- The **mysql system database** is not replicated. Accounts and permissions are not replicated from the master server to the replica, and should be created manually.

Steps to configure Data-in Replication

There's a number of steps to configure Data-in Replication:

- Create an Azure Database for MySQL Server to be used as a host for the replica, and create any necessary user accounts and privileges.
- Configure replication on the master server.
- Dump and restore the master server.
- Use Data-in Replication stored procedures to configure the target server.

See **How to configure Azure Database for MySQL Data-in Replication**³ for more information.

Read replicas

Read replicas use native MySQL replication technology to create asynchronous replica instances of Azure Database for MySQL servers. The replica servers are read-only and there can be up to five replicas for each master. For each read replica, the monthly cost is billed based on the vCores and storage it uses.

Uses for read replicas

Reporting servers

By creating a read-only replica of the master server, you direct all reporting, BI, and analytical workloads to the replica. This removes the workload from the master server and reduces conflicts while the master server runs its write-intensive workloads.

Bringing data close to users

You create cross-region replicas to bring data close to users and improve their read speeds. Cross-region replicas can be in a universal replica region or the paired region of the master server. The available regions are listed when you create a replica server.

³ <https://docs.microsoft.com/en-us/azure/mysql/howto-data-in-replication>

MySQL server □ ×

* Server name

* Location
 ^

Recommended ⓘ

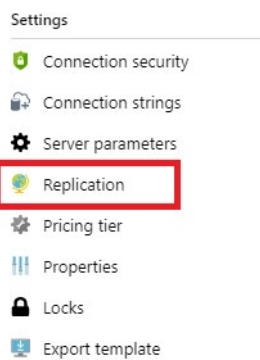
- (US) East US
- (US) East US 2
- (US) South Central US
- (US) West US 2
- (US) Central US
- (US) North Central US
- (US) West US
- (Asia Pacific) Australia East
- (Asia Pacific) Southeast Asia
- (Asia Pacific) East Asia
- (Asia Pacific) Japan East
- (Asia Pacific) Korea Central
- (Europe) North Europe
- (Europe) UK South
- (Europe) West Europe

Other ⓘ

- (Asia Pacific) Australia Central
- (Asia Pacific) Australia Central 2
- (Asia Pacific) Australia Southeast
- (Asia Pacific) Japan West
- (Asia Pacific) Korea South
- (Europe) UK West

Configure read replicas

You configure a read replica in the Azure portal:




You then specify the name and region of the replica:

MySQL server □ ×

* Server name
 ✓

* Location
 ▼

[Supported Locations](#)

* Pricing tier
General Purpose, 2 vCore(s), 5 GB 

Monthly cost **171.46** GBP

[Automation options](#)

[NOTE]

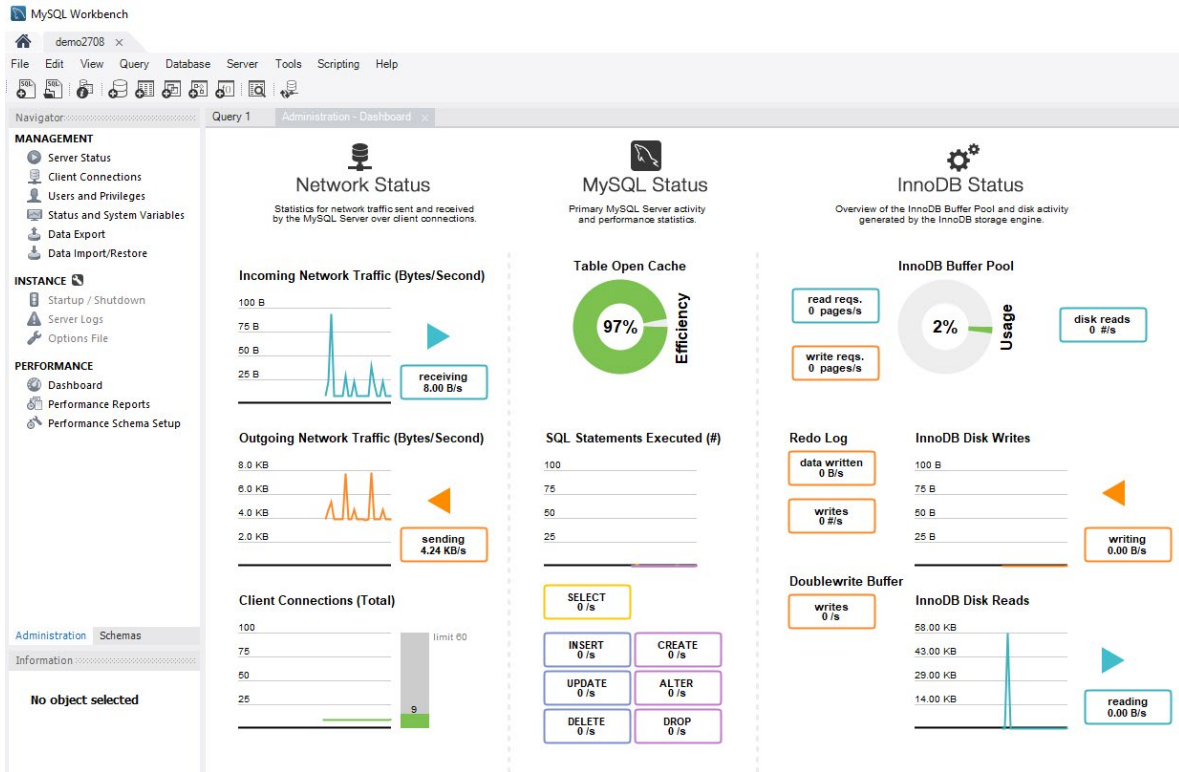
Read replicas aren't available in the basic pricing tier.

For more information on read replicas, see [Read replicas in Azure Database for MySQL](#)⁴.

Management and monitoring

Azure Database for MySQL has a wide array of monitoring tools to help you optimize your server, be notified of events, and proactively respond to metrics. You can also use familiar MySQL administration tools, such as recent versions of MySQL Workbench, PHPMyAdmin, and Navicat, to manage and monitor Azure Database for MySQL servers:

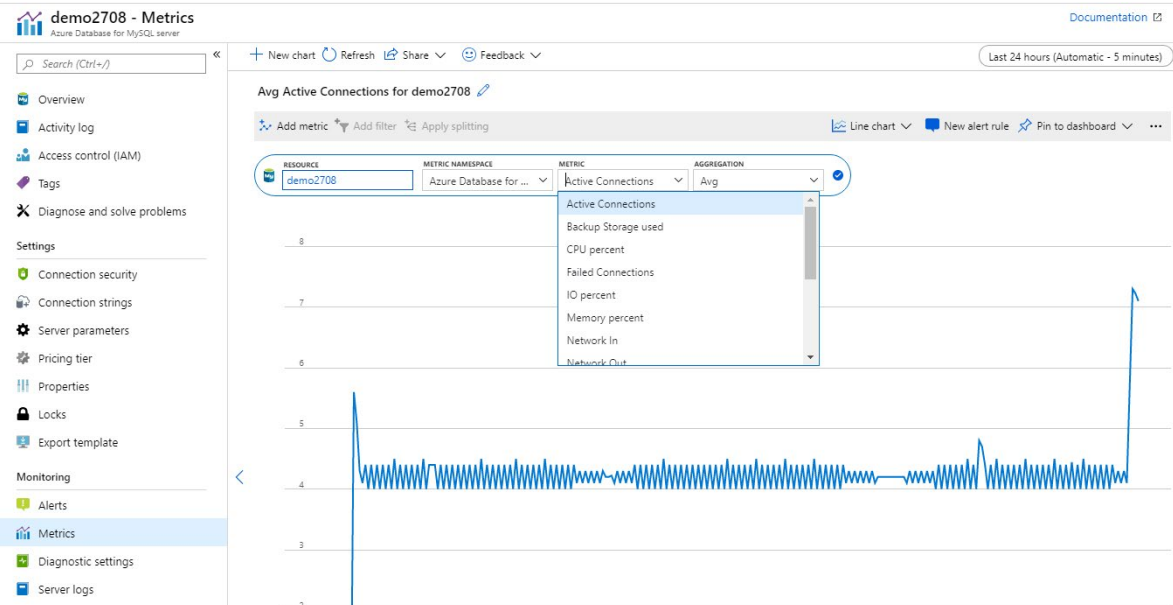
⁴ <https://docs.microsoft.com/en-us/azure/mysql/concepts-read-replicas>



Azure tools for monitoring Azure Database for MySQL

The tools available in the Azure portal for managing and monitoring Azure Database for MySQL include the following:

- Azure metrics.** Metrics provide numeric data every minute and are stored for 30 days. There's a wide array of metrics that you use to monitor your server—you can also configure alerts to respond to metrics.



See **Azure Monitor data platform**⁵ for more information.

- **Server and audit logs.** You enable server logs to monitor slow queries and provide audit logging for your server. Server logs are available outside SQL Database for MySQL through Azure Diagnostic Logs.

Server parameters
Allows changing of MySQL server parameters

Save Discard

Search to filter items...

PARAMETER NAME	VALUE	PARAMET...	DESCRIPTION
log_bin_trust_function_creators	OFF	Dynamic	This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that ...
log_queries_not_using_indexes	ON	Dynamic	Logs queries that are expected to retrieve all rows to slow query log.
log_slow_admin_statements	OFF	Dynamic	Include slow administrative statements in the statements written to the slow query log.
log_slow_slave_statements	ON	Dynamic	When the slow query log is enabled, this variable enables logging for queries that have taken more than long_query_time seconds to execute on the...
log_throttle_queries_not_using_indexes	10	Dynamic	Limits the number of such queries per minute that can be written to the slow query log.
long_query_time	10	Dynamic	If a query takes longer than this many seconds, the server increments the Slow_queries status variable.
slow_query_log	ON	Dynamic	Enable or disable the slow query log

See **Slow query logs in Azure Database for MySQL**⁶ for more information.

Audit logs are a preview feature to provide audit logging to track database activity. To turn on audit logging, set the **audit_log_enabled** parameter to **ON**.

For more information on audit logs, see **Audit Logs in Azure Database for MySQL**⁷.

- **Query Store.** This is used to track the performance of your server over time and give troubleshooting information. Query Store retains query history and run-time statistics so you can identify resource intensive or long running queries.

To enable Query Store, set the **query_store_capture_mode** server parameter to **ALL**:

Search (Ctrl+/)

Save Discard Reset all to default

query_store_capture_mode	ALL
query_store_capture_utility_queries	NO
query_store_retention_period_in_days	7
query_store_wait_sampling_capture_mode	NONE
query_store_wait_sampling_frequency	30
range_alloc_block_size	4096
range_optimizer_max_mem_size	8388608
relay_log_space_limit	1073741824
server_id	2271663710
session_track_schema	ON

To view query store data about queries, run the following query:

```
SELECT * FROM mysql.query_store;
```

To view data about wait statistics, run the following query:

⁵ <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-platform>

⁶ <https://docs.microsoft.com/en-us/azure/mysql/concepts-server-logs>

⁷ <https://docs.microsoft.com/en-us/azure/mysql/concepts-audit-logs>


```
SELECT * FROM mysql.query_store_wait_stats;
```

[NOTE]

Query Store is a preview feature and is not available in the basic pricing tier.

For more information on Query Store, see **Monitor Azure Database for MySQL performance with Query Store**⁸.

[comment]: <> Query Store currently not in Azure Database for MySQL

- **Query Performance Insight.** Query Performance Insight displays data from Query Store as visualizations to enable you to identify queries that affect performance. Query Performance Insight is in the **Intelligent Performance** section of your Azure Database for MySQL, in the Azure portal.

[NOTE]

Query Performance Insight is a preview feature and is not available in the basic pricing tier.

For more information on Query Performance Insight, see **Query Performance Insight in Azure Database for MySQL**⁹.

- **Performance Recommendations.** Performance Recommendations uses data from the Query Store to analyze workloads, and combines this with database characteristics to suggest new indexes to improve performance. Performance Recommendations is in the **Intelligent Performance** section of your Azure Database for MySQL in the Azure portal.

[NOTE]

Performance Recommendations is a preview feature and is not available in the basic pricing tier.

For more information on Performance Recommendations, see **Performance Recommendations in Azure Database for MySQL**¹⁰.

Client connectivity

MySQL drivers

Azure Database for MySQL uses the MySQL community edition and is compatible with a wide range of drivers—it supports a variety of programming languages. Connection strings are provided in the Azure portal:

⁸ <https://docs.microsoft.com/en-us/azure/mysql/concepts-query-store>

⁹ <https://docs.microsoft.com/en-us/azure/mysql/concepts-query-performance-insight>

¹⁰ <https://docs.microsoft.com/en-us/azure/mysql/concepts-performance-recommendations>

For more information on MySQL drivers, see **MySQL drivers and management tools compatible with Azure Database for MySQL**¹¹

Configure the firewall

The simplest way to configure the firewall is to use the Connection Security settings for your service in the Azure portal. Add a rule for each client IP address range. You can also use this page to enforce SSL connections to your service.

You click **Add Client IP** in the toolbar to add the IP address of your desktop computer.

If you've configured read-only replicas, you must add a firewall rule to each one to make them accessible to clients.

¹¹ <https://docs.microsoft.com/en-us/azure/mysql/concepts-compatibility>

Transient connection errors

When you connect to a database over the internet, transient connection errors are inevitable and should be handled by client applications.

For information on transient connectivity errors, see **Handling of transient connectivity errors for Azure Database for MySQL**¹².

MySQL features that aren't supported in Azure Database for MySQL

While most features in MySQL are available in Azure Database for MySQL, some aren't supported. You should review these features to ensure you mitigate any potential issues when migrating.

Storage engines

Azure Database for MySQL supports the InnoDB and MEMORY storage engines. InnoDB is the default storage engine for MySQL, providing a balance between high performance and high reliability. All new tables in MySQL will use the InnoDB storage engine unless specified otherwise.

For more information on the InnoDB storage engine, see **Introduction to InnoDB**¹³.

To store data in memory, the MEMORY storage engine is available. This data is at risk from any form of crash or outage—the MEMORY storage engine should only be used as a temporary, high performance store.

For more information on the MEMORY storage engine, see **The MEMORY Storage Engine**¹⁴.

MyISAM, BLACKHOLE, ARCHIVE, and FEDERATED storage engines are not supported in Azure Database for MySQL. MyISAM data should be converted to the InnoDB storage engine. BLACKHOLE, ARCHIVE, and FEDERATED storage engines have specialist roles and are not used as typical data stores.

Privileges and roles

The **DBA** role is not exposed because many server settings and parameters can break transaction rules and degrade performance. For similar reasons, the **SUPER** privilege is restricted, as is the **DEFINER** clause that uses the **SUPER** privilege.

Restore

Two restore features function differently in Azure Database for MySQL:

- Point-in-time restore creates a new server with an identical configuration to the server that it's based on.
- You can't restore a deleted server.

¹² <https://docs.microsoft.com/en-us/azure/mysql/concepts-connectivity>

¹³ <https://dev.mysql.com/doc/refman/5.7/en/innodb-introduction.html>

¹⁴ <https://dev.mysql.com/doc/refman/5.7/en/memory-storage-engine.html>

Lesson 1: Summary

In this lesson, you saw how to:

- Describe the features of Azure Database for MySQL, Single Server version.
- Describe how to create inbound and read-only replicas for Azure Database for MySQL.
- List the tools that Azure Database for MySQL provides to manage and monitor databases.
- Explain the connectivity options for client applications that use databases running in Azure Database for MySQL.
- Summarize the MySQL features that aren't supported by Azure Database for MySQL.

Knowledge Check

Multiple choice

With the increasing risk of cyber attacks, you're concerned that your database servers have the latest bug fixes.

How can you ensure that your Azure Database for MySQL servers have the latest bug fix updates applied?

- Review the Azure Performance Recommendations and apply bug fixes when suggested.
- Set an alert for bug fix updates and create a runbook to apply the updates automatically.
- Bug fix updates are automatically applied. No action needs to be taken.
- Perform regular upgrades when a new version of Azure Database for MySQL becomes available.
- Use the Advanced Threat Protection feature to ensure bug fixes are applied.

Multiple choice

You want to keep your writeable on-premises version of a MySQL database, but create replica read-only copies for users in another country.

What's the most straightforward method to achieve this with Azure Database for MySQL?

- Create an Azure Database for MySQL instance in the region closest to your users and configure Data-In Replication.
- Create a read replica copy of your database located in the region closest to your users.
- Create a writeable Azure Database for MySQL instance in the region closest to your users, and create a read-only replica in your on-premises MySQL server.
- Create a writeable Azure Database for MySQL instance in the region closest to your users, and create a writeable replica in your on-premises MySQL server.

Migrate on-premises MySQL database to Azure

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

In this lesson, you'll see how to migrate a MySQL database from on-premises to Azure Database for MySQL. You'll learn how to perform an online migration using the Azure Database Migration Service, and how to do an offline migration by manually transferring data from on-premises databases to Azure Database for MySQL.

Learning objectives

By the end of this lesson, you'll be able to:

- Create a MySQL server using Azure Database for MySQL.
- Perform an online migration of a MySQL database to Azure Database for MySQL using the Azure Database Migration Service.
- Perform an offline migration using MySQL tools.

Creating a MySQL server with Azure Database for MySQL

You can create an instance of the Azure Database for MySQL service using the Azure portal. If you need to create a number of instances of this service, you script the process by using the Azure CLI.

Create an Azure Database for MySQL using the portal

In the Azure portal, select the **Databases** command in **Azure Marketplace**, and select **Azure Database for MySQL**.

The screenshot shows the Azure Marketplace interface. On the left, a sidebar lists various categories, with 'Databases' highlighted. The main content area is divided into 'Azure Marketplace' and 'Featured' sections. Under 'Featured', several database services are listed, including 'Azure SQL Managed Instance', 'SQL Database', 'SQL Data Warehouse', 'Azure Database for MariaDB', 'Couchbase Enterprise Edition (Hourly Pricing) (preview)', and 'Azure Database for MySQL'. The 'Azure Database for MySQL' entry is highlighted with a red rectangular box.

Category	Service Name	Action
Featured	Azure SQL Managed Instance	Quickstart tutorial
	SQL Database	Quickstart tutorial
	SQL Data Warehouse	Quickstart tutorial
	Azure Database for MariaDB	Learn more
	Couchbase Enterprise Edition (Hourly Pricing) (preview)	Learn more
	Azure Database for MySQL	Quickstart tutorial
	Internet of Things	

Enter the details for the service. These include:

- **Server name.** This must be a unique name between 3 and 63 characters, containing only lowercase letters, numbers, and hyphens.
- **Data source.** If you're creating a new server for migration purposes, select **None**. The **Backup** option enables you to restore a backup taken from another instance of Azure Database for MySQL into this service.
- **Admin username.** This is the name of a user account that will be created with administrative privileges. Azure creates some accounts for its own use, and other names are restricted; you can't use **azure_superuser**, **admin**, **administrator**, **root**, **guest**, or **public**.
- **Password.** This must be between 8 and 128 characters. It should contain a mixture of uppercase and lowercase letters, numbers, and nonalphanumeric characters. Azure Database for MySQL currently only supports password authentication; integration with Azure Active Directory isn't available yet.
- **Version.** Select the version that corresponds to the on-premises database that you're migrating.
- **Compute + storage.** Select **Configure server** to set the pricing tier and specify the resources that you require for the service. The options were covered in Lesson 1. Remember that, if you select the **General purpose** or **Memory optimized** pricing tiers, you scale up and down the number of virtual processor cores later. However, you can't reduce the amount of storage; it can only increase after the server has been created.

Create MySQL server

Microsoft

[Basics](#) [Tags](#) [Review + create](#)

Create an Azure Database for MySQL server. [Learn more](#)

Project details

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

* Subscription ⓘ

* Resource group ⓘ [Create new](#)

Server details

Enter required settings for this server, including picking a location and configuring the compute and storage resources.

* Server name ⓘ ✓

* Data source ⓘ None Backup

* Admin username ⓘ ✓

* Password ⓘ ✓

* Confirm password ⓘ ✓

* Location ⓘ ✓

* Version ⓘ ✓

Compute + storage ⓘ **General Purpose**
4 vCores, 100 GB storage
[Configure server](#)

[Review + create](#)

[Next : Tags >](#)

Select **Review + Create** to deploy the service. Deployment will take several minutes.

After the service has been deployed, select the **Connection security** option and add the appropriate firewall rules to enable clients to connect—as described in the topic *Client Connectivity* in Lesson 1. You must also select the **Allow access to Azure services** option.

Create an Azure Database for MySQL instance using the Azure CLI

You create an instance of Azure Database for MySQL using the `az mysql server create` command. The statement below shows an example that creates a MySQL instance. Most of the parameters are self-explanatory, except for the following:

- **sku-name.** You construct this from a combination of the pricing tier (*B* for Basic, *GP* for General Purpose, and *MO* for Memory Optimized), the compute generation (Gen4 or Gen5), and the number

of virtual CPU cores. In the example below, the server is created using the General Purpose pricing tier, with four CPU cores of the Gen5 generation.

- **storage-size.** This is the amount of disk storage required, specified in megabytes. The example below allocates 10 gigabytes.

```
az mysql server create \
  --name contoso-MySQL-server \
  --resource-group MySQLrg \
  --admin-user contosoadmin \
  --admin-password 7Hh7*ku5k$$£jhk \
  --sku-name GP_Gen5_4 \
  --storage-size 10240
```

Perform online migration

You can perform an online migration from an on-premises MySQL installation to Azure Database for MySQL with the Azure Database Migration Service.

In the online scenario, the Azure Database Migration Service copies all of your existing data to Azure, and then continuously performs a synchronization operation from the source database. Any new transactions that are performed against the on-premises system will be copied to the new database in Azure. This process continues until you've reconfigured your client applications to use the new database in Azure—you then terminate the synchronization operation.

Prerequisites

The following prerequisites are necessary for an online migration:

- All tables must use the InnoDB storage engine. If you have tables that use MyISAM storage, you must convert them before migration.

For more information, see [Converting Tables from MyISAM to InnoDB¹⁵](#)

Configure the source server and export the schema

The first step in performing an online migration is to prepare the source server to support binary logging. On the source server, edit the `my.ini` (Windows) or `my.cnf` (Unix) file and configure the following parameters. To change these parameters, you must restart the server, so you should only do this task when the system is expected to be quiescent:

```
server_id = 1 or greater (relevant only for MySQL 5.6)
log-bin = <path> (relevant only for MySQL 5.6) For example: log-bin = E:\MySQL_logs\BinLog
binlog_format = row
Expire_logs_days = 5 (it's recommended to not use zero; relevant only for MySQL 5.6)
Binlog_row_image = full (relevant only for MySQL 5.6)
log_slave_updates = 1
```

After you've restarted the server, export the schema for the source database using the `mysqldump` utility:

```
mysqldump -h [servername] -u [username] -p[password] [database name] --no-data > db_schema.sql
```

¹⁵ <https://dev.mysql.com/doc/refman/5.7/en/converting-tables-to-innodb.html>

Remove all foreign key references in the exported schema file. This step is required because the data won't necessarily be migrated in any specific sequence—this could cause referential integrity violations which, in turn, might mean that the migration process fails. However, you should make a record of all the foreign keys as you'll need to recreate them later.

Run the following SQL statement in MySQL Workbench to find all the foreign keys in your database, and generate a script that removes them—and a script to recreate them after migration. Run the DropQuery and save the AddQuery for later execution:

```
SET group_concat_max_len = 8192;
  SELECT SchemaName, GROUP_CONCAT(DropQuery SEPARATOR '\n') as DropQuery, GROUP_CON-
CAT(AddQuery SEPARATOR '\n') as AddQuery
  FROM
  (SELECT
    KCU.REFERENCED_TABLE_SCHEMA as SchemaName,
    KCU.TABLE_NAME,
    KCU.COLUMN_NAME,
    CONCAT('ALTER TABLE ', KCU.TABLE_NAME, ' DROP FOREIGN KEY ', KCU.CONSTRAINT_NAME) AS
DropQuery,
    CONCAT('ALTER TABLE ', KCU.TABLE_NAME, ' ADD CONSTRAINT ', KCU.CONSTRAINT_NAME, ' FOREIGN
KEY (' , KCU.COLUMN_NAME, ') REFERENCES ', KCU.REFERENCED_TABLE_NAME, ' (' , KCU.REFERENCED_
COLUMN_NAME, ') ON UPDATE ',RC.UPDATE_RULE, ' ON DELETE ',RC.DELETE_RULE) AS AddQuery
  FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE KCU, information_schema.REFERENTIAL_
CONSTRAINTS RC
  WHERE
    KCU.CONSTRAINT_NAME = RC.CONSTRAINT_NAME
    AND KCU.REFERENCED_TABLE_SCHEMA = RC.UNIQUE_CONSTRAINT_SCHEMA) Queries
GROUP BY SchemaName;
```

Create a target database and import the schema

The next stage is to create a target database in your Azure Database for MySQL service. You can use a familiar tool like MySQL Workbench to connect to the server, or use the Azure CLI as shown in the following example:

```
az mysql db create \
--name [database name] \
--server-name [server name] \
--resource-group [azure resource group]
```

Import the schema into the target database. On the machine holding the db_schema.sql file, run the following command:

```
mysql -h [Azure Database for MySQL host] -U [user name] [database name] < db_schema.sql
```

Disable any triggers in the target database—there are two reasons to do this:

- It helps to optimize the migration process as data is copied in.
- Triggers are often used to implement complex forms of referential integrity. For the reasons described earlier, this type of integrity checking could fail while data is transferred. Use the following SQL statement to find all the triggers in your database and generate a script that disables them:

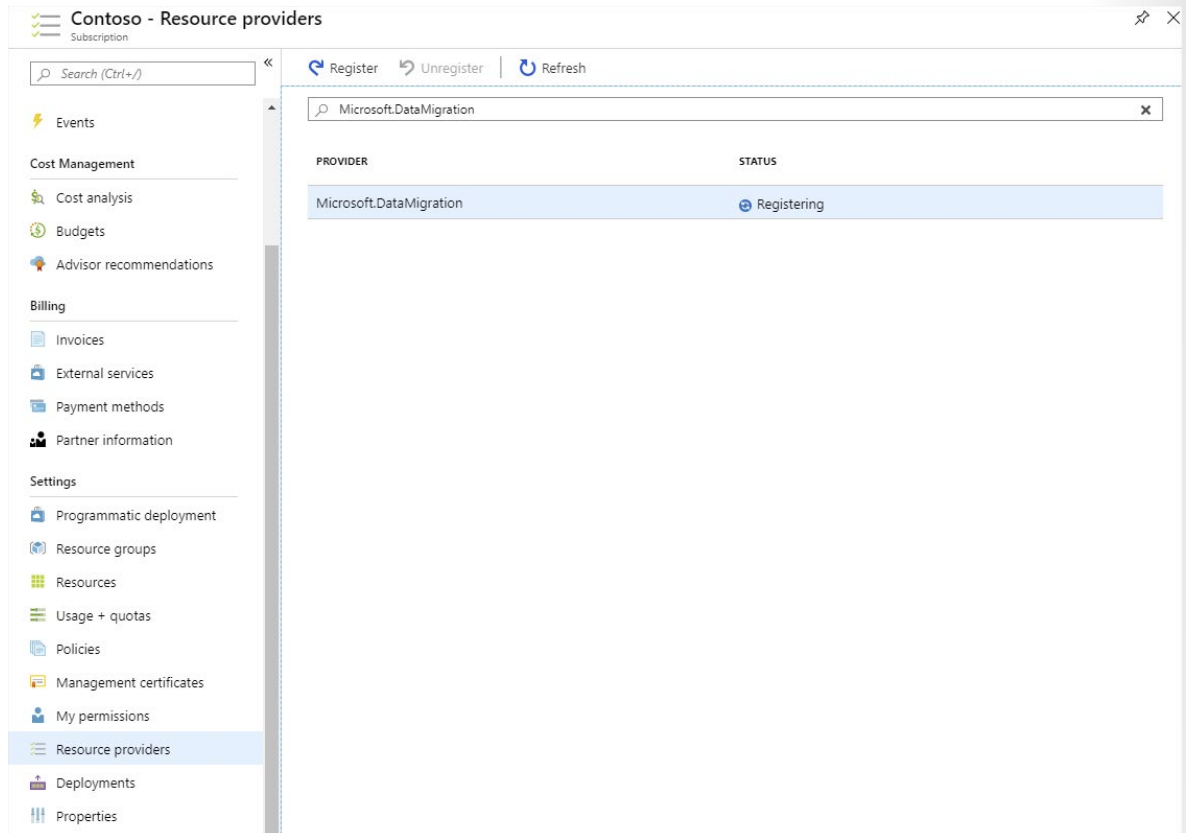
```
SELECT Concat('DROP TRIGGER ', Trigger_Name, ';') FROM information_schema.TRIGGERS;
```

Create an Azure Database Migration Service instance

In the Azure portal, you can now create an instance of the Azure Database Migration Service.


Before you create the Azure Database Migration Service instance, you must register the **Microsoft.DataMigration** resource provider with your subscription. You do this as follows:

1. In the left menu bar of the Azure portal, select **All services**.
2. On the **All services** page, select **Subscriptions**.
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. Select **Register**, and wait for the **Status** to change to **Registered**. You might need to select **Refresh** to see the status to change.



When the resource provider has been registered, you create the service. Select the **Create a resource** command in the left menu bar, and search for **Azure Database Migration Service**. Select **Create**.

Azure Database Migration Service
Microsoft



Azure Database Migration Service [Save for later](#)

Microsoft

[Create](#)

Once you've completed the step by step guidance for your migration scenario, proceed with creating a migration service by clicking the **Create** button below.

Additional resources:

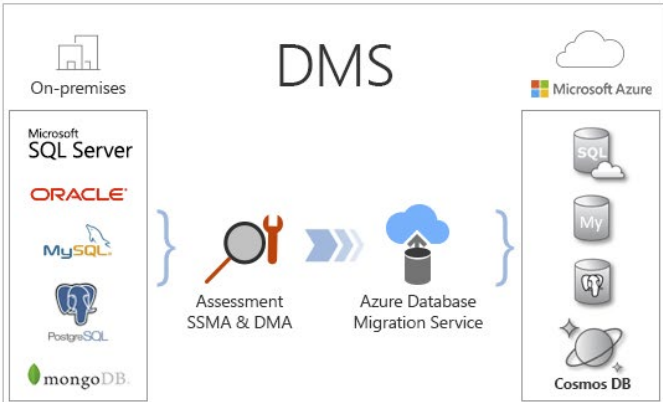
Use these tools to assess your database(s) for feature parity and potential compatibility issues.

- SQL Server on-premise database(s): [Data Migration Assistant \(DMA\)](#)
- Migrating from Oracle: [SQL Server Migration Assistant \(SSMA\)](#)

Useful Links

[Documentation](#)

[Privacy Statement](#)



On the **Create Migration Service** page, enter a name for your instance of the service, specify the subscription—this should be the same subscription that you registered the resource provider against—resource group, and location. You must also provide a virtual network; the Database Migration Service depends on a firewall created for this virtual network to provide the necessary protection. If you're migrating databases from Azure virtual machines, you might be able to place the Database Migration Service in the same virtual network used by these virtual machines. The pricing tier determines the number of virtual processor cores that will be available to the service. If you want to perform an online migration, select the **Premium** tier; the **Standard** tier only supports offline migration.

Create Migration Service

Service Name: mysqlmigrate

Subscription: CM Azure Subscription

Select a resource group: mysql

Location: UK South

Virtual network: MySQL_Migration_vnet/default

Pricing tier: Premium: 4 vCores

Pricing tier

The selected tier supports both offline and online migrations.

Standard	Premium
For large data sizes 1 vCores, 2 vCores, 4 vCores	For offline and online migrations with minimal downtime 4 vCores

vCores: 4 vCores \$0.00 USD/hour

Est. monthly cost: \$0.00 USD

Use the 4 vCore Premium SKU for free for the first 6 months (183 days) from Azure Database Migration Service creation before billing starts.

Wait for the service to be deployed before continuing. This operation will take a few minutes.

Create a migration project using the Database Migration Service

You can now use your Database Migration Service instance for an online migration. To do this, you create a new Database Migration project. Go to the page for your migration service instance and select **New Migration Project**.

mysqlmigrate
Azure Database Migration Service

Search (Ctrl+/)

+ New Migration Project Delete service Refresh Start Service Stop Service

Great job! Your database migration service was successfully created. You can create your first migration project now.

Resource group: mysql Status: Online

Virtual network & IP Address: MySQL_Migration_vnet/subnets/default 10.0.0.4 Location: UK South

Subscription: CM Azure Subscription Subscription ID:

SKU: Premium: 4 vCores Service/UI Version: 4.4.4499.1/4.4.4504.1

NAME	SOURCE	TARGET	CREATED
No database migration projects to display			

Great job! Your database migration service was successfully created. You can create your first migration project now.

New migration project

On the **New migration project** page, set the source server type to **MySQL**, set the target server type to **Azure Database for MySQL**, and select **Online data migration**. The **Type of activity** page lists the steps you must take on the source server to enable online migration. The text at the bottom of the **New migration project** page describes the process for migrating the schema to the target.

New migration project

Project name
migratemySQL ✓

* Source server type
MySQL

* Target server type
Azure Database for MySQL

* Choose type of activity
Online data migration >

To successfully use Database Migration Service (DMS) to migrate data, you need to:

1. Create the target Azure Database for MySQL.
2. Deploy schema, indexes and routines to target database:
 1. Using MySQL Workbench OR
 2. Using mysqldump --no-data

[Install MySQL Workbench](#)

Type of activity

Choose type of activity
Online data migration

Use this option to migrate databases that must be accessible and continuously updated during migration.

To continuously replicate data changes from your source to your target, please implement the steps below on your source server. These steps can be implemented anytime between the moment you create a project and the moment you start a migration activity. After your migration is complete, you will reverse those preparation steps.

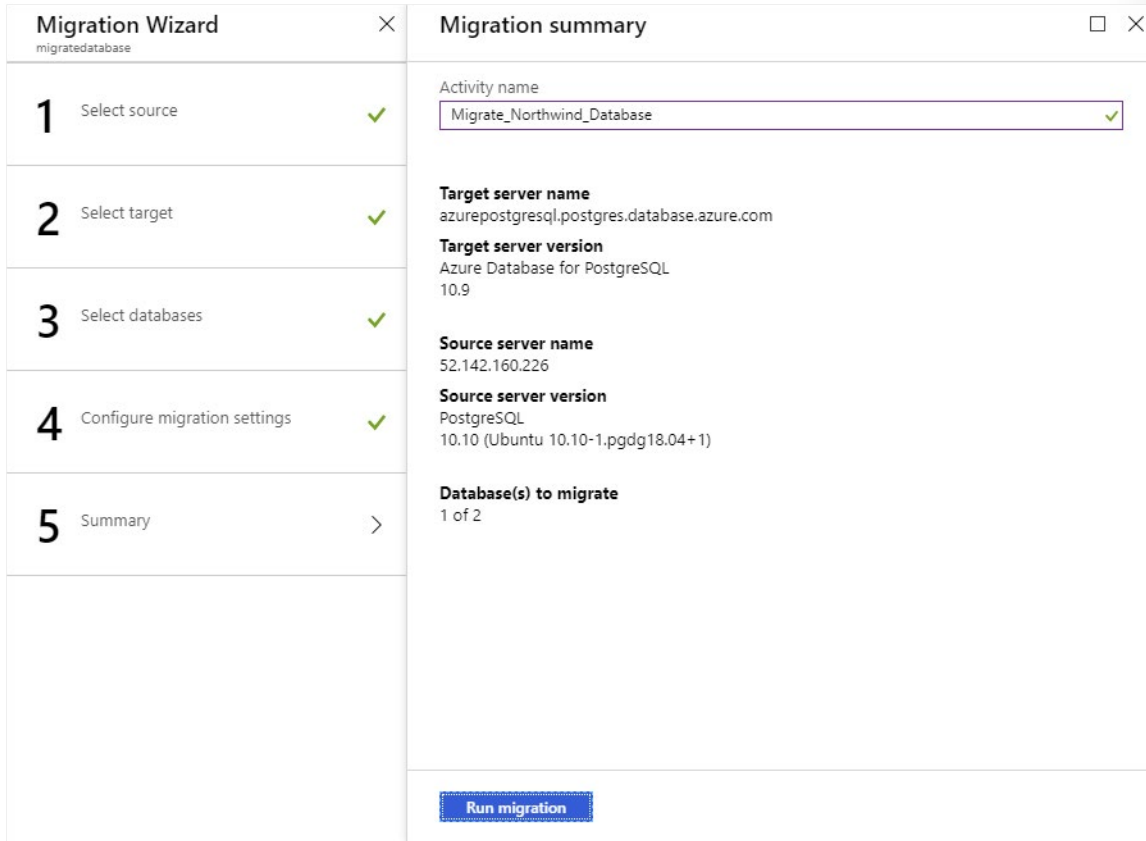
- MySQL server source must match the version that Azure Database for MySQL supports.
- Azure Database for MySQL supports - MySQL community edition, InnoDB engine and migration across source and target with same versions.
- Enable binary logging in my.ini (Windows) or my.cnf (Unix).
- User must have ReplicationAdmin role.

Verify that you've completed these steps, then select **Create and run activity**

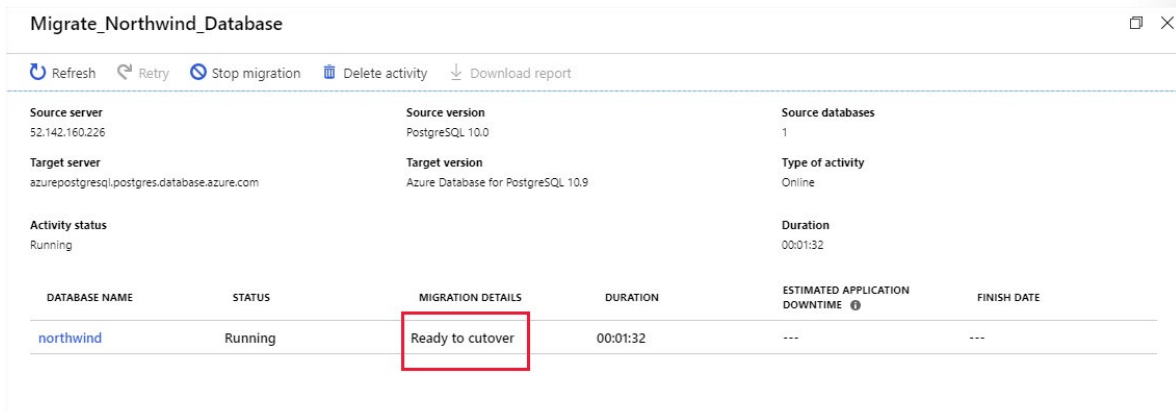
Create and run a migration activity

The new migration project starts a wizard that guides you through the process. You provide the following details:

- On the **Add Source Details** page, add the address of the source server, the source database, and an account that can connect to this database to retrieve the data. The account must have **SUPERUSER** privileges to perform migration.
- On the **Target details** page, specify the address of your Azure Database for MySQL service, the database into which you want to migrate the data, and the details of an account that has administrative rights.
- On the **Map to target databases** page, select the source database and target database. You can migrate a single database or multiple databases.
- On the **Migration settings** page, specify any additional settings that you want to configure, such as the maximum number of tables to load in parallel.
- On the **Migration summary** page, enter a name for the activity, and then select **Run migration**.



The activity status page appears showing the progress of the migration, and any errors that have occurred. If the migration fails, you can correct the issues and retry the activity. If you're doing an online migration, the status changes to **Ready to cutover** when the existing data has been transferred. However, the activity continues running, to transfer any additional changes that appear while applications are still actively using the original database.



Reinstate foreign keys and triggers

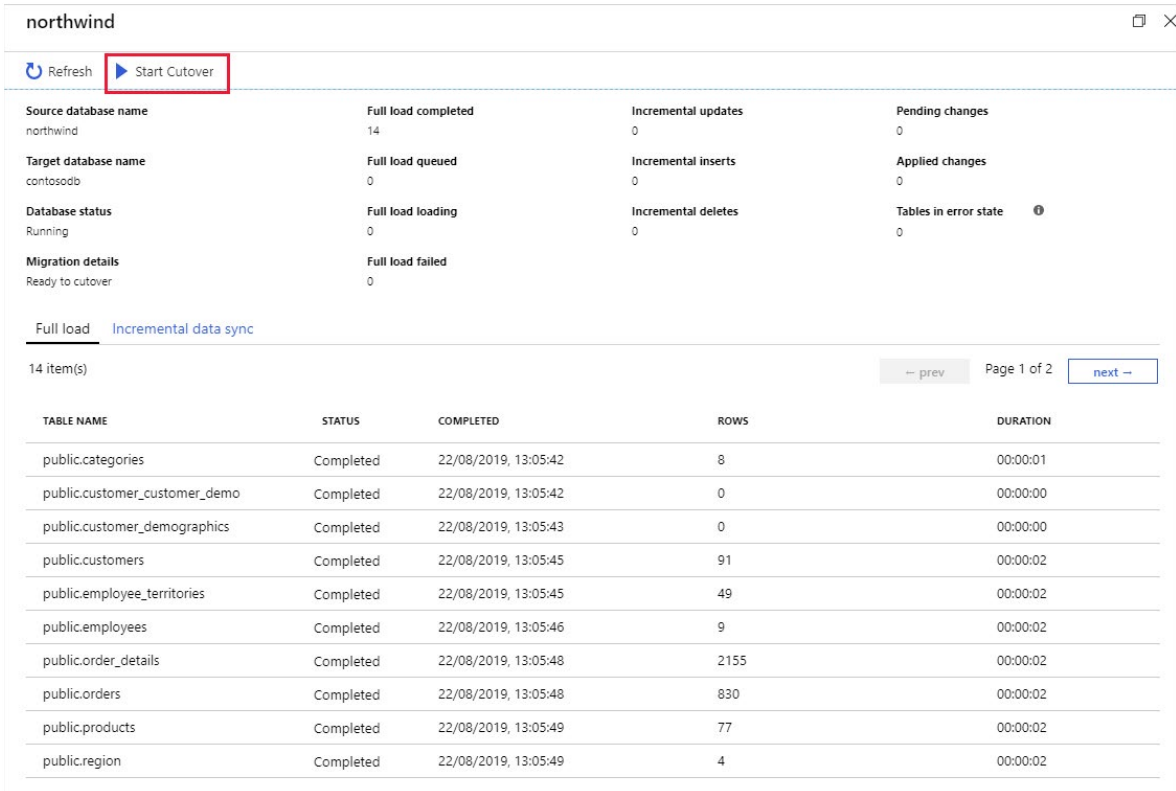
You've now transferred the data, and applications can start using it. You should recreate the foreign keys that you removed before migrating the data, and reinstate any triggers. If some applications are still

connected to the original database, write-ahead logging ensures that the target database in Azure is kept up to date. Write-ahead logging won't be adversely affected by foreign keys and triggers.

If your database contains any materialized views, you'll need to populate them using the `REFRESH MATERIALIZED VIEW` SQL statement.

Cut over to the new database

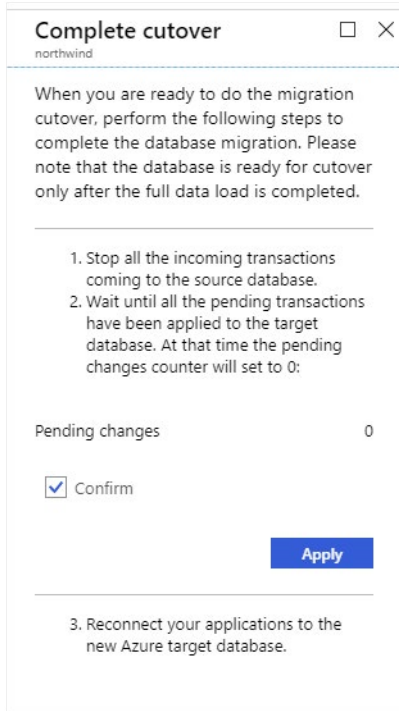
When all applications have been switched to the new database, you can complete the migration process and cut over to the new database. On the activity status page, select the name of the database being migrated to see a summary of the work performed.



The screenshot shows the migration progress for the 'northwind' database. The 'Start Cutover' button is highlighted with a red box. Below the summary, a table lists 14 items with columns for TABLE NAME, STATUS, COMPLETED, ROWS, and DURATION.

TABLE NAME	STATUS	COMPLETED	ROWS	DURATION
public.categories	Completed	22/08/2019, 13:05:42	8	00:00:01
public.customer_customer_demo	Completed	22/08/2019, 13:05:42	0	00:00:00
public.customer_demographics	Completed	22/08/2019, 13:05:43	0	00:00:00
public.customers	Completed	22/08/2019, 13:05:45	91	00:00:02
public.employee_territories	Completed	22/08/2019, 13:05:45	49	00:00:02
public.employees	Completed	22/08/2019, 13:05:46	9	00:00:02
public.order_details	Completed	22/08/2019, 13:05:48	2155	00:00:02
public.orders	Completed	22/08/2019, 13:05:48	830	00:00:02
public.products	Completed	22/08/2019, 13:05:49	77	00:00:02
public.region	Completed	22/08/2019, 13:05:49	4	00:00:02

Select **Start cutover**. You'll see a page asking you to confirm that the operation is complete. Any remaining entries from the write-ahead log for the source database will be drained, and updates will stop. Further changes to the source database will not be propagated.



Demonstration: Performing an online migration

In this demonstration, you'll see how to migrate a sample database running on a MySQL server to Azure Database for MySQL. The sample database used is a version of the Northwind Traders database, adapted for MySQL. The database contains no foreign keys.

The main tasks in this demonstration are:

1. Configure the source server and export the schema.
2. Create a target database and import the schema.
3. Create an Azure Database Migration Service instance.
4. Create and run a migration project using the Database Migration Service.
5. Modify data, and cut over to the new database.

Setup

This demonstration uses a preconfigured virtual machine with MySQL and the sample database already installed. The sign-in account for the virtual machine and the MySQL database is *azureuser*, with the password *Pa55w.rdDemo*. To set up this virtual machine, perform the following steps:

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's running a MySQL server that's hosting the AdventureWorks database 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're running the **Bash** shell.

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

```
git clone <repo> workshop
```

5. Move to the *workshop/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 takes 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**. Make a note of the IP address.

7. Move to the *workshop/migration_samples/setup/mysql/northwind* folder.

```
cd ~/workshop/migration_samples/setup/mysql/northwind
```

8. Run the following command to create the "northwind" database on the Azure virtual machine. Replace *[nn.nn.nn.nn]* with the IP address of the virtual machine.

```
bash create_db.sh [nn.nn.nn.nn]
```

This command takes a couple of minutes to run. The script will display a warning message, and an error message concerning the use of the *log_bin_trust_function_creators* variable. Ignore both of these messages.

Task 1: Configure the source server and export the schema

1. 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.

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

2. On the virtual machine, switch to the root account. Enter the password for the *azureuser* user if prompted.

```
sudo bash
```

3. Restart the MySQL service:

```
service mysql restart
```

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 Fri 2019-08-30 14:05:06 UTC; 23s ago
Process: 26676 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid
(code=exited, status=0/SUCCESS)
Process: 26657 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/
SUCCESS)
Main PID: 26678 (mysqld)
Tasks: 27 (limit: 4070)
CGroup: /system.slice/mysql.service
┌└26678 /usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid
```

5. Leave the root account and return to the azureuser account:

```
exit
```

6. You're going to migrate a database named **northwind**. This database contains the details of products, customers, and orders for the Northwind Traders organization. Run the following command to connect to the database:

```
mysql -pPa55w.rd northwind
```

7. Run the following command to list the tables in the database:

```
SELECT TABLE_SCHEMA, TABLE_NAME FROM information_schema.tables WHERE TABLE_TYPE='BASE
TABLE' AND TABLE_SCHEMA='northwind';
```

8. You should see the following tables listed:

```
+-----+-----+
| TABLE_SCHEMA | TABLE_NAME      |
+-----+-----+
| northwind     | Categories        |
| northwind     | CustomerCustomerDemo |
| northwind     | CustomerDemographics |
| northwind     | Customers          |
| northwind     | EmployeeTerritories |
| northwind     | Employees          |
| northwind     | Order Details     |
| northwind     | Orders             |
| northwind     | Products           |
| northwind     | Region             |
| northwind     | Shippers           |
| northwind     | Suppliers           |
| northwind     | Territories        |
+-----+-----+
```

9. Run the following commands to view the data in the **orders** and **customers** tables:

```
SELECT * FROM orders;
SELECT * FROM customers;
```

10. Exit `mysql` with the **quit** command.
11. At the bash prompt, run the following command to export the schema for the **northwind** database to a file named **northwind_schema.sql**.

```
mysqldump -u azureuser -pPa55w.rd northwind --no-data > northwind_schema.sql
```

Task 2: Create a target database and import the schema

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 **Create MySQL server** page, enter the following details, and then select **Review + create**:

Property	Value
Resource group	Select Create new , and create a new resource group with a name of your choice
Server name	northwind <i>nnn</i> , where <i>nnn</i> is a suffix of your choice to make the server name unique
Data source	None
Admin username	northwindadmin
Password	Pa55w.rdDemo
Confirm password	Pa55w.rdDemo
Location	Select your nearest location
Version	5.7
Compute + storage	Select Configure server , select the Basic pricing tier, and then select OK

6. On the **Review + create** page, select **Create**. Wait for the service to be created before continuing.
7. When the service has been created, go to the page for the service in the portal, and select **Connection security**.
8. On the **Connection security page**, set **Allow access to Azure services** to **ON**.
9. 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.
10. Select **Save**, and wait for the firewall rules to be updated.
11. Return to the Cloud Shell, and leave the virtual machine.

```
exit
```

12. 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 created for the service:

```
az mysql db create \
  --name azurenorthwind \
```



```
mysql -h northwind[nnn].mysql.database.azure.com -u northwindadmin@northwind[nnn] -pPa55w.
rdDemo azurenorthwind < dropkeys.sql 2>>errors.txt
```

Task 3: Create an Azure Database Migration Service instance

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 to 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, and then select **Create**.

Property	Value
Service name	MySQL_migration_service
Select a resource group	Specify the same resource group that you used for the Azure Database for MySQL service
Location	Select your nearest location
Virtual network	Create a new virtual network named migration-servicevnet
Pricing tier	Premium, with 4 vCores

9. Wait while the Database Migration Service is created.

Task 4: Create and run a migration project using the Database Migration Service

1. In the Azure portal, go to the page for your Database Migration Service.
2. Select **New Migration Project**.
3. On the **New migration project** page, enter the following details, and then select **Create and run activity**.

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

4. When the **Migration Wizard** starts, on the **Add Source Details** page, enter the following details, and then select **Save**.

Property	Value
Source server name	Specify the IP address of the virtual machine running MySQL
Server port	3306
User Name	azureuser
Password	Pa55w.rd

- On the **Target details** page, enter the following details, and then select **Save**.

Property	Value
Target server name	northwind[nnn].mysql.database.azure.com
Username	northwindadmin@northwind[nnn]
Password	Pa55w.rdDemo

- on the **Select databases** page, select the **northwind** source database, and the **azurenorthwind** target database. Click **Save**.
- On the **Migration settings** page, expand the **northwind** dropdown, expand the **Advanced online migration settings** dropdown, verify that **Maximum number of instances to load in parallel** is set to 5, and then click **Save**.
- On the **Migration summary** page, in the **Activity name** box, type **MySQL_Migration_Activity**, and then select **Run migration**.
- On the **MySQL_Migration_Activity** page, select **Refresh** at 15-second intervals. You'll see the status of the migration operation as it progresses. Wait until the **MIGRATION DETAILS** column changes to **Ready to cutover**.

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

- Return to the **MySQL_Migration_Activity** page in the Azure portal.
- Select the **northwind** database.
- On the **northwind** page, verify that the status for all tables is marked as **COMPLETED**.
- Select **Incremental data sync**. Verify that the status for every table is marked as **Syncing**.
- Switch back to the bash prompt for the MySQL virtual machine.
- Run the following command to connect to the **northwind** database running using MySQL on the virtual machine:

```
mysql -h northwind[nnn].mysql.database.azure.com -u northwindadmin@northwind[nnn] -pPa55w.rdDemo azurenorthwind < addkeys.sql 2>>errors.txt
```
- Run the following command to connect to the **northwind** database running using MySQL on the virtual machine:

```
mysql -h northwind[nnn].mysql.database.azure.com -u northwindadmin@northwind[nnn] -pPa55w.rdDemo azurenorthwind
```
- Execute the following SQL statements to display, and then remove orders 10248, 10249, and 10250 from the database.

```
SELECT * FROM orders WHERE OrderID IN (10248, 10249, 10250);
SELECT * FROM `order details` WHERE OrderID IN (10248, 10249, 10250);
DELETE FROM `order details` WHERE OrderID IN (10248, 10249, 10250);
DELETE FROM orders WHERE OrderID IN (10248, 10249, 10250);
```

9. Close the *mysql* utility with the **quit** command.
10. Return to the **northwind** page in the Azure portal, and then select **Refresh**. Verify that the **public.order_details** table indicates that 8 rows have been deleted, and 3 rows have been removed from the **public.orders** table.
11. Select **Start cutover**.
12. On the **Complete cutover** page, select **Confirm**, and then select **Apply**. Wait until the status changes to **Completed**.
13. Switch back to the bash prompt for the MySQL virtual machine.
14. Run the following command to connect to the **azurenorthwind** database running using your Azure Database for MySQL service:

```
mysql -h northwind[nnn].mysql.database.azure.com -u northwindadmin@northwind[nnn] -pPa55w.rdDemo azurenorthwind
```

15. Execute the following SQL statements to display the orders and order details in the database. Quit after the first page of each table:

```
SELECT * FROM orders;
SELECT * FROM `order details`;
```

16. Run the following SQL statements to display the orders and details for orders 10248, 10249, and 10250:

```
SELECT * FROM orders WHERE OrderID IN (10248, 10249, 10250);
SELECT * FROM `order details` WHERE OrderID IN (10248, 10249, 10250);
```

Both queries should return 0 rows.

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

Perform offline migration

An offline migration takes a *snapshot* of the source database at a particular point in time, and copies that data to the target database. Any changes made to the source data after the snapshot has been taken will not be reflected in the target database.

You have at least two options if you want to perform an offline migration to Azure Database for MySQL:

Export and import

You can export a database from MySQL and import it into Azure Database for MySQL using tools such as **MySQL Workbench**. This is useful in a number of scenarios:

- You want to select which tables to import.

- You want to select which database objects to export and import including views, stored procedures, constraints, and functions.
- You want to also migrate data from other sources. For example, you need to add data from text files and use **mysqlimport** to add these as tables to your database.

You can perform an export and import using MySQL Workbench. From the **Server** menu, select **Data Export** or **Data Import**.

[NOTE]

You could also use **mysqldump** to export specific tables by listing their names after the database name.

For more information, see **Migrate your MySQL database by using import and export**¹⁶.

Dump and restore

Use dump and restore when you want to efficiently move the entire database.

There are a number of considerations when using dump and restore:

- Prevent triggers firing during restore by using the **exclude-triggers** option.
- Dump the entire database in a single transaction with the **single-transaction** option.
- Disable foreign key constraints firing during the process with the **disable-keys** option.
- Defer index creation until after the restore is complete with the **defer-table-indexes** option.

For more information on **mysqldump**, see **mysqldump — A Database Backup Program**¹⁷.

[NOTE]

You also use **mysqldump** to export specific tables by listing their names after the database name.

[NOTE]

You can't currently use the Azure Database Migration Service to perform an offline migration of a MySQL database.

Migrate by using dump and restore

Perform the following steps to migrate a database by using the dump and restore approach.

1. Export the data to another file with the **mysqldump** command:

```
mysqldump -h [host name] -u [username] -p[password] [database name] > db_data.sql
```

At this point, `db_data.sql` is a SQL script that you modify using a text editor.

2. Create the target database in Azure Database for MySQL. You do this with the Azure CLI:

```
az mysql db create \
  --name [database name] \
  --server-name [server name] \
  --resource-group [azure resource group]
```

3. Import the data into the target database with the **mysql** command:

¹⁶ <https://docs.microsoft.com/en-us/azure/mysql/concepts-migrate-import-export>

¹⁷ <https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html>


```
mysql -h [host name] -u [username] -p[password] [database name] < db_data.sql
```

Lesson 2: Summary

In this lesson, you saw how to:

- Create a MySQL server using Azure Database for MySQL.
- Perform an online migration of a MySQL database to Azure Database for MySQL using the Azure Database Migration Service.
- Perform an offline migration using MySQL tools.

Knowledge Check

Multiple choice

In MySQL, you have a database that includes large tables of data for employees and customers. In Azure Database for MySQL, you only want customer data.

What's the easiest way to migrate the required data to Azure Database for MySQL?

- Dump the database from MySQL, restore the database in Azure Database for MySQL, and then delete the unnecessary tables.
- Delete the unnecessary tables before performing a dump and restore.
- Perform a mysqldump, and then edit the sql file to omit the tables that aren't required.
- Perform a Data Export from MySQL Workbench, and select the required objects.

Multiple choice

You're performing an online migration on a live MySQL database.

What happens to transactions that occur during the migration?

- They're applied to the source database and need to be captured so they can be applied to the destination database after migration.
- They're applied to the source database and synchronized to the destination database.
- They're lost and the database should be taken offline before migration.
- They're blocked and client applications will receive an error. The database should be taken offline before migration.

Application Migration

Lesson 3: Application migration

In this lesson, you'll learn how to reconfigure your applications so they connect to your databases running in Azure Database for MySQL. You'll see how to test your applications, to verify that the system is functioning correctly.

Learning objectives

By the end of this lesson, you'll be able to:

- Create the necessary users and assign the appropriate privileges in your databases running under Azure Database for MySQL.
- Reconfigure applications to connect to databases running in your Azure Database for MySQL service.
- Test and verify that applications using your databases running under Azure Database for MySQL are still functioning correctly.

Create users and assign privileges

Your original on-premises server and database will contain users, the operations they perform, and the objects over which they do these operations. Azure Database for MySQL uses the same authentication and authorization mechanisms as MySQL running on-premises.

When you transfer a MySQL database to Azure Database for MySQL using the Azure Database Migration Service, the users aren't copied. You must manually recreate the necessary user accounts for the administrator and users of the tables in the target database. To do these tasks, you use the SQL language or a utility such as MySQL Workbench. Run the `CREATE USER` command. You use the `GRANT` command to assign the necessary privileges to a user. For example:

```
CREATE USER 'myuseraccount'@'%' IDENTIFIED BY 'mY!P@ss0rd';
GRANT ALL PRIVILEGES ON DATABASE [Database Name].* TO myuseraccount;
FLUSH PRIVILEGES;
```

To view the existing grants in the on-premises database, run the following SQL statement:

```
USE [Database Name];
```

```
SHOW GRANTS FOR 'myuseraccount'@'%';
```

Reconfigure applications

Reconfiguring an application to connect to Azure Database for MySQL is a straightforward process. However, it's crucial that you develop a strategy for migrating applications.

Considerations when reconfiguring MySQL applications

In a corporate environment, you might have many applications running against the same MySQL databases. There could be a large number of users running these applications. You want to be assured that, when you switch from the existing system to Azure Database for MySQL, your systems will still work,

users can continue doing their jobs, and business-critical operations remain operational. Module 1, Lesson 2, *Considerations for migration*, discussed many of the issues in general terms.

When migrating a MySQL database to Azure, there are some specifics to consider:

- If you're performing an offline migration, the data in the original MySQL database and the new databases running on Azure might start to diverge quickly if the old database is still being used. An offline migration is suitable when you take a system entirely out of operation for a short while, and then switch all applications to the new system before starting up again. This approach might not be possible for a business-critical system. If you're migrating to MySQL running on an Azure virtual machine, you can configure MySQL replication between your on-premises system and that running in Azure. Native MySQL replication operates in one direction only, but third-party solutions are available that support bidirectional replication between MySQL servers. These solutions won't work with Azure Database for MySQL.
- If you're performing an online migration, the Azure Database for MySQL service sets up replication from the on-premises database to the database running in Azure. After the initial data transfer, replication ensures that any changes made in the on-premises database are copied to the database in Azure, but not the other way round.

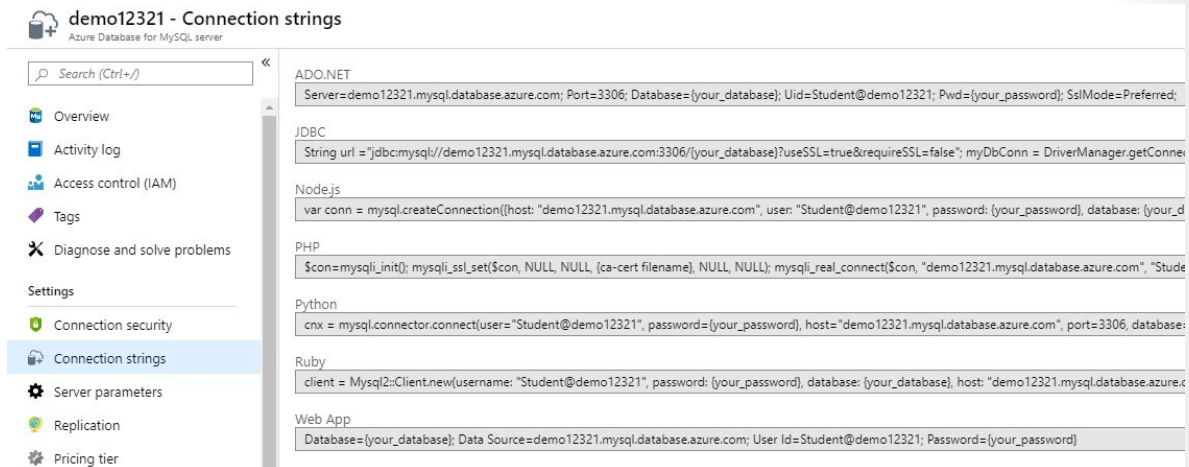
In both cases, you should ensure that you don't lose live data through an accidental overwrite. For example, in the online scenario, an application connected to the database running in Azure Database for MySQL could have its changes blindly overwritten by an application still using the on-premises database. Therefore, you should consider the following approaches:

- Migrate applications based on their workload type. An application that accesses the data for reading only can move safely to the database running in Azure Database for MySQL, and will see all changes made by applications still using the on-premises database. You can also adopt the converse strategy if read-only applications don't require fully up-to-date data.
- Migrate users based on their workload type. This strategy is similar to the previous one, except that you might have users who only generate reports while others modify the data. You can have the same application configured to connect to the appropriate database according to the user's requirements.
- Migrate applications based on the datasets they use. If different applications use different subsets of the data, you might be able to migrate these applications independently of each other.

Reconfiguring an application

To reconfigure an application, you point it at the new database. Most well-written applications should isolate the connection logic—this should be the only part of the code that requires changing. In many cases, connection information might be stored as configuration information, so you just need to update that information.

You'll find the connection information for your Azure Database for MySQL service in the Azure portal, on the **Connection strings** page for your Azure Database for MySQL service. Azure provides the information for many common programming languages and frameworks.



Open network ports

As mentioned in Lesson 1 of this module, Azure Database for MySQL is a protected service that runs behind a firewall. Clients can't connect unless their IP address is recognized by the service. You must add the IP addresses, or address block ranges, for clients running applications that need to connect to your databases.

Test and verify applications

Before you switch applications and users to the new database, it's important to ensure that you've configured everything correctly.

Start by "dry-running" applications and connect as each role to ensure the correct functionality is available.

Next, perform "soak tests" to mimic the number of users running typical workloads concurrently for a period of time. Monitor the system, and verify that you've allocated sufficient resources to your Azure Database for MySQL service.

You can now start to roll out the system to users. It might be beneficial to implement some form of "canary testing", where a small subset of users is transferred to the system unawares. This gives you an unbiased opinion as to whether users are having the same, better, or worse experience with the new database.

Lesson 3: Summary

In this lesson, you saw how to:

- Create the necessary users and assign appropriate privileges in your databases running under Azure Database for MySQL.
- Reconfigure applications to connect to databases running in your Azure Database for MySQL service.
- Test and verify that applications using your databases running under Azure Database for MySQL are still functioning correctly.

Knowledge Check

Multiple choice

You have many user accounts in MySQL.

How are these accounts migrated to Azure Database for MySQL?

- The accounts are automatically migrated by the Azure Database Migration Service.
- The accounts are automatically migrated by a dump and restore migration.
- You must manually create the accounts post-migration.
- You can't use MySQL accounts—you must use Azure Active Directory accounts.
- You must create the Azure Account Migration resource.

Multiple choice

You have many applications, using a variety of programming languages, that must be redirected to your migrated database.

Where do you find the connection settings for your client application?

- In the INFORMATION_SCHEMA.CONNECTION_STRINGS table in Azure Database for MySQL.
- In the INFORMATION_SCHEMA.CONNECTION_SECURITY table in Azure Database for MySQL.
- In the Connection strings page for Azure Database for MySQL.
- In the Connection settings page for your Azure portal.

Module summary

Module summary

In this module, you learned about the benefits of migrating MySQL workloads to Azure, and the features that Azure provides to help manage and optimize MySQL systems. You saw how to create an instance of the Azure Database for MySQL service, and then you migrated an on-premises MySQL database to Azure. You also reconfigured an application that uses the database to connect to Azure.

Takeaways

In this module you:

- Learned about the features and limitations of Azure Database for MySQL.
- Migrated an on-premises MySQL database to Azure Database for MySQL.
- Reconfigured an application that used your on-premises MySQL database to connect to Azure Database for MySQL instead.

Module Lab Information

Answers

Multiple choice

With the increasing risk of cyber attacks, you're concerned that your database servers have the latest bug fixes.

How can you ensure that your Azure Database for MySQL servers have the latest bug fix updates applied?

- Review the Azure Performance Recommendations and apply bug fixes when suggested.
- Set an alert for bug fix updates and create a runbook to apply the updates automatically.
- Bug fix updates are automatically applied. No action needs to be taken.
- Perform regular upgrades when a new version of Azure Database for MySQL becomes available.
- Use the Advanced Threat Protection feature to ensure bug fixes are applied.

Explanation

Bug fix updates are automatically applied. No action needs to be taken.

Multiple choice

You want to keep your writeable on-premises version of a MySQL database, but create replica read-only copies for users in another country.

What's the most straightforward method to achieve this with Azure Database for MySQL?

- Create an Azure Database for MySQL instance in the region closest to your users and configure Data-In Replication.
- Create a read replica copy of your database located in the region closest to your users.
- Create a writeable Azure Database for MySQL instance in the region closest to your users, and create a read-only replica in your on-premises MySQL server.
- Create a writeable Azure Database for MySQL instance in the region closest to your users, and create a writeable replica in your on-premises MySQL server.

Explanation

Data-in Replication uses the native replication functionality of MySQL to replicate data from an external MySQL Server into Azure Database for MySQL.

Multiple choice

In MySQL, you have a database that includes large tables of data for employees and customers. In Azure Database for MySQL, you only want customer data.

What's the easiest way to migrate the required data to Azure Database for MySQL?

- Dump the database from MySQL, restore the database in Azure Database for MySQL, and then delete the unnecessary tables.
- Delete the unnecessary tables before performing a dump and restore.
- Perform a mysqldump, and then edit the sql file to omit the tables that aren't required.
- Perform a Data Export from MySQL Workbench, and select the required objects.

Explanation

All of the options would work, but MySQL Workbench allows you to select which objects to export in a straightforward manner.

Multiple choice

You're performing an online migration on a live MySQL database. What happens to transactions that occur during the migration?

- They're applied to the source database and need to be captured so they can be applied to the destination database after migration.
- They're applied to the source database and synchronized to the destination database.
- They're lost and the database should be taken offline before migration.
- They're blocked and client applications will receive an error. The database should be taken offline before migration.

Explanation

The Azure Database Migration Service copies all of your existing data to Azure, and then continuously performs a synchronization operation from the source database. Any new transactions performed against the on-premises system will be copied to the new database in Azure.

Multiple choice

You have many user accounts in MySQL. How are these accounts migrated to Azure Database for MySQL?

- The accounts are automatically migrated by the Azure Database Migration Service.
- The accounts are automatically migrated by a dump and restore migration.
- You must manually create the accounts post-migration.
- You can't use MySQL accounts—you must use Azure Active Directory accounts.
- You must create the Azure Account Migration resource.

Explanation

Azure Database for MySQL uses the same authentication and authorization mechanisms as MySQL running on-premises. When you transfer a MySQL database to Azure Database for MySQL using the Azure Database Migration Service, users aren't copied. You must manually recreate the necessary user accounts.

Multiple choice

You have many applications, using a variety of programming languages, that must be redirected to your migrated database.

Where do you find the connection settings for your client application?

- In the INFORMATION_SCHEMA.CONNECTION_STRINGS table in Azure Database for MySQL.
- In the INFORMATION_SCHEMA.CONNECTION_SECURITY table in Azure Database for MySQL.
- In the Connection strings page for Azure Database for MySQL.
- In the Connection settings page for your Azure portal.

Explanation

You'll find the connection information for your Azure Database for MySQL service in the Azure portal, on the Connection strings page for your Azure Database for MySQL service.



Module 3 Migrate on-premises PostgreSQL to Azure Database for PostgreSQL

Module introduction

Module 3: Migrate on-premises PostgreSQL databases to Azure Database for PostgreSQL

Module introduction

In this module, you'll learn about the benefits of migrating PostgreSQL workloads to Azure, and the features that Azure provides to help manage and optimize PostgreSQL systems. You'll see how to create an instance of the Azure Database for PostgreSQL service, and then you'll learn how to migrate on-premises PostgreSQL databases to Azure. You'll also see how to reconfigure an application that uses the database to connect to Azure instead.

Learning objectives

By the end of this module, you'll be able to:

- Describe the features and limitations of Azure Database for PostgreSQL.
- Migrate an on-premises PostgreSQL database to Azure Database for PostgreSQL.
- Reconfigure existing applications that use your on-premises PostgreSQL databases to connect to Azure Database for PostgreSQL.

Introduction to Azure Database for PostgreSQL

Lesson 1: Introduction to Azure Database for PostgreSQL

In this lesson, you'll learn about Azure Database for PostgreSQL in detail. You'll be introduced to the two versions of this service, and learn about when it's appropriate to use them. You'll find out about the features that Azure provides to help to protect, manage, and monitor databases running in Azure Database for PostgreSQL.

Learning objectives

By the end of this lesson, you'll be able to:

- Describe the features of Azure Database for PostgreSQL, single server version.
- Explain the additional features provided by Azure Database for PostgreSQL, HyperScale (Citius).
- Describe how to create read-only replicas for Azure Database for PostgreSQL.
- List the tools that Azure Database for PostgreSQL provides to manage and monitor databases.
- Explain the connectivity options for client applications that use databases running in Azure Database for PostgreSQL.
- Summarize the PostgreSQL extensions that are supported by Azure Database for PostgreSQL.

Azure Database for PostgreSQL, Single Server

The Azure Database for PostgreSQL service is an implementation of the community version of PostgreSQL. The single server version of this service is suitable for a broad range of traditional, transactional workloads. The service provides the common features used by typical PostgreSQL systems, including geo-spatial support and full-text search.

Microsoft have adapted PostgreSQL for the Azure platform, and it's closely integrated with many Azure services. The Azure Database for PostgreSQL service is fully managed by Microsoft. Microsoft handle updates and patches to the software, and provide an SLA of 99.99% availability. This means you can just focus on the databases and applications running, using the service.

You can deploy multiple databases in each instance of this service.

Pricing tiers

When you create an instance of the Azure Database for PostgreSQL service, you specify the compute and storage resources that you want to allocate by selecting a *Pricing tier*. A pricing tier combines the number of virtual processor cores, the amount of storage available, and various backup options. The more resources you allocate, the higher the cost.

The Azure Database for PostgreSQL service uses storage to hold your database files, temporary files, transaction logs, and the server logs. You can optionally specify that you want the storage available to be increased when you get close to the current capacity. If you don't select this option, servers that run out of storage will continue running, but operate as read-only.

The Azure portal groups pricing tiers into three broad ranges:

- **Basic**, which is suitable for small systems and development environments, but has variable I/O performance.
- **General Purpose**, which provides predictable performance, up to 6000 IOPS, depending on the number of processor cores and the storage space available.
- **Memory Optimized**, which uses up to 32 memory-optimized virtual processor cores, and also provides predictable performance of up to 6000 IOPS.

Microsoft also have a *Large storage* option in preview, which can provision up to 16 TB of storage and support up to 20,000 IOPS.

You can fine-tune the number of processor cores and storage that you require. You can scale up and down the processing resources—you can't scale storage down, only up—and switch between the General Purpose and Memory Optimized pricing tiers as necessary after you've created your databases. You only pay for what you need.

Configure

Basic
Up to 2 vCores with
Variable I/O performance (1-2 vCores)

General purpose
Up to 64 vCores with
predictable I/O performance (2-64 vCores)

Memory optimized
up to 32 memory optimized vCores
with predictable I/O performance (2-32 vCores)

Please note that changing to and from the Basic pricing tier or changing the backup redundancy options after server creation is not supported.

Compute generation - [Learn more about compute generation](#)

Gen 5 ✓

vCore - [What is a vCore?](#)

4 vCores

Would you like to configure larger storage (Public Preview)? - [Learn more](#)

Storage (type: General purpose storage)

672 GB

Can use up to **2016** available IOPS

Auto-growth - [Learn More](#)

Yes No

Backup retention period

15 Days

Backup redundancy options - [Learn more details](#)

Locally redundant ✓
Recover from data loss within region

Geo-redundant
Recover from regional outage or disaster

Price summary

Gen 5 Compute generation	
Cost per vCore	85.41
vCores selected	× 4
+	
General purpose storage	
Cost per GB / month	0.13
Storage amount selected	× 672
Est. monthly cost	428.00 GBP

Additional charge per usage
See [pricing details](#) for more detail.

OK

[NOTE]

If you change the number of processor cores, Azure creates a new server with this compute allocation. When the server is running, client connections are switched to the new server. This switch can take up to a minute. During this interval, no new connections can be made, and any in-flight transactions will be rolled back.

If you only change the storage size of backup options, there's no interruption in service.

The pricing tier and the processing resources allocated determine the maximum number of concurrent connections the service will support. For example, if you select the General Purpose pricing tier and allocate 64 virtual cores, the service supports 1900 concurrent connections. The Basic tier, with two virtual cores, handles up to 100 concurrent connections. Azure itself requires five of these connections to monitor the server. If you exceed the number of available connections, clients will receive the error **FATAL: sorry, too many clients already**. See [Limits in Azure Database for PostgreSQL - Single Server](#)¹ for more information.

Prices can change. Visit the [Azure Database for PostgreSQL pricing](#)² page for the latest information.

Server parameters

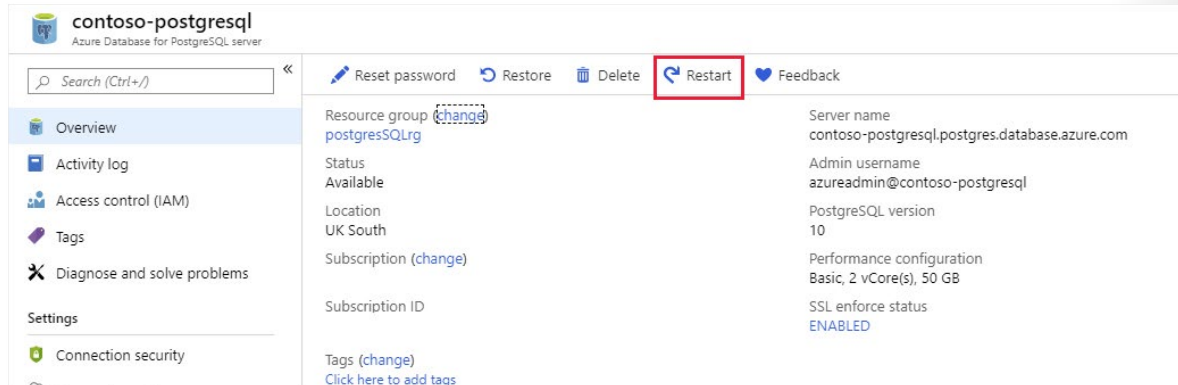
In an on-premises installation of PostgreSQL, you set server configuration parameters in the `postgresql.conf` file. Use Azure Database for PostgreSQL to modify configuration parameters through the **Server parameters** page. Not all parameters for an on-premises installation of PostgreSQL are relevant to Azure Database for PostgreSQL, so the Server parameters page only lists those parameters that are appropriate to Azure.

PARAMETER NAME	VALUE	PARAMETER TYPE	DESCRIPTION
array_nulls	<input type="checkbox"/> ON <input type="checkbox"/> OFF	Dynamic	Enable input of NULL elements in arrays.
backslash_quote	SAFE_ENCODING	Dynamic	Sets whether "\" is allowed in string literals.
bytea_output	HEX	Dynamic	Sets the output format for bytea.
check_function_bodies	<input type="checkbox"/> ON <input type="checkbox"/> OFF	Dynamic	Check function bodies during CREATE FUNCTION.
client_encoding	SQL_ASCII	Dynamic	Sets the client's character set encoding.
client_min_messages	NOTICE	Dynamic	Sets the message levels that are sent to the client.
connection_throttling	<input type="checkbox"/> ON <input type="checkbox"/> OFF	Dynamic	Enables temporary connection throttling per IP for too m...
constraint_exclusion	PARTITION	Dynamic	Enables the planner to use constraints to optimize queries.
cpu_index_tuple_cost	0.005	Dynamic	Sets the planner's estimate of the cost of processing each...
cpu_operator_cost	0.0025	Dynamic	Sets the planner's estimate of the cost of processing each...
cpu_tuple_cost	0.01	Dynamic	Sets the planner's estimate of the cost of processing each...
cursor_tuple_fraction	0.1	Dynamic	Sets the planner's estimate of the fraction of a cursor's ro...
datestyle	iso, mdy	Dynamic	Sets the display format for date and time values.
deadlock_timeout	1000	Dynamic	Sets the amount of time, in milliseconds, to wait on a lock...
debug_print_parse	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF	Dynamic	Logs each query's parse tree.
debug_print_plan	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF	Dynamic	Logs each query's execution plan.
debug_print_rewritten	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF	Dynamic	Logs each query's rewritten parse tree.
default_statistics_target	100	Dynamic	Sets the default statistics target.
default_text_search_config	pg_catalog.english	Dynamic	Sets default text search configuration.
default_transaction_deferrable	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF	Dynamic	Sets the default deferrable status of new transactions.
default_transaction_isolation	READ COMMITTED	Dynamic	Sets the transaction isolation level of each new transaction.
default_transaction_read_only	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF	Dynamic	Sets the default read-only status of new transactions.
default_with_oids	<input type="checkbox"/> ON <input checked="" type="checkbox"/> OFF	Dynamic	Create new tables with OIDs by default.

Changes to parameters marked as *Dynamic* take effect immediately. Static parameters require a server restart. You restart the server using the **Restart** button on the **Overview** page in the portal:

¹ <https://docs.microsoft.com/azure/postgresql/concepts-limits#maximum-connections>

² <https://azure.microsoft.com/pricing/details/postgresql/server/>



High availability

Azure Database for PostgreSQL is a highly available service. It contains built-in failure detection and failover mechanisms. If a processing node stalls due to a hardware or software issue, a new node will be switched in to replace it. Any connections currently using that node will be dropped but automatically opened against the new node. Any transactions being performed by the failing node will be rolled back. For this reason, you should always ensure that clients are configured to detect and retry failing operations.

Supported PostgreSQL versions

The Azure Database for PostgreSQL service currently supports PostgreSQL version 11, back to version 9.5. You specify which version of PostgreSQL to use when you create an instance of the service. Microsoft aim to update the service as new versions of PostgreSQL become available, and will maintain compatibility with the the previous two major versions.

Azure automatically manages upgrades to your databases between minor versions of PostgreSQL—but not major versions. For example, if you have a database that uses PostgreSQL version 10, Azure can automatically upgrade the database to version 10.1. If you want to switch to version 11, you must export your data from the databases in the current service instance, create a new instance of the Azure Database for PostgreSQL service, and import your data into this new instance.

Azure Database for PostgreSQL, HyperScale (Citus)

The Hyperscale version of the Azure Database for PostgreSQL service is a multiserver implementation of PostgreSQL. Clients connect to a *Coordinator* node that distributes work across a number of *Worker* nodes. Each worker node is a PostgreSQL server in its own right, and all worker nodes operate independently of each other. You use this architecture to allocate many more processing and storage resources than are available in the single server model, providing increased scalability and power. This service is suitable for workloads in excess of 100 GB in size, real-time analytics, and high-throughput transactional processing.

When you create an instance of the Azure Database for PostgreSQL HyperScale (Citus) service, you specify the number of worker nodes, and the size of the nodes. You also specify the resources for the coordinator node.

Configure server group

⚠ Adding nodes as well as scaling compute and storage on coordinator and worker nodes after server group creation is not supported yet. Please open a support ticket if you need to scale your server group.

Worker nodes
Expand your server group and scale your database by adding worker nodes. Select up to 32 vCores with 8 GiB RAM per vCore and up to 2 TiB of storage with up to 3 IOPS / GiB per node.

Worker node count: 8 nodes
If you need more than 20 nodes, [contact us](#)

Configuration (per worker node)

vCores [Learn more](#) 8 vCores

Storage [Learn more](#) 1 TiB
Your configuration can use up to 3 IOPS / GiB.

Coordinator node
The coordinator node coordinates your queries and manages your schema.

Coordinator node **4 vCores, 4 GiB RAM per vCore**
0.5 TiB storage, up to 3 IOPS / GiB.
[Change configuration](#)

Cost summary

Worker nodes	
Cost per vCore / month (in GBP)	43.78
vCores selected	x 8
Node count	x 8
Worker node storage - General purpose	
Cost per GiB / month (in GBP)	0.05
Storage selected (in GiB)	x 1,024
Node count	x 8
Coordinator node	
Cost per vCore / month (in GBP)	38.65
vCores selected	x 4
Coordinator node storage - General purpose	
Cost per GiB / month (in GBP)	0.05
Storage selected (in GiB)	x 512
ESTIMATED COST / MONTH	3,403.89 GBP
ADDITIONAL CHARGE PER USAGE	
See pricing details for more detail.	

[Save](#)

Coordinator and worker nodes

Data is sharded and distributed between worker nodes. The query engine in the coordinator can parallelize complex queries, directing the processing towards the appropriate worker nodes. Worker nodes are selected according to which shards hold the data being processed. The coordinator then accumulates the results from the worker nodes before sending them back to the client. More straightforward queries might be performed using only a single worker node. Clients also connect to the coordinator, and never communicate directly with a worker node.

You can scale the number of worker nodes up and down in your service, as required.

Distributing data

You distribute data across worker nodes by creating *distributed* tables. A distributed table is split into shards, and each shard is allocated to storage on a worker node. You indicate how to split the data by defining a column as the *distribution* column. The data is sharded based on the values of the data in this column. When you design a distributed table, it's important to select the distribution column carefully; you should use a column with a large number of distinct values that would typically be used to group related rows. For example, in a table for an e-commerce system that stores information about customers' orders, the customer ID might be a reasonable distribution column. All orders for a given customer will be held in the same shard, but orders for all customers will be spread across shards.

You can also create *reference* tables. These tables contain lookup data, such as the names of cities or status codes. A reference table is replicated in its entirety to every worker node. The data in a reference table should be relatively static; each change requires updating every copy of the table.

Finally, you can create *local* tables. A local table isn't sharded, but is stored on the coordinator node. Use local tables for holding small tables with data that's unlikely to be required by joins. Examples include the names of users and their login details.

Replicate data in Azure Database for PostgreSQL, Single Server

The Hyperscale version of Azure Database for PostgreSQL is inherently distributed, supporting replication and sharding. The Single Server version of this service supports read-only replication. Read-only replicas are useful for handling read-intensive workloads. Client connections can be spread across replicas, easing the burden on a single instance of the service. If your clients are located in different regions of the world, you use cross-region replication to position data close to each set of clients, and reduce latency.

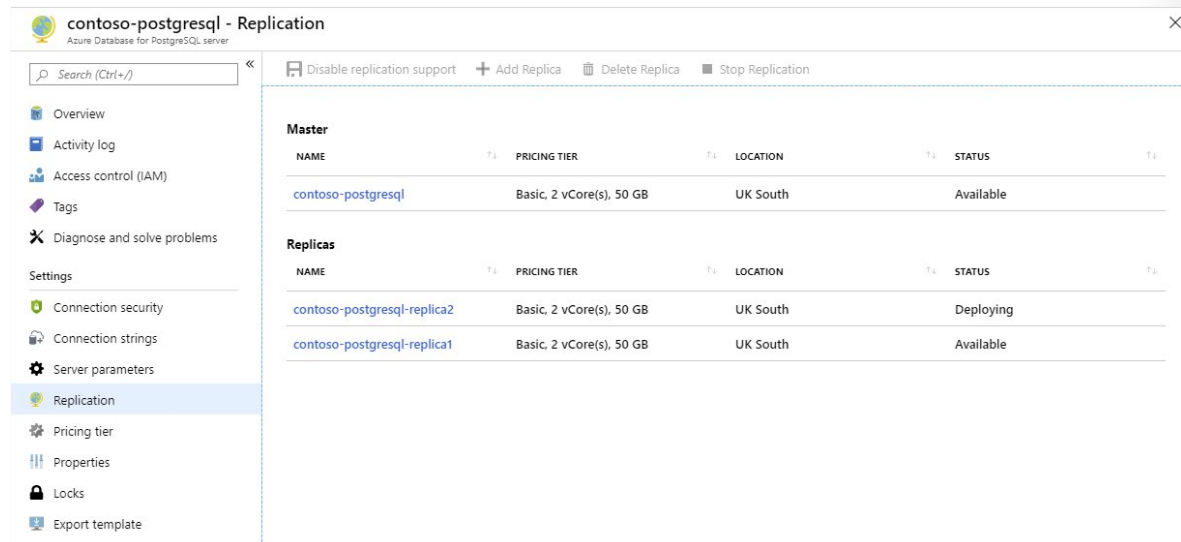
You can also use replicas as part of a contingency plan for disaster recovery. If the master server becomes unavailable, you might still be able to connect to a replica.

[NOTE]

If the master is lost or deleted, all read-only replicas become read-write servers instead. However, these servers will be independent of each other, so any changes made to the data in one server will not be copied to the remaining servers.

Establishing a replica

A read-only replica contains a copy of the databases held in the original server—referred to as the *master*. You use the Azure portal or the CLI to create a replica of a master.



The screenshot shows the Azure portal interface for the 'contoso-postgresql' server. The 'Replication' section is active, displaying a table for the Master server and a table for the Replicas.

Master					
NAME	PRICING TIER	LOCATION	STATUS		
contoso-postgresql	Basic, 2 vCore(s), 50 GB	UK South	Available		

Replicas					
NAME	PRICING TIER	LOCATION	STATUS		
contoso-postgresql-replica2	Basic, 2 vCore(s), 50 GB	UK South	Deploying		
contoso-postgresql-replica1	Basic, 2 vCore(s), 50 GB	UK South	Available		

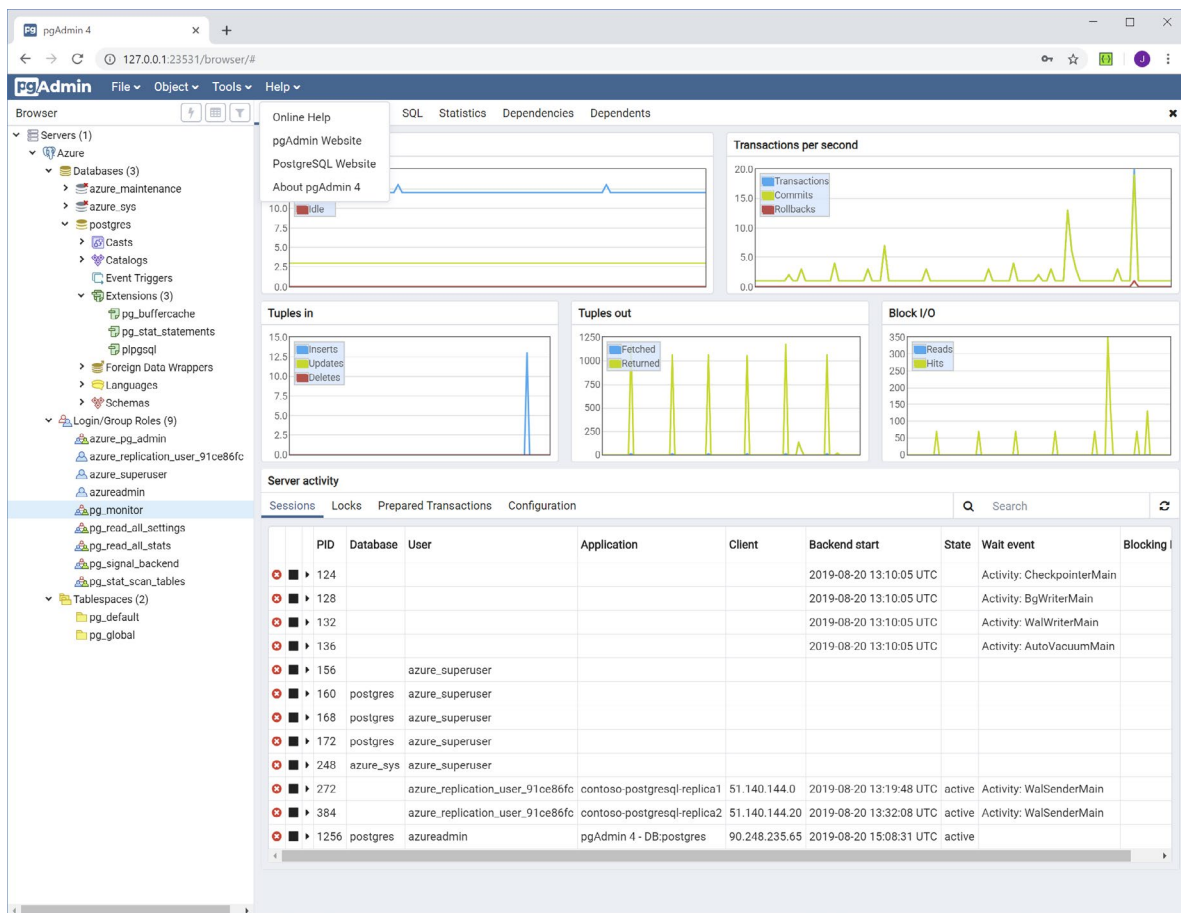
When you create a read-only replica, Azure creates a new instance of the Azure Database for PostgreSQL service, and then copies the databases from the master server to the new server. The replica runs in read-only mode. Any attempt to modify data will fail.

Replica lag

Replication is not synchronous, and any changes made to data in the master server might take some time to appear in the replicas. Client applications that connect to replicas must be able to cope with this level of eventual consistency. Azure Monitor enables you to track the time lag in replication by using the **Max Lag Across Replicas** and **Replica Lag** metrics.

Management and monitoring

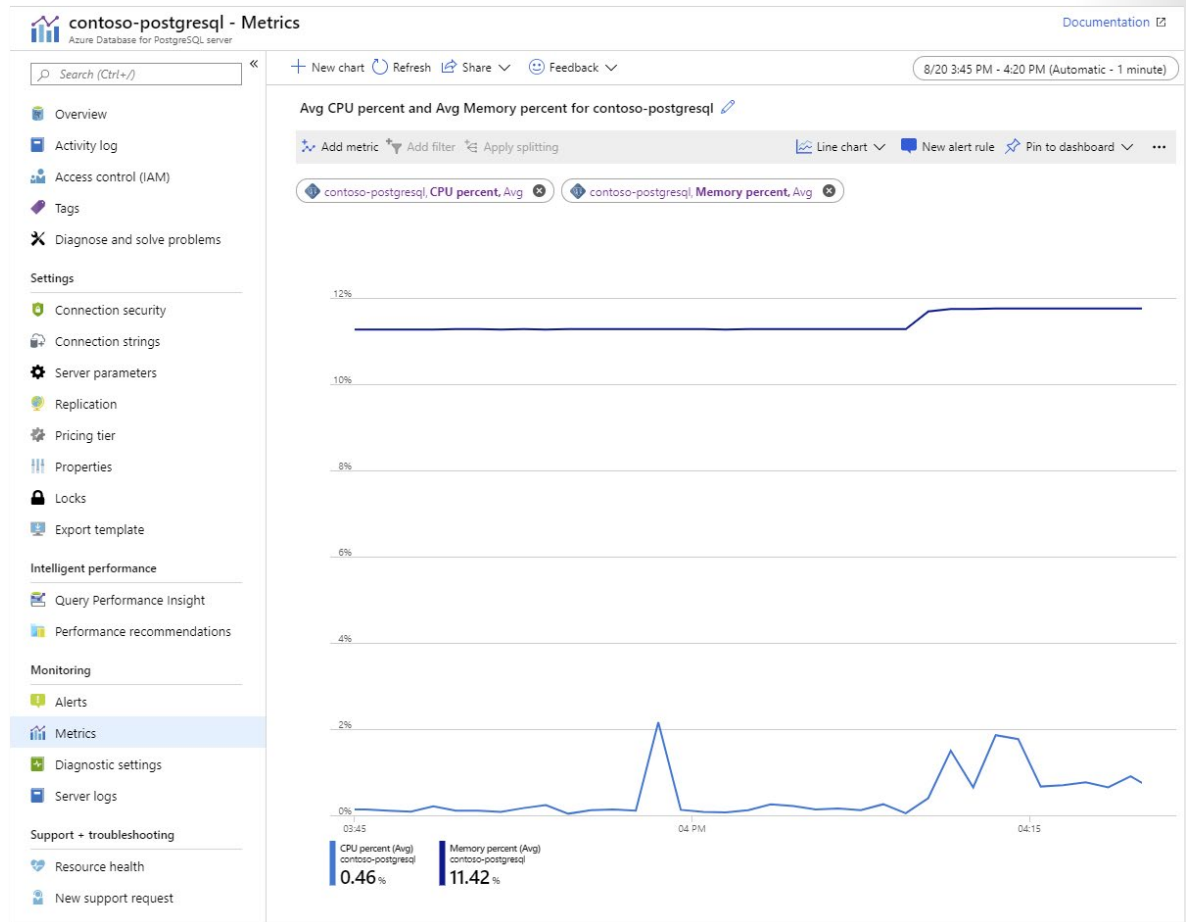
You can use familiar tools such as **pgAdmin** to connect to Azure Database for PostgreSQL to manage and monitor your databases. However, some server-focused functionality, such as performing server backup and restore, are not available because the server is managed and maintained by Microsoft.



Azure tools for monitoring Azure Database for PostgreSQL

Azure provides an extensive set of services that you use to monitor server and database performance, and troubleshoot issues. These services enable you to view how PostgreSQL is utilizing the Azure resources you've allocated. You use this information to assess whether you need to scale your system, modify the structure of tables and indexes in your databases, and visualize runtime statistics and other events. The services available include:

- **Azure Monitor.** The Azure Database for PostgreSQL provides metrics that enable you to track items such as CPU and storage utilization, I/O rates, memory occupancy, the number of active connections, and replication lag:



- **Server Logs.** Azure makes the logs available for each PostgreSQL server. You download them from the Azure portal:

The screenshot shows the 'Server logs' page for a PostgreSQL server. The left sidebar contains navigation options such as Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Intelligent performance, and Monitoring. The main content area displays a table of log files.

NAME	LAST UPDATE TIME	SIZE
postgresql-2019-08-20_150000.log	Tue, 20 Aug 2019 15:53:30 GMT	17KB
postgresql-2019-08-20_140000.log	Tue, 20 Aug 2019 14:56:54 GMT	8KB
postgresql-2019-08-20_131004.log	Tue, 20 Aug 2019 13:57:05 GMT	12KB

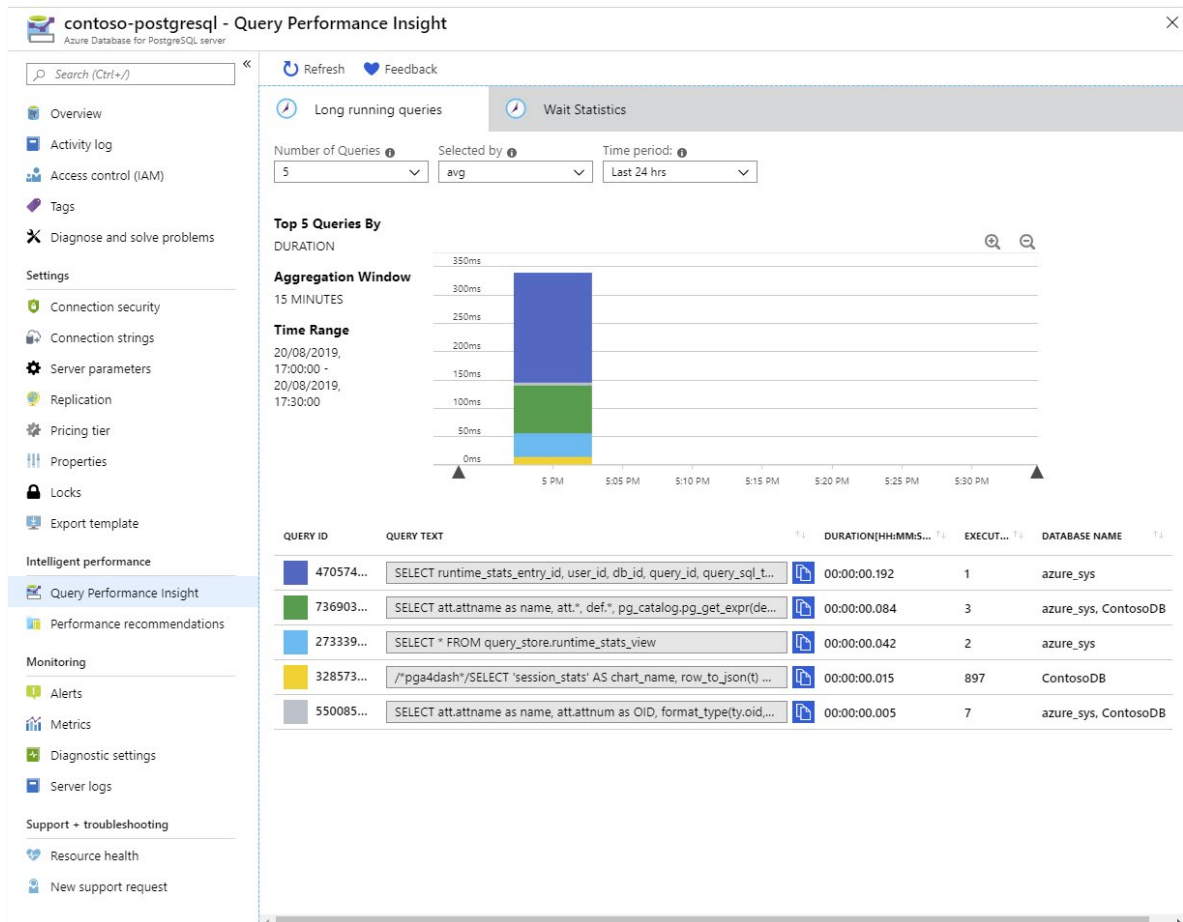
- Query Store and Query Performance Insights.** The Azure Database for PostgreSQL stores information about the queries run against databases on the server, and saves them in a database named *azure_sys*, in the *query_store* schema. You query the *query_store.qs_view* view to see this information. By default, Azure Database for PostgreSQL doesn't capture any query information as it imposes a small overhead, but you can enable tracking by setting the *pg_qs.query_capture_mode* server property to *ALL* or *TOP*.

The screenshot shows the 'Server parameters' configuration page for a PostgreSQL server. The left sidebar contains navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Connection security, Connection strings, Server parameters (highlighted), Replication, Pricing tier, Properties, Locks, Export template, Intelligent performance, Query Performance Insight, and Performance recommendations. The main area displays a list of server parameters with their current values and dynamic/static status. The 'pg_qs.query_capture_mode' parameter is highlighted with a red box and set to 'ALL'.

Parameter Name	Value	Dynamic/Static
log_min_messages	WARNING	Dynamic
log_retention_days	3	Dynamic
log_statement	NONE	Dynamic
log_statement_stats	ON/OFF	Dynamic
max_parallel_workers	8	Dynamic
max_parallel_workers_per_gather	2	Dynamic
max_prepared_transactions	0	Static
max_standby_archive_delay	30000	Dynamic
max_standby_streaming_delay	30000	Dynamic
min_parallel_index_scan_size	524288	Dynamic
min_parallel_table_scan_size	8388608	Dynamic
parallel_setup_cost	1000	Dynamic
parallel_tuple_cost	0.1	Dynamic
pg_qs.max_query_text_length	6000	Dynamic
pg_qs.query_capture_mode	ALL	Dynamic
pg_qs.replace_parameter_placeholders	ON/OFF	Dynamic

You also configure Query Store to capture information about queries that spend time waiting. A query might have to wait while another query releases a lock on a table, or because the query is performing a lot of I/O, or because memory is running short. You see this information in the `query_store.runtime_stats_view` view.

If you prefer to visualize these statistics rather than running SQL statements, use Query Performance Insight in the Azure portal:



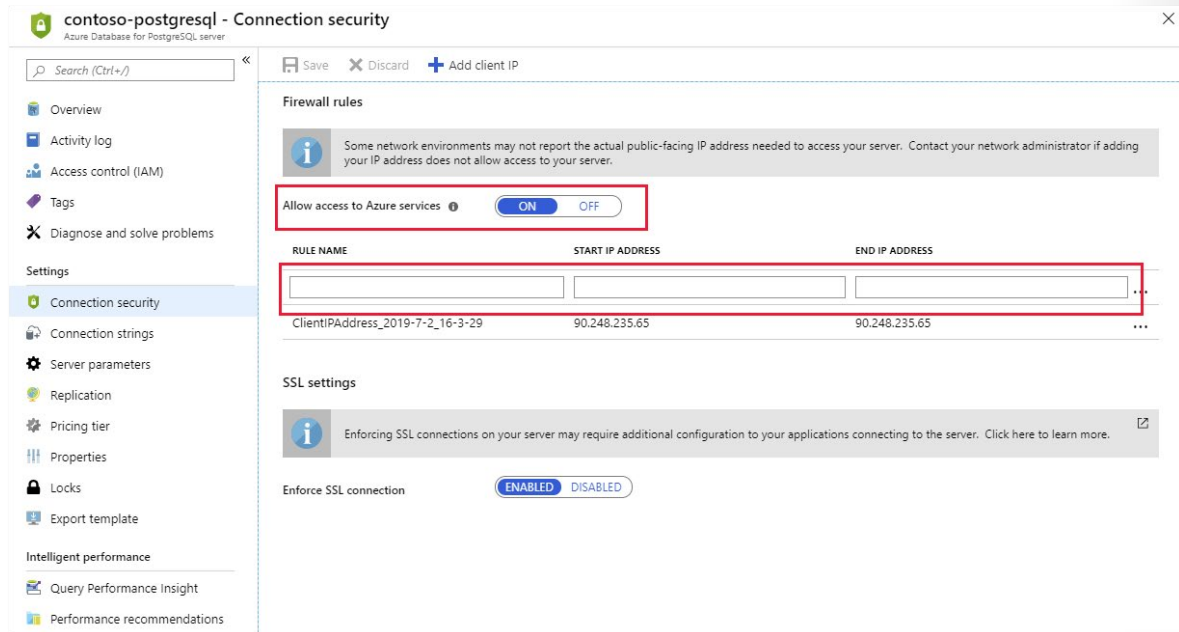
- Performance Recommendations.** The Performance Recommendations utility, also available in the Azure portal, examines the queries your applications are running. It also looks at the structures in the database, and recommends how to organize your data—and whether you should consider adding or removing indexes.

Client connectivity

Azure Database for PostgreSQL runs behind a firewall. To access your service and database you must add a firewall rule for the IP address ranges from which your clients connect. If you need to access the service from within Azure—such as an application running using Azure App Services—you must also enable access to Azure services.

Configure the firewall

The simplest way to configure the firewall is to use the Connection Security settings for your service in the Azure portal. Add a rule for each client IP address range. You also use this page to enforce SSL connections to your service.



You click **Add Client IP** in the toolbar to add the IP address of your desktop computer.

If you've configured read-only replicas, you must add a firewall rule to each one to make them accessible to clients.

Client connection libraries

If you're writing your own client applications, you must use the appropriate database driver to connect to a PostgreSQL database. Many of these libraries are programming-language dependent. They are maintained by independent third parties. Azure Database for PostgreSQL supports client libraries for Python, PHP, Node.js, Java, Ruby, Go, C# (.NET), ODBC, C, and C++. You'll find a list of the currently supported libraries online at **Connection libraries for Azure Database for PostgreSQL - Single Server**³.

Client retry logic

As mentioned earlier, some events—such as failover during high availability recovery, and scaling up the CPU resources—can cause a brief loss in connectivity. Any transactions in progress will be rolled back. Azure Database for PostgreSQL automatically redirects a connected client to a working node, but any operations being performed by the client at that time will return an error. You should treat this occurrence as a transient exception. Your application code should be prepared to catch these exceptions and retry them.

Lesson 1: Summary

In this lesson, you learned how to:

- Describe the features of Azure Database for PostgreSQL, Single Server version.
- Explain the additional features provided by Azure Database for PostgreSQL, HyperScale (Citrus).
- Describe how to create read-only replicas for Azure Database for PostgreSQL.

³ <https://docs.microsoft.com/azure/postgresql/concepts-connection-libraries>

- List the tools that Azure Database for PostgreSQL provides to manage and monitor databases.
- Explain the connectivity options for client applications that use databases running in Azure Database for PostgreSQL.
- Summarize the PostgreSQL extensions that are supported by Azure Database for PostgreSQL.

Knowledge Check

Multiple choice

You have customers running applications that access your PostgreSQL database. The operations these applications perform are typically queries that generate reports. These customers are located in Australia, Europe, and the United States. Your corporate headquarters is in the United States—this is where you insert, update, and delete most of the data.

How can you minimize query latency for your customers?

- Run copies of the database using Azure Database for PostgreSQL in Australia, Europe, and America. Configure bidirectional replication between all three sites so that they're kept synchronized with each other.
- Run copies of the database using Azure Database for PostgreSQL in Australia, Europe, and America. At regular intervals, back up the data at each site, and restore it on the other two sites.
- Implement read replicas with Azure Database for PostgreSQL. Make the United States office the master server, and replicate the data to replicas running in Australia and Europe.
- Host your database in an instance of the Azure Database for PostgreSQL service, and scale up to the maximum number of virtual processor cores.
- Use a dedicated high-bandwidth network connection between each customer and the database.

Multiple choice

You want to use the pgAdmin utility to monitor your database running in Azure Database for PostgreSQL. You're attempting to connect from your desktop computer. You're using the correct server name, protocol (SSL), username, and password, but you receive an error.

What might the problem be?

- You can't use pgAdmin to monitor a database running in Azure Database for PostgreSQL.
- The computer running the pgAdmin utility must be in the same virtual network as the Azure Database for PostgreSQL service instance.
- Check that you've added a firewall rule to your Azure Database for PostgreSQL service that allows connections from your desktop computer.
- Azure Database for PostgreSQL doesn't support SSL connections.
- Azure Database for PostgreSQL doesn't support username and password authentication.

Migrate on-premises PostgreSQL database to Azure

Lesson 2: Migrate an on-premises PostgreSQL database to Azure

In this lesson, you'll learn how to migrate a PostgreSQL database from on-premises to Azure Database for PostgreSQL. You'll see how to perform an online migration using the Azure Database Migration Service, and how to do an offline migration by manually transferring data from on-premises databases to Azure Database for PostgreSQL.

Learning objectives

By the end of this lesson, you'll be able to:

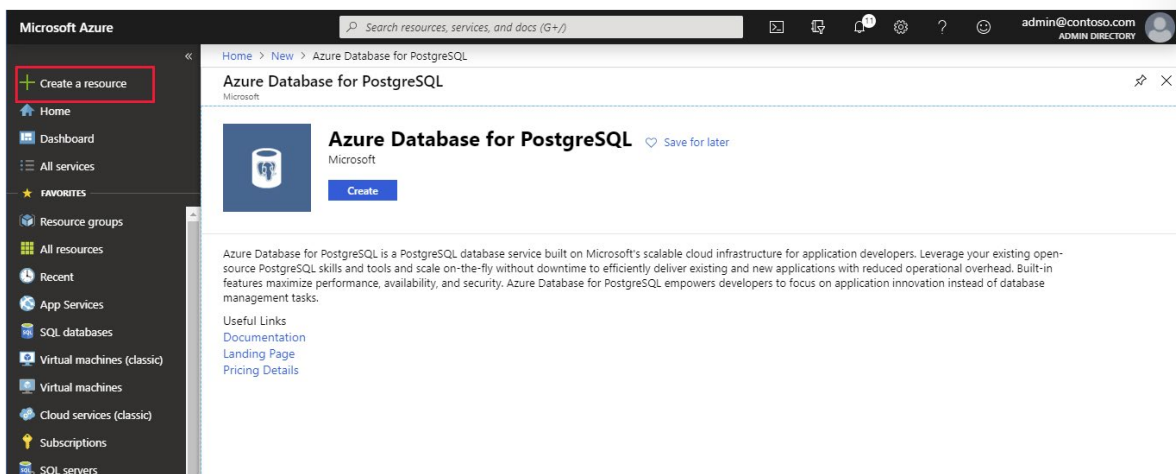
- Create a PostgreSQL server using Azure Database for PostgreSQL.
- Perform an online migration of a PostgreSQL database to Azure Database for PostgreSQL using the Azure Database Migration Service.
- Perform an offline migration using PostgreSQL tools.

Creating a PostgreSQL server with Azure Database for PostgreSQL

You create an instance of the Azure Database for PostgreSQL service using the Azure portal. If you need to create a number of instances of this service, you script the process by using the Azure CLI.

Create an Azure Database for PostgreSQL, Single Server instance using the portal

In the Azure portal, select the **Create a resource** command in the left menu bar, and search for **Azure Database for PostgreSQL** in the Azure Marketplace.



Click **Create**, and select the version you require—either **Single server** or **Hyperscale (Citius)**. The following steps assume you selected **Single server**. We'll cover **Hyperscale (Citius)** later.

The screenshot shows a Microsoft Azure portal page titled "Select Azure Database for PostgreSQL deployment option". Under the heading "How do you plan to use the service?", there are two cards. The left card is for "Single server", described as "Best for broad range of traditional transactional workloads." and "Enterprise ready, fully managed community PostgreSQL server with up to 64 vCores, optional geospatial support, full-text search and more." It has a "Create" button highlighted with a red box and a "Learn more" link. The right card is for "Hyperscale (Citius) server group - PREVIEW", described as "Best for ultra-high performance and data needs beyond 100GB." and "Ideal for multi-tenant applications and real-time analytical workloads that need sub-second response. Supports both transactional/operational workloads and hybrid transactional analytics workloads." It also has a "Create" button and a "Learn more" link.

On the **Single server** page, enter the details for the service. These details include:

- **Server name.** This must be a unique name between 3 and 63 characters, containing only lowercase letters, numbers, and hyphens.
- **Data source.** If you're creating a new server for migration purposes, select **None**. The **Backup** option enables you to restore a backup taken from another instance of Azure Database for PostgreSQL into this service.
- **Admin username.** This is the name of a user account that you'll create with administrative privileges. Azure creates some accounts for its own use, while other names are restricted—you can't use **azure_superuser**, **azure_pg_admin**, **admin**, **administrator**, **root**, **guest**, **public**, or any name that starts with **pg_**.
- **Password.** This must be between 8 and 128 characters. It must contain a mixture of uppercase and lowercase letters, numbers, and non-alphanumeric characters. Azure Database for PostgreSQL currently only supports password authentication; integration with Azure Active Directory is not yet available.
- **Version:** Select the version that corresponds to the on-premises database that you're migrating.
- **Compute + storage.** Select **Configure server** to set the pricing tier and specify the resources that you require for the service. The options were covered in Lesson 1. Remember that, if you select the **General purpose** or **Memory optimized** pricing tiers, you can scale up and down the number of virtual processor cores later. However, you can't reduce the amount of storage—it can only increase after the server has been created.

Single server
Microsoft

Basics Tags Review + create

Create an Azure Database for PostgreSQL server. [Learn more](#)

Project details

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

* Subscription

* Resource group [Create new](#)

Server details

Enter required settings for this server, including picking a location and configuring the compute and storage resources.

* Server name

* Data source

* Admin username

* Password

* Confirm password

* Location [Supported Locations](#)

* Version

Compute + storage **Basic**
2 vCores, 50 GB storage
[Configure server](#)

[Review + create](#) [Next: Tags >](#)

Click **Review + Create** to deploy the service. Deployment will take several minutes.

After the service has been deployed, select the **Connection security** option and add the appropriate firewall rules to enable clients to connect, as described in the topic *Client Connectivity* in Lesson 1. You must also select the **Allow access to Azure services** option.

Create an Azure Database for PostgreSQL, Hyperscale (Citus) instance using the portal

If you need to create a HyperScale (Citus) instance of Azure Database for PostgreSQL, the process is similar to creating a Single Server instance, except you will be presented with some additional options when you specify the **Compute + storage** resources. Specifically, you must provide the following information:

- The number of worker nodes. Azure Database for PostgreSQL currently supports from 2 to 20 nodes.
- The configuration for the worker nodes. Each worker node has the same resources. You select the number of processor cores (up to 32 per node) and storage (up to 2 TiB per node).
- The size of the coordinator node (up to 32 processor cores, and 2 TiB storage).

Create an Azure Database for PostgreSQL instance using the Azure CLI

You can create an instance of Azure Database for PostgreSQL using the `az postgres server create` command. The statement below shows an example that creates a single server instance. Most of the parameters are self-explanatory, except for the following:

- **sku-name.** You construct this from a combination of the pricing tier (*B* for Basic, *GP* for General Purpose, and *MO* for Memory Optimized), the compute generation (Gen4 or Gen5), and the number of virtual CPU cores. In the example below, the server is created using the General Purpose pricing tier, with 4 CPU cores of the Gen5 generation.
- **storage-size.** This is the amount of disk storage required, specified in megabytes. The following example allocates 10 gigabytes:

```
az postgres server create \  
  --name contoso-postgresql-server \  
  --resource-group postgresqlrg \  
  --admin-user contosoadmin \  
  --admin-password 7Hh7*ku5k$$$£jhk \  
  --sku-name GP_Gen5_4 \  
  --storage-size 10240
```

Perform online migration

You can do an online migration from an on-premises PostgreSQL installation to Azure Database for PostgreSQL with the Azure Database Migration Service.

In the online scenario, the Azure Database Migration Service copies all of your existing data to Azure, and then continuously performs a synchronization operation from the source database. Any new transactions that are performed against the on-premises system are copied to the new database in Azure. This process continues until you've reconfigured your client applications to use the new database in Azure, at which point you terminate the synchronization operation.

Configure the source server and export the schema

The first step in performing an online migration is to prepare the source server to support full write-ahead logging. On the source server, edit the `postgresql.conf` file and configure the following write-ahead logging parameters. To change these parameters, you restart the server—only do this when the system is expected to be quiescent:

```
wal_level = logical  
max_replication_slots = 5  
max_wal_senders = 10
```

After you've restarted the server, export the schema for the source database using the `pg_dump` utility:

```
pg_dump -o -h [server host] -U [user name] -d [database name] -s > db_schema.sql
```

Finally, make a list of all the extensions that your database uses. You'll need to enable these extensions in the target database. To do this, you either use the `\dx` `psql` command, or run the following query:

```
SELECT *
FROM pg_extension;
```

Create a target database and import the schema

The next stage is to create a target database in your Azure Database for PostgreSQL service. You use a familiar tool such as pgAdmin to connect to the server, or you might use the Azure CLI in the following example:

```
az postgres db create \
  --name [database name] \
  --server-name [server name] \
  --resource-group [azure resource group]
```

On the target database, enable any extensions used by the sources database.

Import the schema into the target database. On the machine holding the db_schema.sql file, run the following command:

```
psql -h [Azure Database for PostgreSQL host] -U [user name] -d [database name] -f db_schema.sql
```

Remove all foreign key references in the target database. You need this step because the data won't necessarily be migrated in any specific sequence, possibly prompting referential integrity violations that will cause the migration process to fail. However, you should make a record of all the foreign keys as you'll need to recreate them later. Run the following SQL statement using the psql utility to find all the foreign keys in your database and generate a script that removes them:

```
SELECT Queries.tableName
       ,concat('alter table ', Queries.tableName, ' ', STRING_AGG(concat('DROP CONSTRAINT ', Queries.
foreignkey), ',')) as DropQuery
       ,concat('alter table ', Queries.tableName, ' ',
               STRING_AGG(concat('ADD CONSTRAINT ', Queries.foreignkey, ' FOREIGN KEY
(' , column_name, '), ' REFERENCES ', foreign_table_name, '(' , foreign_column_name, ') '), ',')) as AddQuery
FROM
(SELECT
tc.table_schema,
tc.constraint_name as foreignkey,
tc.table_name as tableName,
kcu.column_name,
ccu.table_schema AS foreign_table_schema,
ccu.table_name AS foreign_table_name,
ccu.column_name AS foreign_column_name
FROM
information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage AS kcu
  ON tc.constraint_name = kcu.constraint_name
  AND tc.table_schema = kcu.table_schema
JOIN information_schema.constraint_column_usage AS ccu
  ON ccu.constraint_name = tc.constraint_name
  AND ccu.table_schema = tc.table_schema
WHERE constraint_type = 'FOREIGN KEY') Queries
```

```
GROUP BY Queries.tablename;
```

Disable any triggers in the target database—there are two reasons for this:

- It helps to optimize the migration process as data is copied in.
- Triggers are often used to implement complex forms of referential integrity and, for the reasons described earlier, this type of integrity checking could fail while data is being transferred. Use the following SQL statement to find all the triggers in your database and generate a script that disables them:

```
SELECT concat ('alter table ', event_object_table, ' disable trigger ', trigger_name)
FROM information_schema.triggers;
```

[NOTE]

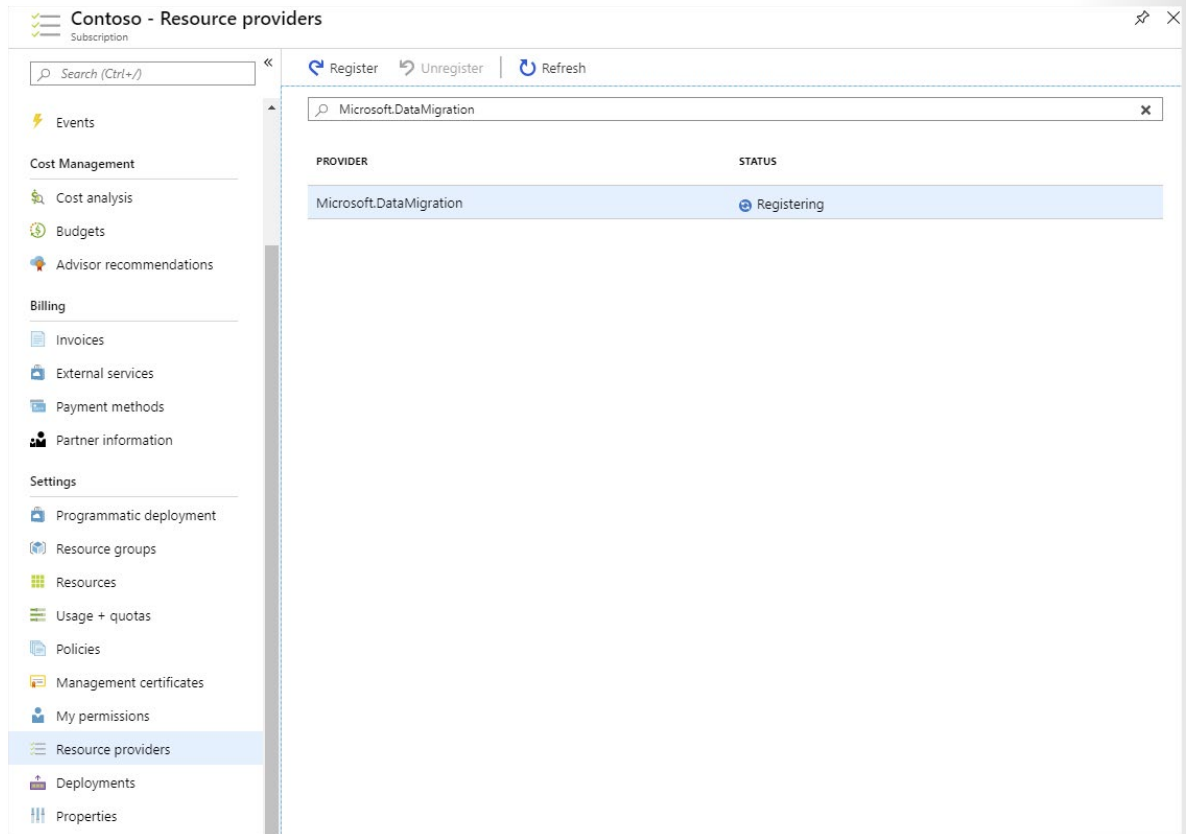
You can find the text for a trigger using the `\df+` command in psql.

Create an Azure Database Migration Service instance

In the Azure portal, you can now create an instance of the Azure Database Migration Service.


Before you create an instance of the Azure Database Migration Service, you must register the **Microsoft.DataMigration** resource provider with your subscription. You can do this as follows:

1. In the left menu bar of the Azure portal, select **All services**.
2. On the **All services** page, select **Subscriptions**.
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. Select **Register**, and wait for the **Status** to change to **Registered**. You might need to select **Refresh** to see which status to change.



When the resource provider is registered, you can create the service. Select the **Create a resource** command in the left menu bar, and search for **Azure Database Migration Service**.

Azure Database Migration Service
Microsoft



Azure Database Migration Service [Save for later](#)

Microsoft

[Create](#)

Once you've completed the step by step guidance for your migration scenario, proceed with creating a migration service by clicking the **Create** button below.

Additional resources:

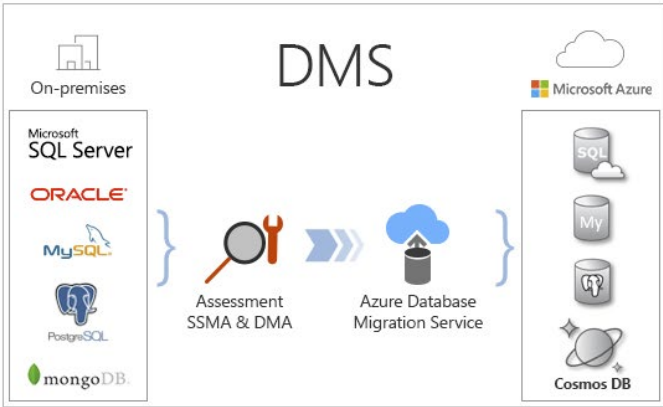
Use these tools to assess your database(s) for feature parity and potential compatibility issues.

- SQL Server on-premise database(s): [Data Migration Assistant \(DMA\)](#)
- Migrating from Oracle: [SQL Server Migration Assistant \(SSMA\)](#)

Useful Links

[Documentation](#)

[Privacy Statement](#)



On the **Create Migration Service** page, enter a name for your instance of the service, specify the subscription—this should be the same subscription that you registered the resource provider against—resource group, and location. You must also provide a virtual network because the Database Migration Service depends on a firewall created for this virtual network to provide the necessary protection. If you're migrating databases from Azure virtual machines, you might be able to place the Database Migration Service in the same virtual network used by these virtual machines. The pricing tier determines the number of virtual processor cores that are available to the service. If you want to do an online migration, you must select the **Premium** tier; the **Standard** tier only supports offline migration.

The screenshot shows the 'Create Migration Service' wizard in the Azure portal. The 'Pricing tier' section is highlighted with a red box, showing the 'Premium' tier selected. The 'Premium' tier is described as 'For offline and online migrations with minimal downtime' and '4 vCores'. The estimated monthly cost is \$0.00 USD. The 'Standard' tier is also visible, described as 'For large data sizes' and '1 vCores, 2 vCores, 4 vCores'.

Wait for the service to be deployed before continuing. This operation will take a few minutes.

Create a migration project using the Database Migration Service

You can now use your Database Migration Service instance to perform an online migration. To do this, you create a new Database Migration project. Go to the page for your migration service instance and select **New Migration Project**.

The screenshot shows the Azure Database Migration Service instance page. The '+ New Migration Project' button is highlighted with a red box. A success message is displayed: 'Great job! Your database migration service was successfully created. You can create your first migration project now.' The page also shows the service details, including the resource group, virtual network, subscription, and SKU (Premium: 4 vCores).

NAME	SOURCE	TARGET	CREATED
No database migration projects to display			

On the **New migration project** page, set the source server type to **PostgreSQL**, set the target server type to **Azure Database for PostgreSQL**, and select **Online data migration**. The **Type of activity** page lists the steps you must take on the source server to enable online migration. The text at the bottom of the **New migration project** page describes the process for migrating the schema to the target.

New migration project

Project name
migratedatabase ✓

* Source server type
PostgreSQL

* Target server type
Azure Database for PostgreSQL

* Choose type of activity
Online data migration >

To successfully use Database Migration Service (DMS) to migrate data, you need to:

1. Migrate schema using `pg_dump -o -h hostname -U db_username -d db_name -s > your_schema.sql`
2. Remove foreign keys in schema at target Azure Database for PostgreSQL
3. Disable triggers at target Azure Database for PostgreSQL
4. Provision Database Migration Service and create a migration task

Please refer to [this tutorial](#) for more details.

Create and run activity

Type of activity

Choose type of activity
Online data migration

Use this option to migrate databases that must be accessible and continuously updated during migration.

To continuously replicate data changes from your source to your target, please implement the steps below on your source server. These steps can be implemented anytime between the moment you create a project and the moment you start a migration activity. After your migration is complete, you will reverse those preparation steps.

- Azure Database for PostgreSQL supports community edition and the source PostgreSQL server must match the same major version that Azure Database for PostgreSQL supports. For example, upgrading from PostgreSQL 9.5.x to 9.6.x is not supported.
- Enable logical replication in the `postgresql.conf` file.

Save

Verify that you have completed these steps, then select **Create and run activity**.

Create and run a migration activity

The new migration project starts a wizard that guides you through the process. You provide the following details:

- On the **Add Source Details** page, the address of the source server, the source database, and an account that can connect to this database and retrieve the data. The account must have **SUPERUSER** privileges to perform migration.

- On the **Target details** page, specify the address of your Azure Database for PostgreSQL service, the database into which you want to migrate the data, and the details of an account that has administrative rights.
- On the **Map to target databases** page, select the source database and target database. You can migrate a single database or multiple databases.
- On the **Migration settings** page, specify any additional settings that you want to configure, such as the maximum number of tables to load in parallel.
- On the **Migration summary** page, enter a name for the activity, and then select **Run migration**.

Migration Wizard		Migration summary	
1	Select source	✓	
2	Select target	✓	
3	Select databases	✓	
4	Configure migration settings	✓	
5	Summary	>	
		<p>Activity name Migrate_Northwind_Database ✓</p> <p>Target server name azurepostgresql.postgres.database.azure.com</p> <p>Target server version Azure Database for PostgreSQL 10.9</p> <p>Source server name 52.142.160.226</p> <p>Source server version PostgreSQL 10.10 (Ubuntu 10.10-1.pgdg18.04+1)</p> <p>Database(s) to migrate 1 of 2</p> <p>Run migration</p>	

The activity status page appears showing the progress of the migration, and any errors that have occurred. If the migration fails, you correct the issues and retry the activity. If you're doing an online migration, the status changes to **Ready to cutover** after the existing data has been transferred. However, the activity continues running, to transfer any additional changes that appear while applications are still actively using the original database.

Migrate_Northwind_Database

Refresh Retry Stop migration Delete activity Download report

Source server 52.142.160.226	Source version PostgreSQL 10.0	Source databases 1
Target server azurepostgresql.postgres.database.azure.com	Target version Azure Database for PostgreSQL 10.9	Type of activity Online
Activity status Running		Duration 00:01:32

DATABASE NAME	STATUS	MIGRATION DETAILS	DURATION	ESTIMATED APPLICATION DOWNTIME ⓘ	FINISH DATE
northwind	Running	Ready to cutover	00:01:32	---	---

Reinstate foreign keys and triggers

At this point, you've transferred the data, and applications can start using it. You should recreate the foreign keys that you removed prior to migrating the data, and reinstate any triggers. If some applications are still connected to the original database, write-ahead logging ensures that the target database in Azure is kept up to date. Write-ahead logging will not be adversely affected by foreign keys and triggers.

Cut over to the new database

When all applications have been switched to the new database, you complete the migration process and cut over to the new database. On the activity status page, select the name of the database being migrated to see a summary of the work performed.

northwind

Refresh Start Cutover

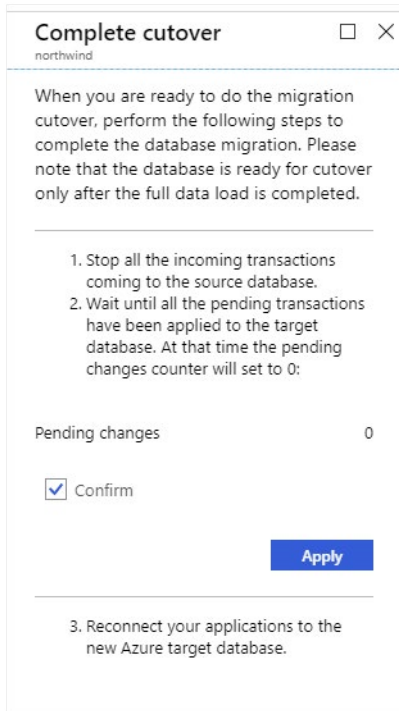
Source database name northwind	Full load completed 14	Incremental updates 0	Pending changes 0
Target database name contosodb	Full load queued 0	Incremental inserts 0	Applied changes 0
Database status Running	Full load loading 0	Incremental deletes 0	Tables in error state ⓘ 0
Migration details Ready to cutover	Full load failed 0		

Full load Incremental data sync

14 item(s) prev Page 1 of 2 next

TABLE NAME	STATUS	COMPLETED	ROWS	DURATION
public.categories	Completed	22/08/2019, 13:05:42	8	00:00:01
public.customer_customer_demo	Completed	22/08/2019, 13:05:42	0	00:00:00
public.customer_demographics	Completed	22/08/2019, 13:05:43	0	00:00:00
public.customers	Completed	22/08/2019, 13:05:45	91	00:00:02
public.employee_territories	Completed	22/08/2019, 13:05:45	49	00:00:02
public.employees	Completed	22/08/2019, 13:05:46	9	00:00:02
public.order_details	Completed	22/08/2019, 13:05:48	2155	00:00:02
public.orders	Completed	22/08/2019, 13:05:48	830	00:00:02
public.products	Completed	22/08/2019, 13:05:49	77	00:00:02
public.region	Completed	22/08/2019, 13:05:49	4	00:00:02

Select **Start cutover**. You'll see a page asking you to confirm that the operation is complete. At this point, any remaining entries from the write-ahead log for the source database will be drained, and updates will stop. Any further changes made to the source database will not be propagated.



Demonstration: Performing an online migration

In this demonstration, you will see how to migrate a sample database running on a PostgreSQL server to Azure Database for PostgreSQL. The sample database used is a version of the Northwind Traders database, adapted for PostgreSQL. The database contains foreign keys.

The main tasks in this demonstration are:

1. Configure the source server and export the schema
2. Create a target database and import the schema
3. Create an Azure Database Migration Service instance
4. Create and run a migration project using the Database Migration Service
5. Reinstate foreign keys and triggers
6. Modify data, and cutover to the new database

Setup

This demonstration uses a preconfigured virtual machine with PostgreSQL and the sample database already installed. The sign-in account for the virtual machine and the PostgreSQL database is *azureuser*, with password *Pa55w.rdDemo*. To setup this virtual machine, perform the following steps:

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. 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-workshop
```
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 up to 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**. Make a note of the IP address.

7. Move to the *workshop/migration_samples/setup/postgresql/northwind* folder.


```
cd ~/workshop/migration_samples/setup/postgresql/northwind
```
8. Run the following command to create the northwind database on the Azure virtual machine. Replace *[nn.nn.nn.nn]* with the IP address of the virtual machine.


```
bash create_db.sh [nn.nn.nn.nn]
```

Enter the password **Pa55w.rd** when prompted; this will occur three times. The script will take a couple of minutes to run.

Task 1: Configure the source server and export the schema

1. Using the Azure Cloud shell, connect to the virtual machine containing the PostgreSQL server and database. Replace *[nn.nn.nn.nn]* with the IP address of the virtual machine. Specify the password **Pa55w.rdDemo** when prompted.


```
ssh azureuser@[nn.nn.nn.nn]
```
2. On the virtual machine, switch to the root account. Enter the password for the *azureuser* user if prompted.


```
sudo bash
```
3. Move to the directory */etc/postgresql/10/main*:


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

- Open the `postgresql.conf` file using the nano editor:

```
nano postgresql.conf
```

- Scroll to the bottom of the file and find the following parameters. Verify that they are set to the values specified below. Change them if not:

```
wal_level = logical
max_replication_slots = 5
max_wal_senders = 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`.

- Restart the PostgreSQL service:

```
service postgresql restart
```

- Verify that Postgres 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.
```

- Leave the `root` account and return to the `azureuser` account:

```
exit
```

- You are going to migrate a database named **northwind**. This database contains the details of products, customers, and orders for the Northwind Traders organization. Run the following command to connect to the database:

```
psql -d northwind
```

- At the `northwind=#` prompt, enter the `\d` command to list the tables in the database. You should see the following tables listed:

```
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | categories     | table | azureuser
public | customer_customer_demo | table | azureuser
public | customer_demographics | table | azureuser
public | customers       | table | azureuser
public | employee_territories | table | azureuser
```

```
public | employees      | table | azureuser
public | order_details     | table | azureuser
public | orders             | table | azureuser
public | products           | table | azureuser
public | region             | table | azureuser
public | shippers           | table | azureuser
public | suppliers           | table | azureuser
public | territories         | table | azureuser
public | us_states          | table | azureuser
(14 rows)
```

12. Run the following commands to view the data in the **orders** and **customers** tables:

```
SELECT * FROM orders;
SELECT * FROM customers;
```

13. Exit psql with the `\q` command.

14. At the bash prompt, run the following command to export the schema for the **northwind** database to a file named **northwind_schema.sql**

```
pg_dump -o -d northwind -s > northwind_schema.sql
```

Task 2. Create a target database and import the schema

1. Switch to the Azure portal.
2. Click + **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, click **Create**.
5. On the **Select Azure Database for PostgreSQL deployment option** page, in the **Single server** box, click **Create**.
6. On the **Single server** page, enter the following details, and then click **Review + create**:

Property	Value
Resource group	Click Select existing , and specify the resource group that you used for the virtual machine in the Setup section of this demo.
Server name	northwindnnn , where <i>nnn</i> is a suffix of your choice to make the server name unique
Data source	None
Admin username	northwindadmin
Password	Pa55w.rdDemo
Confirm password	Pa55w.rdDemo
Location	Select your nearest location
Version	10
Compute + storage	Click Configure server , select the Basic pricing tier, and then click OK

You will see a series of messages as each table is created. You will also see a number of errors: **role "azureuser" does not exist**. This error occurs because the script tries to change the owner of each table to **azureuser** (the original owner in the **northwind** database). This user doesn't exist in the **azurenorthwind** database. Instead, the tables will be owned by the **northwindadmin** account. You can ignore these errors.

17. 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 **azurenorthwind** database. You will run the *dropkeys.sql* script shortly:

```
psql -h northwind[nnn].postgres.database.azure.com -U northwindadmin@northwind[nnn] -d azurenorthwind -f findkeys.sql -o dropkeys.sql -t
```

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

18. 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 northwind[nnn].postgres.database.azure.com -U northwindadmin@northwind[nnn] -d azurenorthwind -f createkeys.sql -o addkeys.sql -t
```

19. Run the *dropkeys.sql* script to remove the foreign keys from the **azurenorthwind** database:

```
psql -h northwind[nnn].postgres.database.azure.com -U northwindadmin@northwind[nnn] -d azurenorthwind -f dropkeys.sql
```

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

Task 3. Create an Azure Database Migration Service instance

[NOTE]

Only perform this task if you didn't perform the equivalent demonstration in Module 2; you can reuse the same Database Migration Service instance for this demonstration.

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

Property	Value
Service name	postgresql_migration_service

Property	Value
Select a resource group	Specify the same resource group that you used for the Azure Database for PostgreSQL service
Location	Select your nearest location
Virtual network	Create a new virtual network named migration-servicevnet
Pricing tier	Premium, with 4 vCores

9. Wait while the Database Migration Service is created.

Task 4. Create and run a migration project using the Database Migration Service

1. In the Azure portal, go to the page for the **postgresql_migration_service** Database Migration Service.
2. Click **New Migration Project**.
3. On the **New migration project** page, enter the following details, and then click **Create and run activity**.

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

4. When the **Migration Wizard** starts, on the **Add Source Details** page, enter the following details, and then click **Save**.

Property	Value
Source server name	Specify the IP address of the virtual machine running PostgreSQL
Server port	5432
Database	northwind
User Name	azureuser
Password	Pa55w.rd

5. on the **Target details** page, enter the following details, and then click **Save**.

Property	Value
Target server name	northwind[nnn].postgres.database.azure.com
Database	azurenorthwind
User Name	northwindadmin@northwind[nnn]
Password	Pa55w.rdDemo

6. on the **Map to target databases** page, select the **northwind** database and map it to **azurenorthwind**. Deselect the **postgres** database. Click **Save**.
7. On the **Migration settings** page, expand the **northwind** dropdown, expand the **Advanced online migration settings dropdown**, verify that **Maximum number of instances to load in parallel** is set to 5, and then click **Save**.

8. On the **Migration summary** page, in the **Activity name** box type **PostgreSQL_Migration_Activity**, and then click **Run migration**.
9. On the **PostgreSQL_Migration_Activity** page, click **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**.

Task 5. Reinstate foreign keys

1. Switch back to the bash prompt for the PostgreSQL virtual machine.
2. Run the following command to recreate the foreign keys in the **azurenorthwind** database. You generated the **addkeys.sql** script earlier:

```
psql -h northwind[nnn].postgres.database.azure.com -U northwindadmin@northwind[nnn] -d azurenorthwind -f addkeys.sql
```

You will see a series of **ALTER TABLE** statements as the foreign keys are added. There shouldn't be any error messages.

Task 6. Modify data, and cutover to the new database

1. Return to the **PostgreSQL_Migration_Activity** page in the Azure portal.
2. Click the **northwind** database.
3. On the **northwind** page, verify that the status for all tables is marked as **Completed**.
4. Click **Incremental data sync**. Verify that the status for every table is marked as **Syncing**.
5. Switch back to the bash prompt for the PostgreSQL virtual machine.
6. Run the following command to connect to the **northwind** database running using PostgreSQL on the virtual machine. The password is **Pa55w.rd**:

```
psql northwind
```

7. Execute the following SQL statements to display, and then remove orders 10248, 10249, and 10250 from the database.

```
SELECT * FROM orders WHERE order_id IN (10248, 10249, 10250);
SELECT * FROM order_details WHERE order_id IN (10248, 10249, 10250);
DELETE FROM order_details WHERE order_id IN (10248, 10249, 10250);
DELETE FROM orders WHERE order_id IN (10248, 10249, 10250);
```

8. Close the *psql* utility with the **\q** command.
9. Return to the **northwind** page in the Azure portal, and then click **Refresh**. Verify that the **public.order_details** table indicates that 8 rows have been deleted, and 3 rows have been removed from the **public.orders** table.
10. Click **Start cutover**.
11. On the **Complete cutover** page, select **Confirm**, and then click **Apply**. Wait until the status changes to **Completed**.
12. Switch back to the bash prompt for the PostgreSQL virtual machine.

13. Run the following command to connect to the **azurenorthwind** database running using your Azure Database for PostgreSQL service. The password is **Pa55w.rdDemo**:

```
psql -h northwind[nnn].postgres.database.azure.com -U northwindadmin@northwind[nnn] -d azurenorthwind
```

14. Execute the following SQL statements to display the orders and order details in the database. Quit after the first page of each table:

```
SELECT * FROM orders;  
SELECT * FROM order_details;
```

15. Run the following SQL statements to display the orders and details for orders 10248, 10249, and 10250.

```
SELECT * FROM orders WHERE order_id IN (10248, 10249, 10250);  
SELECT * FROM order_details WHERE order_id IN (10248, 10249, 10250);
```

Both queries should return 0 rows. They have been removed as part of the synchronization process, before cutover.

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

17. Close the connection to the virtual machine and return to the Cloud Shell:

```
exit
```

Perform offline migration

An offline migration takes a “snapshot” of the source database at a particular point in time, and copies that data to the target database. Any changes made to the source data after the snapshot has been taken will not be reflected in the target database.

You have at least two options if you want to perform an offline migration to Azure Database for PostgreSQL, or to a PostgreSQL server running elsewhere, such as an Azure virtual machine:

- Export the schema and data from the source database using the *pg_dump* utility, and import the schema and data into the target database using the *psql* utility. This technique enables you to modify, reformat, and clean the schema and data if necessary, before transferring it to the target database.
- Dump the data from the source database, again using *pg_dump*, and restore the data into the target database using *pg_restore*. This technique is quicker than using export and import, but the data is dumped in a format that can't easily be changed. Use this approach if you don't need to tweak the schema or data.

[NOTE]

You can't currently use the Azure Database Migration Service to perform an offline migration of a PostgreSQL database.

Migrate by using export and import

Perform the following steps to migrate a database by using the export and import approach.

1. Export the schema by using the *pg_dump* command from a bash prompt:

```
pg_dump -o -h [source database server] -U [user name] -d [database] -s > db_schema.sql
```

2. Export the data to another file with the `pg_dump` command:

```
pg_dump -o -h [source database server] -U [user name] -d [database] -a > db_data.sql
```

At this point, `db_schema.sql` and `db_data.sql` are SQL scripts that you modify using a text editor.

3. Create the target database in Azure Database for PostgreSQL. You do this with the Azure CLI:

```
az postgres db create \
  --name [database name] \
  --server-name [server name] \
  --resource-group [azure resource group]
```

4. Import the schema into the target database with the `psql` command:

```
psql -d [target database name] -h [server name in Azure Database for PostgreSQL] -U [user name] -f
db_schema.sql
```

5. Import the data into the target database with the `psql` command:

```
psql -d [target database name] -h [server name in Azure Database for PostgreSQL] -U [user name] -f
db_data.sql
```

Migrate by using backup and restore

These steps describe the process to migrate a database by using backup and restore.

1. Back up the database—from a bash prompt, run the following command. Specify the name of a user who has the necessary privileges to back up the database:

```
pg_dump [database name] -h [source database server] -U [user name] -Fc > database_backup.bak
```

2. Create the target database in Azure Database for PostgreSQL:

```
az postgres db create \
  --name [database name] \
  --server-name [server name] \
  --resource-group [azure resource group] \
```

3. Restore the backup into the new database with the `pg_restore` command from a bash prompt. Specify the name of a user with administrative rights in your Azure Database for PostgreSQL service:

```
pg_restore -d [target database name] -h [server name in Azure Database for PostgreSQL] -Fc -U [user
name] database_backup.bak
```

Lesson 2: Summary

In this lesson, you learned how to:

- Create a PostgreSQL server using Azure Database for PostgreSQL.

- Perform an online migration of a PostgreSQL database to Azure Database for PostgreSQL using the Azure Database Migration Service.
- Perform an offline migration using PostgreSQL tools.

Knowledge Check

Multiple choice

You want to perform an online migration of an on-premises PostgreSQL database to Azure Database for PostgreSQL.

Which tools should you use?

- The Azure Database Migration Service, using the Standard pricing tier.
- The Azure Database Migration Service, running using the Premium pricing tier.
- The pg_dump and pg_restore utilities.
- The pg_dump and psql utilities.
- The pgAdmin utility.

Multiple choice

You should disable referential integrity checks and triggers in the target database before performing an online migration.

True or false?

- True.
- False.

Application Migration

Lesson 3: Application migration

In this lesson, you'll learn how reconfigure your applications so that they connect to your databases running in Azure Database for PostgreSQL. You'll see how to test your applications, to verify that the system is functioning correctly.

Learning objectives

By the end of this lesson, you will be able to:

- Create the necessary roles and assign the appropriate privileges in your databases running under Azure Database for PostgreSQL.
- Reconfigure applications to connect to databases running in your Azure Database for PostgreSQL service.
- Test and verify that applications using your databases running under Azure Database for PostgreSQL are still functioning correctly.

Create roles and assign privileges

Your original on-premises server and database will contain roles that define the privileges associated with users, the operations they can do, and the objects they perform these operations over. Azure Database for PostgreSQL uses the same authentication and authorization mechanisms as PostgreSQL running on-premises.

When you transfer a PostgreSQL database to Azure Database for PostgreSQL using the Azure Database Migration Service, the roles and role assignments aren't copied. You must manually recreate the necessary roles and user accounts for the administrator and users of the tables in the target database. You use the `psql` or `pgAdmin` utilities to do these tasks. Run the `CREATE ROLE` command. You use the `GRANT` command to assign the necessary privileges to a role. For example:

```
CREATE ROLE myuseraccount WITH LOGIN NOSUPERUSER CREATEDB PASSWORD 'mY!P@ss0rd';
GRANT ALL PRIVILEGES ON DATABASE mydatabase TO myuseraccount;
```

[NOTE]

You also use the `createuser` command from the bash prompt to create PostgreSQL roles.

To view the existing roles in the on-premises database, run the following SQL statement:

```
SELECT rolname
FROM pg_roles;
```

You can use the `\du` command in the `psql` utility to display the privileges assigned to roles.

List of roles		
Role name	Attributes	Member of
azureuser	Superuser, Create DB	{ }
myuseraccount	Create DB	{ }
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{ }

Note that Azure Database for PostgreSQL adds some roles of its own. These roles include `azure_pg_admin`, `azure_superuser`, and the administrator user that you specified when you created the service. You sign in using your administrative accounts, but the other two roles are reserved for use by Azure—you shouldn't attempt to use them.

Reconfigure applications

Reconfiguring an application to connect to Azure Database for PostgreSQL is a straightforward process. However, it's more important to determine a strategy for migration applications.

Considerations when reconfiguring PostgreSQL applications

In a corporate environment, you might have many applications running against the same PostgreSQL databases. There could be a large number of users running these applications. You want to be assured that, when you switch from the existing system to Azure Database for PostgreSQL, your systems will still work, users can continue doing their jobs, and your business-critical operations remain operational. Module 1, Lesson 2, *Considerations for migration*, discussed many of the issues in general terms. When you migrate a PostgreSQL database to Azure, there are some specifics to note:

- If you're performing an offline migration, the data in the original PostgreSQL database and the new databases running on Azure can start to diverge quickly if the old database is still being used. An offline migration is suitable when you take a system out of operation entirely for a short while, and then switch all applications to the new system before starting up again. This approach might not be possible for a business-critical system. If you're migrating to PostgreSQL running on an Azure virtual machine, you configure PostgreSQL replication between your on-premises system and that running in Azure. Native PostgreSQL replication operates in one direction only, but third-party solutions are available that support bidirectional replication between PostgreSQL servers (these solutions won't work with Azure Database for PostgreSQL).
- If you're performing an online migration, the Azure Database for PostgreSQL service sets up replication from the on-premises database to the database running in Azure. After the initial data transfer, replication ensures that any changes made in the on-premises database are copied to the database in Azure, but not the other way round.

In both cases, you should ensure that you don't lose live data through an accidental overwrite. For example, in the online scenario, an application connected to the database running in Azure Database for PostgreSQL could have its changes blindly overwritten by an application still using the on-premises database. With this in mind, you should consider the following approaches:

- Migrate applications based on their workload type. An application that accesses the data for reading only can move safely to the database running in Azure Database for PostgreSQL, and will see all changes made by applications still using the on-premises database. You can also adopt the converse strategy if read-only applications don't require fully up-to-date data.
- Migrate users based on their workload type. This strategy is similar to the previous one, except that you might have users that only generate reports while others modify the data. You might have the same application configured to connect to the appropriate database according to user requirements.
- Migrate applications based on the datasets they use. If different applications utilize different subsets of the data, you might be able to migrate these applications independently of each other.

Reconfiguring an application

To reconfigure an application, you point it at the new database. Most well-written applications will isolate the connection logic, and this should be the only part of the code that requires changing. In many cases, the connection information might be stored as configuration information—you only need to update that information.

You'll find the connection information for your Azure Database for PostgreSQL service in the Azure portal, on the **Connection strings** page for your service. Azure provides the information for many common programming languages and frameworks.

The screenshot shows the 'azurepostgresq - Connection strings' page in the Azure portal. The left sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with sub-items: Connection security, Connection strings, Server parameters, Replication, Pricing tier, Properties, Locks, Export template), and Intelligent performance (with sub-items: Query Performance Insight, Performance recommendations). The main content area displays connection strings for the following languages/frameworks: ADO.NET (Npgsql), C++ (libpq), JDBC, Node.js, PHP, psql, Python, Ruby, and Web app. Each entry shows a template string with placeholders for database name, port, user, and password, and includes a copy icon.

Open network ports

As mentioned in Lesson 1 of this module, Azure Database for PostgreSQL is a protected service that runs behind a firewall. Clients can't connect unless their IP address is recognized by the service. You must add the IP addresses, or address block ranges, for clients running applications that need to connect to your databases.

Test and verify applications

Before you switch your applications and users to the new database, it's important to ensure that you've configured everything correctly.

Start by "dry-running" applications and connect each role to ensure the correct functionality is available.

Next, perform "soak tests" to mimic the typical number of users running representative workloads concurrently for a period of time. Monitor the system, and verify that you've allocated sufficient resources to your Azure Database for PostgreSQL service.

At this point, you can start to roll out the system to users. It might be beneficial to implement some form of "canary testing", where a small subset of users is transferred to the system unawares. This gives you an

unbiased opinion as to whether users are having the same, better, or worse experience with the new database.

Lesson 3: Summary

In this lesson, you learned how to:

- Create the necessary roles and assign the appropriate privileges in databases that run under Azure Database for PostgreSQL.
- Reconfigure applications to connect to databases running in your Azure Database for PostgreSQL service.
- Test and verify that applications using your databases running under Azure Database for PostgreSQL are still functioning correctly.

Knowledge Check

Multiple choice

You need to configure applications so they connect to your databases in Azure Database for PostgreSQL. Where can you find this information?

- On the Connection security page for your Azure Database for PostgreSQL service in the Azure portal.
- It will be the same as the connection information used to connect to databases in your on-premises PostgreSQL servers—but the server name will have the "Azure" prefix.
- On the Connection strings page for your Azure Database for PostgreSQL service in the Azure portal.
- You configure applications to connect using the ID and password of the Azure subscription in which the Azure Database for PostgreSQL was created.
- You must contact Microsoft, who will provide this information in a secure email.

Multiple choice

Database roles and privileges are transferred automatically from an on-premises PostgreSQL database to Azure Database for PostgreSQL when you use the Azure Database Migration Service. True or false?

- True.
- False.

Module summary

Module Summary

In this module, you learned about the benefits of migrating PostgreSQL workloads to Azure, and the features that Azure provides to help manage and optimize PostgreSQL systems. You saw how to create an instance of the Azure Database for PostgreSQL service, and then you migrated an on-premises PostgreSQL database to Azure. You also reconfigured an application that uses the database to connect to Azure.

Takeaways

In this module, you:

- Learned about the features and limitations of Azure Database for PostgreSQL.
- Migrated an on-premises PostgreSQL database to Azure Database for PostgreSQL.
- Reconfigured an application that used your on-premises PostgreSQL database to connect to Azure Database for PostgreSQL instead.

Module Lab Information

Answers

Multiple choice

You have customers running applications that access your PostgreSQL database. The operations these applications perform are typically queries that generate reports. These customers are located in Australia, Europe, and the United States. Your corporate headquarters is in the United States—this is where you insert, update, and delete most of the data.

How can you minimize query latency for your customers?

- Run copies of the database using Azure Database for PostgreSQL in Australia, Europe, and America. Configure bidirectional replication between all three sites so that they're kept synchronized with each other.
- Run copies of the database using Azure Database for PostgreSQL in Australia, Europe, and America. At regular intervals, back up the data at each site, and restore it on the other two sites.
- Implement read replicas with Azure Database for PostgreSQL. Make the United States office the master server, and replicate the data to replicas running in Australia and Europe.
- Host your database in an instance of the Azure Database for PostgreSQL service, and scale up to the maximum number of virtual processor cores.
- Use a dedicated high-bandwidth network connection between each customer and the database.

Explanation

Read-only replication is the most suitable solution. The customer workloads in Australia and Europe are read-intensive with few updates. The application can connect to the database in the United States on the rare occasions that it modifies data.

Multiple choice

You want to use the pgAdmin utility to monitor your database running in Azure Database for PostgreSQL. You're attempting to connect from your desktop computer. You're using the correct server name, protocol (SSL), username, and password, but you receive an error.

What might the problem be?

- You can't use pgAdmin to monitor a database running in Azure Database for PostgreSQL.
- The computer running the pgAdmin utility must be in the same virtual network as the Azure Database for PostgreSQL service instance.
- Check that you've added a firewall rule to your Azure Database for PostgreSQL service that allows connections from your desktop computer.
- Azure Database for PostgreSQL doesn't support SSL connections.
- Azure Database for PostgreSQL doesn't support username and password authentication.

Explanation

The most likely cause is that the firewall for your instance of Azure Database for PostgreSQL is blocking connection requests from your desktop computer. You should make sure there's a firewall rule that permits traffic from your computer.

Multiple choice

You want to perform an online migration of an on-premises PostgreSQL database to Azure Database for PostgreSQL.

Which tools should you use?

- The Azure Database Migration Service, using the Standard pricing tier.
- The Azure Database Migration Service, running using the Premium pricing tier.
- The pg_dump and pg_restore utilities.
- The pg_dump and psql utilities.
- The pgAdmin utility.

Explanation

The Azure Database Migration Service, Premium tier, transfers data from an active PostgreSQL database and uses the write-ahead log from the source database to ensure any changes that occur after the initial migration are applied to the target database.

Multiple choice

You should disable referential integrity checks and triggers in the target database before performing an online migration.

True or false?

- True.
- False.

Explanation

During migration, data can be transferred by multiple concurrent threads—the sequence in which data is copied isn't guaranteed. If referential integrity and triggers are active in the target database at this time, they might cause data inserts to fail.

Multiple choice

You need to configure applications so they connect to your databases in Azure Database for PostgreSQL. Where can you find this information?

- On the Connection security page for your Azure Database for PostgreSQL service in the Azure portal.
- It will be the same as the connection information used to connect to databases in your on-premises PostgreSQL servers—but the server name will have the "Azure" prefix.
- On the Connection strings page for your Azure Database for PostgreSQL service in the Azure portal.
- You configure applications to connect using the ID and password of the Azure subscription in which the Azure Database for PostgreSQL was created.
- You must contact Microsoft, who will provide this information in a secure email.

Explanation

The Connections strings page shows the relevant information for a variety of connectivity providers.

Multiple choice

Database roles and privileges are transferred automatically from an on-premises PostgreSQL database to Azure Database for PostgreSQL when you use the Azure Database Migration Service.

True or false?

True.

False.

Explanation

Roles and privileges are not transferred automatically. You must recreate them manually.



Module 4 Protecting monitoring and tuning

Module Introduction

Module 4: Protecting, monitoring and tuning a migrated database

Module introduction

In this module, you'll learn how to use the Azure features available for protecting a database and server against malicious users and disasters. This module also describes how to monitor and tune a database running in Azure Database for MySQL/PostgreSQL.

Learning objectives

By the end of this module, you'll be able to:

- Protect a database running in Azure Database for MySQL and Azure Database for PostgreSQL.
- Monitor and tune a database running in Azure Database for MySQL and Azure Database for PostgreSQL.

Protecting a database and server

Lesson 1: Protecting a database and server

Azure Database for MySQL/PostgreSQL runs in a network environment that's managed by Azure. You can configure Azure features, such as firewall rules and Advanced Threat Protection, to help ensure that your databases and servers are protected against malicious activity. Azure's backup and restore features will also help recover your system if there's a disaster. In this lesson, you'll learn how to use these features to keep databases running and help maintain business continuity.

Learning objectives

By the end of this lesson, you'll understand how to:

- Manage security in Azure Database for MySQL/PostgreSQL.
- Back up and restore a server.

Manage security in Azure Database for MySQL/PostgreSQL

PostgreSQL and MySQL have their own authentication and authorization mechanisms that control which users are allowed to access databases, and the privileges they have over items in those databases. You should continue to manage users and privileges in much the same way that you did before migration. Remember that you can use administrative tools, such as pgAdmin and MySQL Workbench, to connect to servers hosted by Azure.

However, Azure provides additional protection to your servers. This protection operates at three levels:

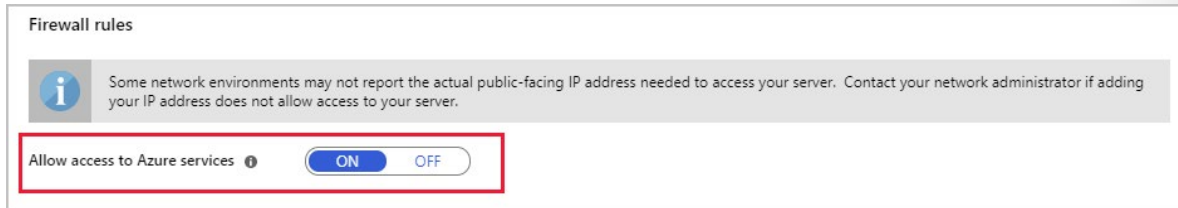
1. It controls access to the server, filtering traffic from unknown or untrusted sources.
2. It protects traffic, ensuring that it can't be manipulated or intercepted as it flows from a client to the server and back again.
3. It protects the server itself from common external threats.

The following sections discuss these items in more detail.

Filtering traffic with firewall rules

Azure Database for MySQL/PostgreSQL runs inside a firewall managed by Microsoft. By default, nothing can pass through this firewall. You add firewall rules to enable traffic from designated blocks of IP addresses, as described in the previous modules. It's recommended that you actively review the IP addresses that are allowed to send traffic at frequent intervals, and remove the IP addresses for clients that are no longer required.

If you're running other Azure services that need to use your databases, you must open the firewall to these services. In the Azure portal, on the **Connection security** page for your Azure Database for MySQL/PostgreSQL service, select the **Allow access to Azure services** action setting so that it's **On**.

**[NOTE]**

It can take up to five minutes for any changes you make to the firewall to become active.

In some situations, opening up your server to all Azure services might be too excessive. If you're running the General Purpose or Memory Optimized versions of Azure Database for MySQL/PostgreSQL, you filter traffic at the virtual network level using Azure virtual network rules. A virtual network rule enables you to allow traffic that originates from your own virtual networks to access the server. Traffic from other networks will be blocked.

VNET rules		+ Adding existing virtual network		+ Create new virtual network			
RULE NAME	VIRTUAL NETWORK	SUBNET	ADDRESS RANGE	ENDPOINT STATUS	RESOURCE GROUP	SUBSCRIPTION ID	STATE
northwind-vnet-...	migrationservi...	default	10.0.0.0/24	Not enabled	northwindrg	0fee6d28-2591-46...	Ready ...

If you need to script firewall maintenance tasks, you use the Azure CLI. The following examples, which add, delete, and display firewall rules, use the `az mysql` command that does operations against Azure Database for MySQL. If you're running a PostgreSQL command, use the corresponding `az postgres` commands instead—the parameters are the same:

```
# Allow access to clients in the range 13.83.152.0 to 13.83.152.15
```

```
az mysql server firewall-rule create \
  --resource-group [resource group name] \
  --server-name [Azure Database for MySQL server name] \
  --name FirewallRule1 \
  --start-ip-address 13.83.152.0 \
  --end-ip-address 13.83.152.15
```

```
# List all firewall rules
```

```
az mysql server firewall-rule list \
  --resource-group [resource group name] \
  --server-name [Azure Database for MySQL server name]
```

```
# Display the details of FirewallRule1
```

```
az mysql server firewall-rule show \
  --resource-group [resource group name] \
  --server-name [Azure Database for MySQL server name] \
  --name FirewallRule1
```

```
# Remove FirewallRule1. Clients in the address range for this rule will be denied access
```

```
az mysql server firewall-rule delete \
  --resource-group [resource group name] \
  --server-name [Azure Database for MySQL server name] \
```

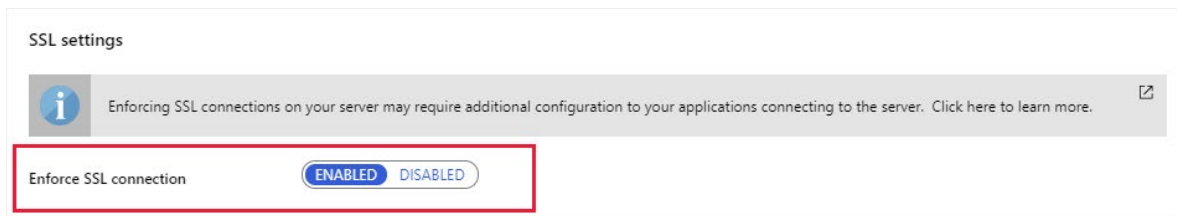
```
--name FirewallRule1
```

To enable access to Azure services, create a firewall rule with a `start-ip-address` and `end-ip-address` value of `0.0.0.0`.

You create and manage virtual network rules in a similar manner, using the `az mysql server vnet-rule` commands.

Protecting traffic using SSL

SSL protection for Azure Database for MySQL/PostgreSQL is enabled by default. You can disable and re-enable SSL using the **Enforce SSL connection** setting on the **Connection security** page for your Azure Database for MySQL/PostgreSQL service in the Azure portal:



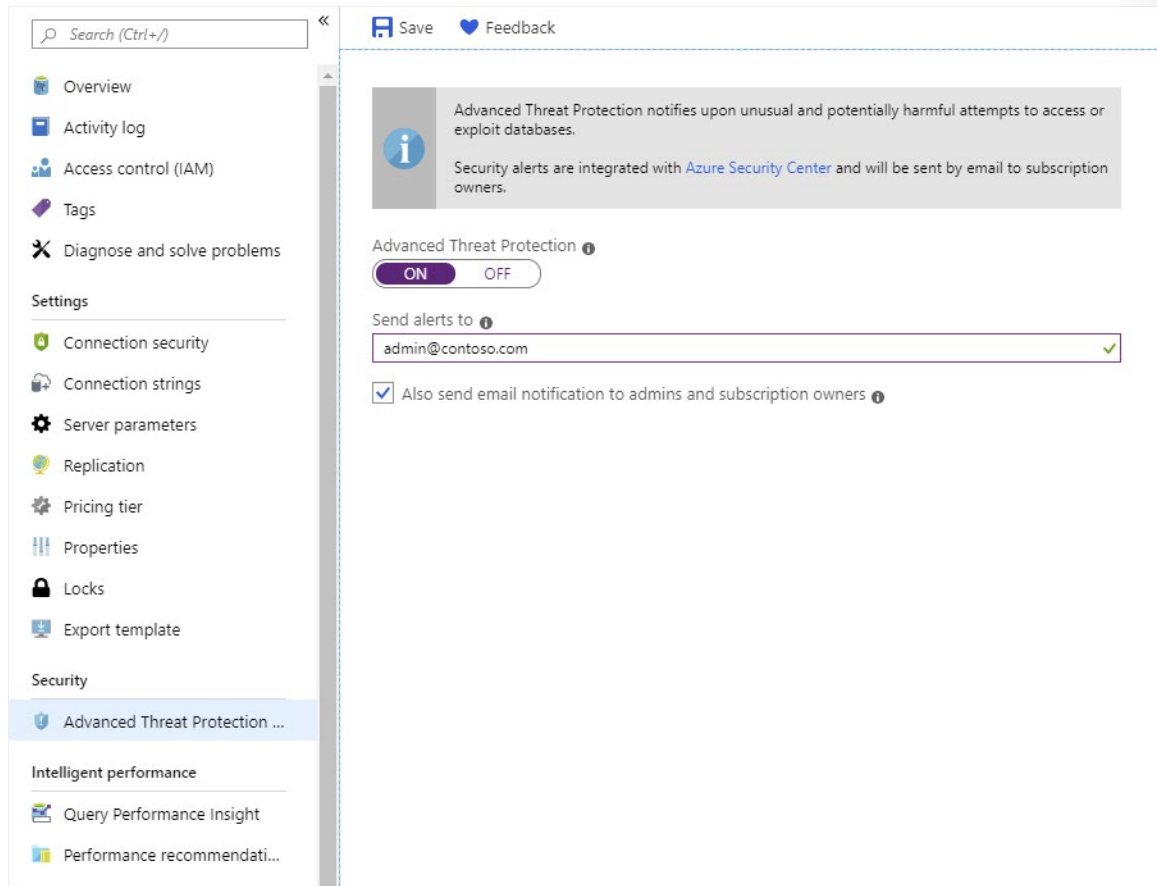
Protecting the server using Azure Advanced Threat Protection

Advanced Threat Protection is an additional layer of security provided by Azure. Advanced Threat Protection monitors access to your server and looks for patterns of unusual or potentially malicious behavior. When such behavior is detected, you arrange for an alert to be sent to specified email addresses.

The patterns of unusual activity detected include:

- Access from an unexpected or unusual location.
- Access from an unusual Azure data center.
- Access from an application that might be harmful, such as a recognized attack tool.
- A large number of failed logins in quick succession, indicating a possible brute-force attack.

You can enable Advanced Threat Protection from the **Advanced Threat Protection** page for your service in the Azure portal:



Backing up and restoring a server

The Azure Database for MySQL/PostgreSQL service automatically backs up your server according to the following schedule:

- A full backup is taken weekly—the first full backup occurs as soon as the server is created.
- A differential backup is taken twice a day.
- A transaction log backup is taken every five minutes.

The entire server is backed up. You can't back up individual databases, and you can't manually force a backup.

Setting backup options

You use these backups to restore to any point in time for which you have retained the backup files. By default, backups are held for seven days, but you can keep them for up to 35 days. You also specify how the backups are stored—locally redundant backups are held within the same region as the server, and geo-redundant backups are copied to data centers in other regions. The geo-redundant option is only available to servers in the General Purpose and Memory Optimized pricing tiers. You set the backup options on the **Pricing Tiers** page for your server in the Azure portal:

The screenshot shows the configuration interface for Azure Database for MySQL/PostgreSQL backups. It includes the following elements:

- Auto-growth**: A toggle switch set to "Yes".
- Backup retention period**: A slider set to 25 Days.
- Backup redundancy options**: Two options are shown: "Locally redundant" (Recover from data loss within region) and "Geo-redundant" (Recover from regional outage or disaster), which is selected with a blue checkmark.

Restoring a server

Azure Database for MySQL/PostgreSQL supports two types of server restore operations—point-in-time, and geo-restore. In both cases, the restore action creates a new server. The original server remains available. If you want applications to use the restored data, you must reconfigure them to use the new server. Additionally, you must remember to open the firewall of the new server to allow clients and services to connect.

[!IMPORTANT]

You can't restore a server that has been deleted. When you delete a server, you also delete the backups associated with it.

Point-in-time restore

A point-in-time restore creates a new server using the backups from your original server, and rolls the server forward to the specified time. You initiate a restore operation using the **Restore** command in the toolbar on the **Overview** page for your server in the Azure portal. You'll be prompted for the name of a new server, and a point in time.

The screenshot shows the "Restore" dialog box in the Azure portal. It contains the following information:

- Restore point (UTC)**: 08/27/2019 at 1:43:30 PM.
- Restore to new server**: A text input field containing "restoredserver" with a green checkmark.
- Location**: A dropdown menu set to "UK South".
- Pricing tier**: "Basic, 2 vCore(s), 50 GB" with a lock icon.

The Azure CLI supports the `az mysql/postgres server restore` commands if you prefer to perform restore operations from the command line. For example:

```
az mysql server restore \
  --resource-group [resource group name] \
  --name [new Azure Database for MySQL server name] \
  --source-server [original Azure Database for MySQL server name] \
  --restore-point-in-time "2019-10-23T02:10:00+08:00"
```

Geo-restore

A geo-restore is a complete restoration of a server, using a backup held in geo-redundant storage. When you create a new server using the Azure portal, you specify a geo-redundant backup as the data source. When the new server is created, it will be populated with the databases in this backup.

The screenshot shows the 'Server details' configuration page in the Azure portal. It contains the following fields and options:

- Server name:** A text input field containing 'geo-restored-server' with a green checkmark on the right.
- Data source:** A radio button group with 'None' and 'Backup' (selected).
- Backup:** A dropdown menu with the text 'Select a backup' and a search icon.
- Location:** A dropdown menu with the text '(Europe) UK South'.

Below the 'Location' field, there is a link labeled 'Supported Locations'.

The Azure CLI provides the `az mysql/postgres server georestore` commands to do a geo-restore from the command line.

It might take up to an hour to replicate a geo-redundant backup to another region. This could result in the loss of up to an hour's worth of data if you need to do a geo-restore from a different region.

Lesson 1: Summary

In this lesson, you saw how to:

- Manage security in Azure Database for MySQL/PostgreSQL.
- Back up and restore a server.

Knowledge Check

Multiple choice

You're writing a script that uses the Azure CLI to create and configure an Azure Database for MySQL service. You need to allow access to the service for other Azure services.

Which IP address range should you specify in the `az mysql server firewall-rule create` command?

- You don't need to create a firewall rule because all Azure services will automatically have access to the server.
- Set the `start-ip-address` parameter to 0.0.0.0, and the `end-ip-address` parameter to 255.255.255.255.
- Set the `start-ip-address` and `end-ip-address` parameters to 0.0.0.0.
- Set the `start-ip-address` and `end-ip-address` parameters to 255.255.255.255.
- Set the `start-ip-address` and `end-ip-address` parameters to the range defined by the subnet that contains your resources.

Multiple choice

Restoring an Azure Database for PostgreSQL server from a backup overwrites the existing data.

True or false?

- True.
- False.

Monitoring and tuning

Lesson 2: Monitoring and tuning

When you've migrated your database to Azure Database for MySQL or Azure Database for PostgreSQL, you should continue to monitor the system. There might be areas you can optimize for better performance. In this lesson, you'll see how to use Azure Monitor to visualize the performance of your databases and servers. You'll learn how to modify the configuration of a server, and how to track the activity being performed by applications and users running against your database.

Learning objectives

By the end of this lesson, you'll understand how to:

- Monitor database and server performance in Azure.
- Configure server parameters.
- Trace activity in a database.
- Track query performance.
- Configure read replicas.

Monitor database and server performance in Azure

Use Azure Monitor to track resource use in Azure Database for MySQL/PostgreSQL. The **Metrics** page for your server in the Azure portal enables you to create charts that help to detect trends in performance, and spot anomalies.

Metrics for Azure Database for MySQL/PostgreSQL

The metrics available for monitoring a server fall into four broad categories:

- Storage metrics
- Connection metrics
- Data processing resource utilization metrics
- Replication metrics

Storage metrics

Storage metrics track the total size of your databases across the server (*Storage used*), and the current amount of storage space allocated to the server (*Storage limit*). In an active system, you'll likely find that the *Storage used* metric grows. If you have the auto-growth option selected for the server, the *Storage limit* metric occasionally increases as the amount of free space diminishes. Additional storage is allocated whenever the amount of free space drops below 5 percent of the current usage. Use the *storage percent* metric to view the proportion of used space to free space on your server. If your server is regularly spending time allocating extra storage, consider assigning more space manually. Do this in the Azure portal—select the **Pricing tier** page for your server, and use the **Storage** slider. Note that you're charged for storage, so don't set the storage available to an unnecessarily large amount.

The *Backup Storage used* metric shows how much space your backups are taking. This metric is important from a cost perspective. You aren't charged for backup storage providing it remains below the size of the storage space allocated to your server (as specified by the pricing tier). When you go above this limit, you incur charges for backup storage.

Connection metrics

The *Active connections* metric shows how many concurrent connections the server is currently supporting. This might not be the same as the number of concurrent users, depending on whether you've configured any type of connection pooling. Azure Database for MySQL/PostgreSQL doesn't currently provide any connection pooling capability, but you could use a proxy service such as *PgBouncer* (for PostgreSQL) to implement this feature. For more information, see **Performance best practices for using Azure Database for PostgreSQL – Connection Pooling**¹

The *Failed connections* metric shows how often users have presented invalid credentials. A large number of these events over a short period of time might indicate a brute-force attack.

Data processing resource utilization metrics

These metrics help you to monitor how your server handles the workload.

The *CPU percent* metric shows how busy the CPU is. A high CPU utilization isn't a problem, unless it has been increasing over time. CPU utilization that's over 90% and still rising indicates that your system is approaching processing capacity. You should consider scaling up to a pricing tier that has more resources.

The *Memory percent* metric indicates memory occupancy. Azure Database for MySQL/PostgreSQL uses memory for caching data, and for running the processes initiated by each client request. High memory utilization isn't a problem until it becomes excessive—typically over 95%, depending on the actual amount of memory available. Very low memory availability can cause connection failures, and slow performance because of memory fragmentation. You should monitor this metric to determine whether memory occupancy is growing over time, and scale your server accordingly.

The *IO percent* metric tracks the amount of disk activity being performed by the server. Ideally, this value should be as low as possible. Disk IO is a slow operation. A high value for this metric, in conjunction with a high value for *Memory percent*, might indicate that the server has insufficient resources to cache data effectively, and is instead having to read and write data to disk storage. A degree of IO activity is inevitable, because your data must be persisted to disk at some point, and transaction logs must be maintained. In most database servers, this writing is performed by a separate process or thread that runs asynchronously.

The *Network In* and *Network Out* metrics show the volume of traffic entering and exiting the server across active connections. The limits to these figures are determined by the bandwidth of the path between the client applications and the server.

Replication metrics

Azure Database for PostgreSQL provides the *Max Lag Across Replicas*, and *Replica Lag* metrics to help you determine how up-to-date any replicas are. These metrics are only meaningful if you've configured read-only replicas.

¹ <https://azure.microsoft.com/blog/performance-best-practices-for-using-azure-database-for-postgresql-connection-pooling/>

The *Max Lag Across Replicas* metric shows how many bytes the slowest replica is behind the master. You can only monitor this metric from the master.

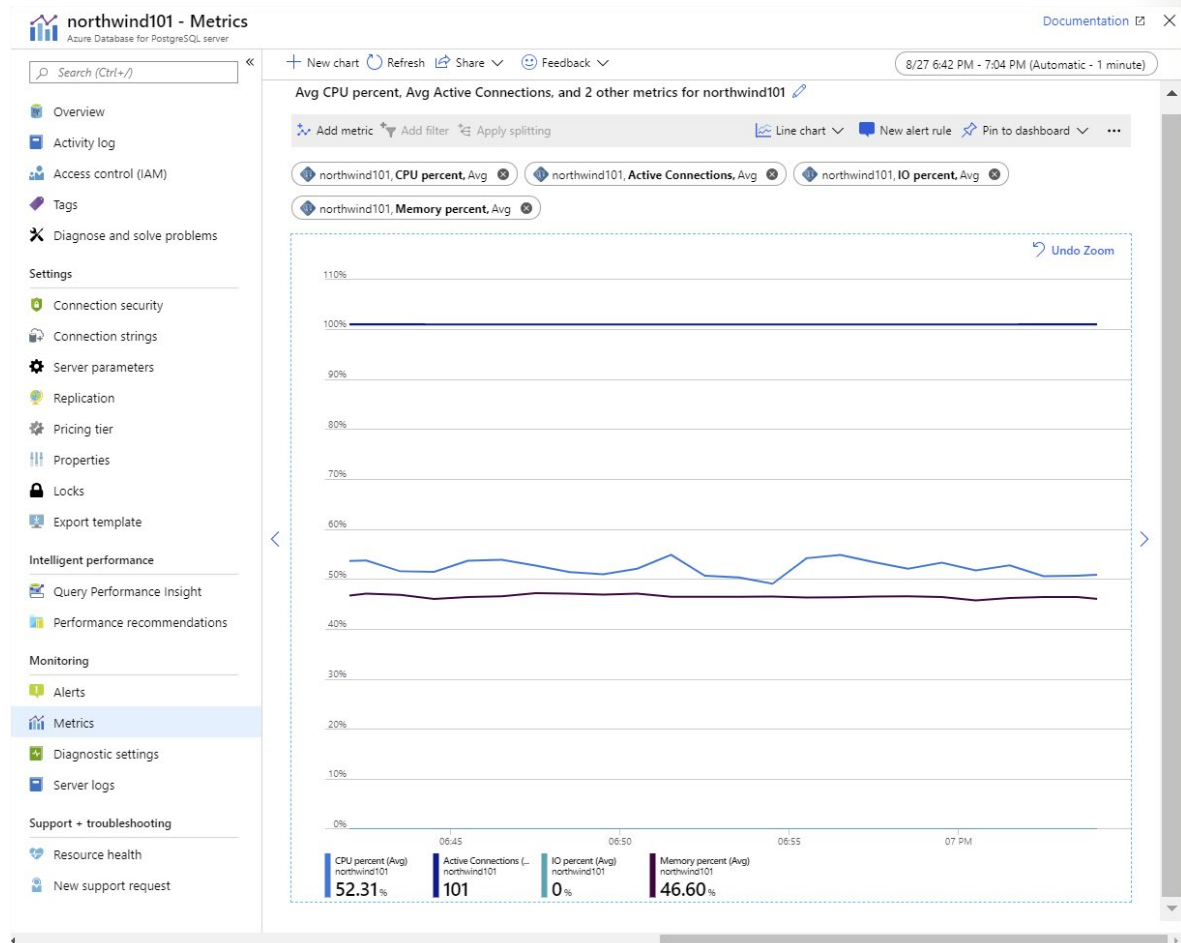
The *Replica Lag* metric shows the time, in seconds, since the latest transaction was received from the master and applied to a replica. This metric only makes sense when viewed on a replica.

Azure Database for MySQL has the *Replication lag in seconds* metric. This metric, which you can only monitor from a replica, shows the number of seconds by which the replica is lagging behind the master.

Creating charts and alerts to monitor performance

The **Metrics** page for a server in the Azure portal enables you to create charts that track metric values. Metrics are gathered at one-minute intervals. For each metric, you specify an aggregation that determines how to report that metric. For example, *Average* will generate an average value for the metric in each minute; *Max* will show the maximum value achieved during each minute; *Min* will show the smallest value; *Sum* totals the metric; and *Count* shows how many times the event generating the metric occurred. Not all aggregations are necessarily meaningful for every metric.

The example chart below captured the average minute-by-minute values for the CPU percent, Memory percent, IO percent, and Active connections metrics. You'll see that there are 101 active connections all running concurrently. The CPU and memory utilization are both stable, and the IO percent is at 0. In this example, the client applications are performing read-intensive workloads and the necessary data is cached in memory.



Note that there's a lag of up to five minutes between the metrics being captured and the results displayed on a chart.

If a metric indicates that a resource is reaching a critical point, you can set an alert to notify an administrator. The example below sends an email to an administrator if the memory utilization exceeds 90 percent.

*** RESOURCE** **HIERARCHY**

northwind101 Contoso > northwindrg

*** CONDITION** **Monthly cost in USD (Estimated)** ⓘ

Whenever the Memory percent is Greater than 90 % \$ 0.10

Total \$ 0.10

ACTIONS

ACTION GROUP NAME **CONTAIN ACTIONS**

Administrators 1 Email

Action rules (preview) allows you to define actions at scale as well as suppress actions. Learn more about this functionality [here](#)

ALERT DETAILS

* Alert rule name ⓘ

Description

* Severity ⓘ

Configure server parameters

Native MySQL and PostgreSQL servers are highly configurable—both use configuration settings stored in parameter files. For PostgreSQL, this information is held in the *postgresql.conf* file, and for MySQL, configuration data is stored in various *my.cnf* files. In Azure Database for MySQL/PostgreSQL, you don't have direct access to these files. Instead, you view and modify server parameters by using the Azure portal or the Azure CLI.

View and set parameters using the Azure portal

The server configuration parameters are available on the **Server parameters** page for your server in the Azure portal. You can modify the parameter values as appropriate for your server. The image below shows the server parameters page for Azure Database for PostgreSQL. The corresponding page for Azure Database for MySQL is similar.

Parameter Name	Value	Type	Description
pg_catalog.config_name	dynamic	Dynamic	Set the maximum query length
pg_qs.query_capture_mode	NONE	Dynamic	Selects which statements are tracked b
pg_qs.replace_parameter_placeholders	ON	Dynamic	Selects whether parameter placeholder
pg_qs.retention_period_in_days	7	Dynamic	Sets the retention period window in da
pg_qs.track_utility	ON	Dynamic	Selects whether utility commands are t
pg_stat_statements.track	TOP	Dynamic	Controls which statements are counted
pgms_wait_sampling.history_period	100	Dynamic	Set the frequency, in milliseconds, at w
pgms_wait_sampling.query_capture_mode	NONE	Dynamic	Selects which statements are tracked b
postgis.gdal_enabled_drivers	DISABLE_ALL	Dynamic	Controls postgis GDAL enabled driver :
quote_all_identifiers	ON	Dynamic	When generating SQL fragments, quot
random_page_cost	4.0	Dynamic	Sets the planner's estimate of the cost
search_path	"User", public	Dynamic	Sets the schema search order for name
seq_page_cost	1.0	Dynamic	Sets the planner's estimate of the cost
shared_preload_libraries	0 selected	Static	Sets which shared libraries are preload
statement_timeout	0	Dynamic	Sets the maximum allowed duration (ir
synchronize_seqscans	ON	Dynamic	Enable synchronized sequential scans.
synchronous_commit	ON	Dynamic	Sets the current transaction's synchron
temp_buffers	8192	Dynamic	Sets the maximum number of tempora
timezone	UTC	Dynamic	Sets the time zone for displaying and i
transform_null_equals	ON	Dynamic	Treats "expr=NULL" as "expr IS NULL".
vacuum_defer_cleanup_age	0	Dynamic	Number of transactions by which VACL
work_mem	4096	Dynamic	Sets the amount of memory to be used
xmlbinary	BASE64	Dynamic	Sets how binary values are to be encod
xmloption	CONTENT	Dynamic	Sets whether XML data in implicit pars

Not all server configuration parameters are available because a large part of the server configuration is controlled by Azure. For example, parameters associated with memory allocation are missing. Additionally, Azure Database for MySQL doesn't support ISAM storage, so the *mysiam* parameters aren't there.

Changes to parameters that are marked as *Dynamic* come into effect immediately. Parameters marked as *Static* require you to restart the server. You do this on the **Overview** page for your server.

Overview page for 'northwind101' (Azure Database for PostgreSQL server). The page shows server details and management options. The 'Restart' button is highlighted with a red box.

Property	Value
Resource group (change)	northwind101
Status	Available
Location	UK South
Server name	northwind101.postgres.database.azure.com
Admin username	northwindadmin@northwind101
PostgreSQL version	10

View and set parameters using the Azure CLI

You can view and modify parameters programmatically with the `az mysql/postgres server configuration` commands. View the settings of every configuration parameter with `az mysql/postgres server configuration list`, and home in on a single parameter using `az mysql/postgres server configuration show [parameter-name]`. The code snippet below shows an example for Azure Database for PostgreSQL:

```
az postgres server configuration show \
  --resource-group northwindrg \
  --server-name northwind101 \
  --name vacuum_defer_cleanup_age
```

The result should look similar to this:

```
{
  "allowedValues": "0-1000000",
  "dataType": "Integer",
  "defaultValue": "0",
  "description": "Number of transactions by which VACUUM and HOT cleanup should be deferred, if any.",
  "id": "*****",
  "name": "vacuum_defer_cleanup_age",
  "resourceGroup": "northwindrg",
  "source": "system-default",
  "type": "Microsoft.DBforPostgreSQL/servers/configurations",
  "value": "0"
}
```

The important item in the output is the **value** field, which shows the current setting for the parameter.

Use the `az mysql/postgres server configuration set` command to change the value of a configuration parameter, as follows:

```
az postgres server configuration set \
  --resource-group northwindrg \
  --server-name northwind101 \
  --name vacuum_defer_cleanup_age \
  --value 5
```

If you need to restart a server after changing a static parameter, run the `az mysql/postgres server restart` command:

```
az postgres server restart \
  --resource-group northwindrg \
  --name northwind101
```

Trace activity

Azure Database for MySQL/PostgreSQL also records diagnostic information in the server logs. Server logs are the native message log files for MySQL and PostgreSQL (not the transaction log files, which are inaccessible in Azure Database for MySQL/PostgreSQL). These files contain messages, server status, and

other error information that you use to debug problems with your databases. The server logs are retained for up to seven days (less, if the total size of the server log files exceeds 7 GB).

Azure Database for MySQL and Azure Database for PostgreSQL record different details in the server logs. The following sections describe the server logs for each service separately.

Server logs in Azure Database for MySQL

In Azure Database for MySQL, the server log provides the information normally available in the *slow query log* and the *audit log* on a MySQL server.

You use the information in the slow query log to help identify slow-running queries. By default, the slow query log is disabled. You enable it by setting the **slow_query_log** server parameter to **ON**. You configure the slow query log to determine what is meant by a *slow query* using the following server parameters:

- **log_queries_not_using_indexes**. This parameter is either **ON** or **OFF**. Set it to **ON** to record all queries that are likely to perform a full table scan rather than an index lookup.
- **log_throttle_queries_not_using_indexes**. Specifies the maximum number of slow queries not using indexes that can be logged per minute.
- **log_slow_admin_queries**. Set this parameter to **ON** to include slow running administrative queries in the log.
- **long_query_time**. The threshold (in seconds) for a query to be considered *slow running*.

After you've enabled the slow query log and the audit log, the log files will start to appear in the **Server logs** page for the server. A new slow query log is created each day. Click a log file to download it:

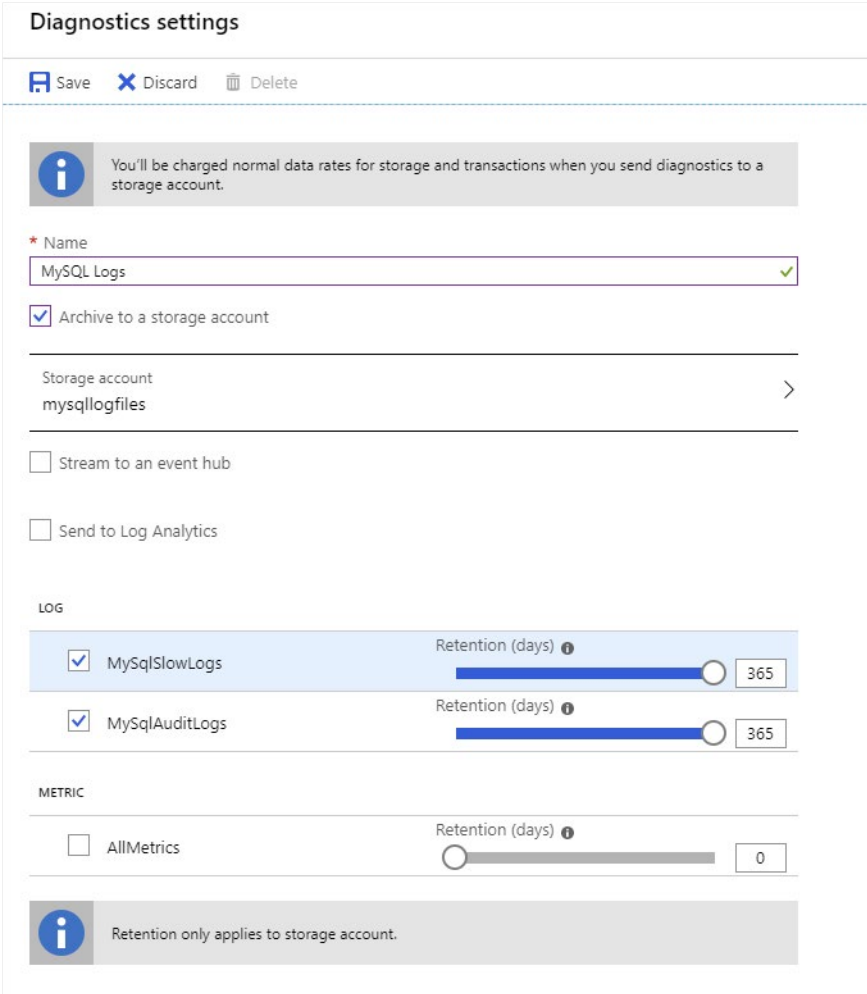
The screenshot shows the 'Server logs' page for a MySQL server named 'northwindmysql'. The page includes a search bar, a list of log files, and a sidebar with navigation options. A message at the top states: 'Server logs are created every 24 hours. You will be able to access each log for up to 7 days after creation.'

NAME	LAST UPDATE TIME	SIZE
mysql-slow-northwindmysql-2019082716.log	Wed, 28 Aug 2019 12:03:40 GMT	9KB

To enable audit logging, set the **audit_log_enabled** server parameter to **ON**. You configure audit logging with the following parameters:

- **audit_log_events**. Specify the events to be audited. In the Azure portal, this parameter provides a drop-down list of events, such as **CONNECTION**, **DDL**, **DML**, **ADMIN**, and others.
- **audit_log_exclude_users**. This parameter is a comma-separated list of users whose activities won't be included in the audit log.

If you need to preserve the slow query log and audit log for more than seven days, you arrange for them to be transferred to Azure storage. Use the **Diagnostics settings** page for your server, and then select **+ Add diagnostic setting**. On the **Diagnostics settings** page, select **Archive to a storage account**, select a storage account in which to save the log files (this storage account must already exist), select **MySQLSlowLogs** and **MySqlAuditLogs**, and specify a retention period of up to 365 days. You can download the log files from Azure storage at any point during this time. Select **Save**:



Diagnostics settings

Save Discard Delete

i You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.

* Name
MySQL Logs ✓

Archive to a storage account

Storage account
mysqllogfile >

Stream to an event hub

Send to Log Analytics

LOG

MySQLSlowLogs Retention (days) 365

MySqlAuditLogs Retention (days) 365

METRIC

AllMetrics Retention (days) 0

i Retention only applies to storage account.

Slow query log data will be written in JSON format to blobs in a container named **insights-logs-mysqlslowlogs**. It can take up to 10 minutes for the log files to appear in Azure storage. Audit records are stored in the **insights-logs-mysqlslowlogs** blob container, again in JSON format.

Server logs in Azure Database for PostgreSQL

In Azure Database for PostgreSQL, the server log contains error log and query log files. Use the information in these files to help locate the sources of any errors and inefficient queries.

You enable logging by setting the various **log_** server configuration parameters to **ON**. These parameters include:

- **log_checkpoints**. A checkpoint occurs whenever every data file has been updated with the latest information from the transaction log. If there's a server failure, this point marks the time at which recovery needs to commence by rolling forward from the transaction log.
- **log_connection** and **log_disconnections**. These settings record each successful connection, and the end of each session.
- **log_duration**. This setting causes the duration of each completed SQL statement to be recorded.
- **log_lock_waits**. This setting causes lock wait events to be recorded. Lock waits can be caused by poorly implemented transactions in application code.
- **log_statement_stats**. This setting writes cumulative information about the performance of the server to the log.

Azure Database for PostgreSQL also provides further parameters to fine-tune the information that's recorded:

- **log_error_verbosity**. This setting specifies the level of detail recorded for each logged message.
- **log_retention_days**. This is the number of days that the server retains each log file before removing it. The default is three days, and you can set it to a maximum of seven days.
- **log_min_messages** and **log_min_error_statement**. Use these parameters to specify the warning and error levels for recording statements.

As with Azure Database for MySQL, the log files generated by Azure Database for PostgreSQL are available on the **Server logs** page. You can also use the **Diagnostic settings** page to copy the logs to Azure storage.

Track query performance

Query Store is an additional feature provided by Azure to help you identify and track poorly running queries. You use it with Azure Database for MySQL and Azure Database for PostgreSQL.

Enabling query performance tracking

Query Store records information in the **mysql** schema in Azure Database for MySQL, and in a database named **azure_sys** in Azure Database for PostgreSQL. Query Store can capture two types of information—data about query execution, and information on wait statistics. Query Store is disabled by default. To enable it:

- If you're using Azure Database for MySQL, set the server parameters **query_store_capture_mode** and **query_store_wait_sampling_capture_mode** to **ALL**.
- If you're using Azure Database for PostgreSQL, set the server parameter **pg_qs.query_capture_mode** to **ALL** or **TOP**, and set the **pgms_wait_sampling.query_capture_mode** parameter to **ALL**.

Analyzing query performance data

You can query the tables used by Query Store directly. If you're running Azure Database for MySQL, connect to your server, and run the following queries:

```
SELECT * FROM mysql.query_store;
```

```
SELECT * FROM mysql.query_store_wait_stats;
```

If you're using Azure Database for PostgreSQL, run the following queries instead:

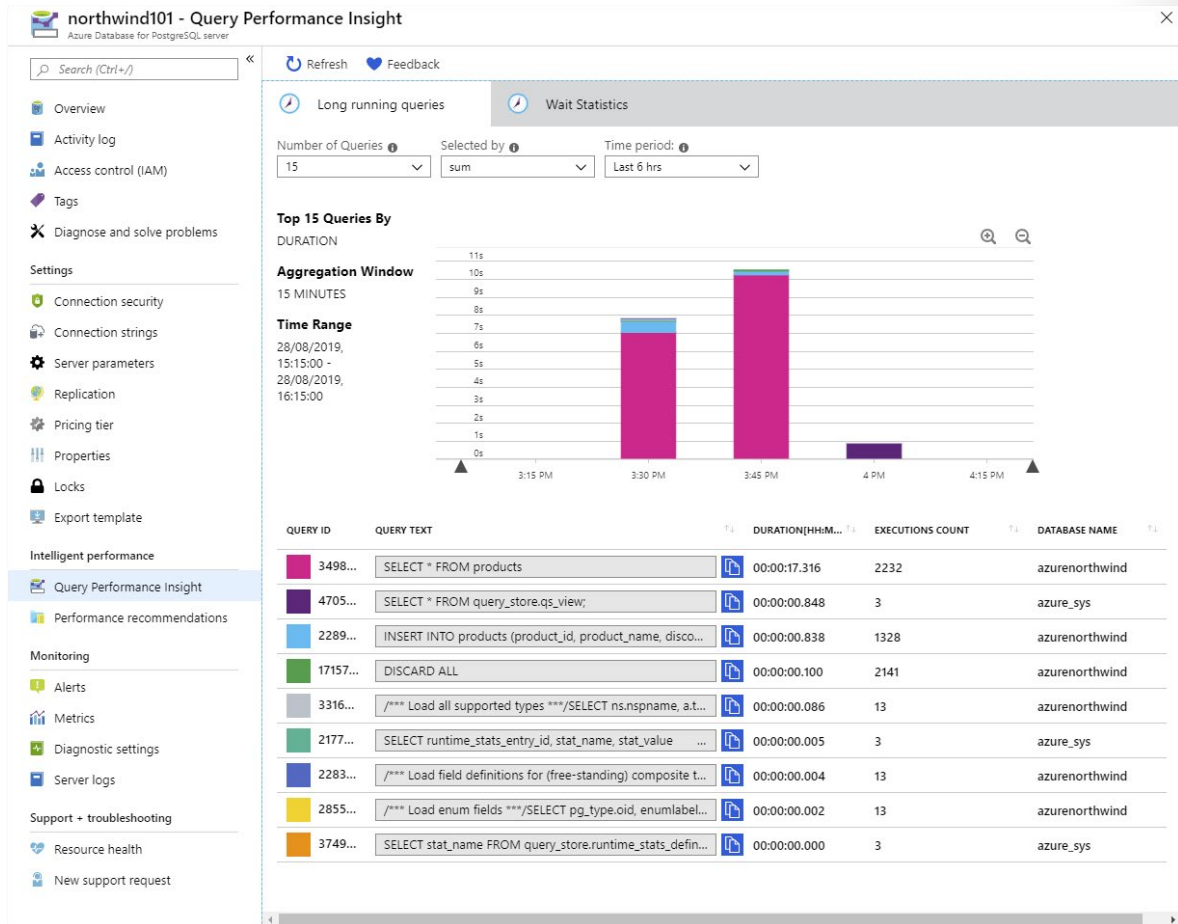
```
SELECT * FROM query_store.qs_view;
```

```
SELECT * FROM query_store.pgms_wait_sampling_view;
```

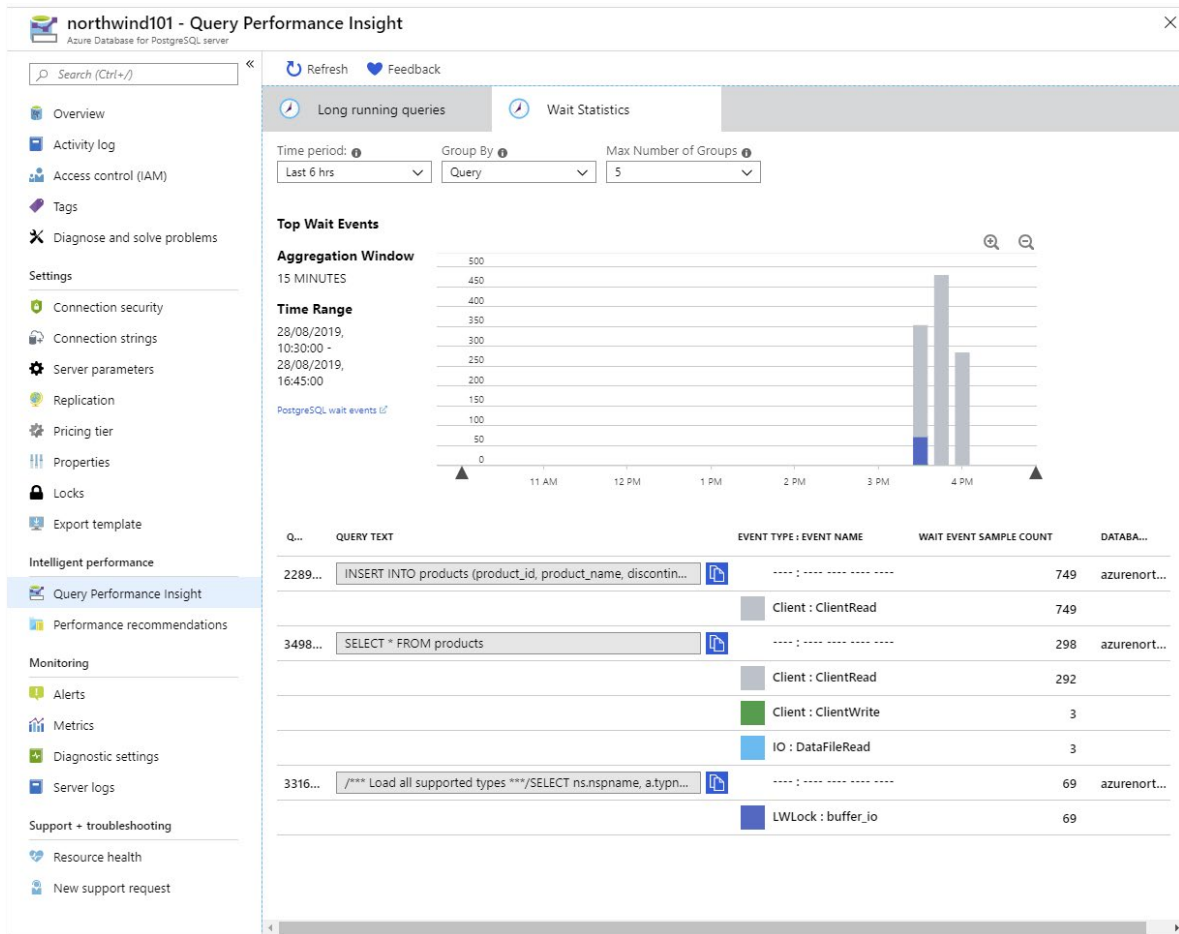
In both cases, the first query will display the text for each recently run query, and a host of statistics about how long the query took to compile and execute. The second query displays information about wait events. A wait event occurs when one query is prevented from running because it requires the resources held by another.

If you examine the Query Store directly, you can generate your own custom reports and gain a detailed insight into how the system is functioning. However, the amount of data available can make it difficult to understand what's happening. Azure Database for MySQL/PostgreSQL provides two additional tools to help you navigate this data—**Query Performance Insight**, and **Query Recommendations**.

Query Performance Insight is a graphical utility, available from the **Query Performance Insight** page for your server. The **Long running queries** tab displays the statistics for the most long running queries. You specify the time period, and zoom in to within a few minutes. The legend shows the text of each query, together with the duration and number of times the query was run. The graph gives a comparative view of the duration of each query. You view the data by the average time for each query, but it's also instructive to display the total time (*duration * execution count*) for each query. The image below shows an example:



The **Wait Statistics** tab shows the wait event information for each query. You'll see the amount of time spent by a query waiting for various resources.



Wait events typically fall into three categories:

- **Lock waits.** These events occur if a query is attempting to read or modify data that's locked by another query. If you experience a large number of lock waits, check for long running transactions, or operations that use a highly restrictive isolation level.
- **IO waits.** This type of wait occurs if a query is performing a significant amount of IO. This could be due to a poorly designed query (check the *WHERE* clause), an inefficient join operation, or a full table scan incurred because of a missing index.
- **Memory waits.** A memory wait occurs if there's insufficient memory available to process a query. Your query could be attempting to read a large amount of data, or it might be blocked by other queries hogging memory. Again, this might indicate that indexes are missing, causing queries to read entire tables into memory.

It's also highly likely that one form of wait triggers another, so you can't necessarily examine these issues in isolation. For example, a transaction that reads and updates data in different tables might be subject to a memory wait. In turn, this transaction could have locked data that causes another transaction to incur a lock wait.

The **Performance Recommendations** page for the server takes the information held in Query Store and uses it to make recommendations for tuning your database for the workloads it's experiencing.

Configure read replicas

You use read replication to copy data from one instance of Azure Database for MySQL/PostgreSQL (referred to as the *master*) to up to five replicas. Use replication to spread the load across servers for read-heavy workloads. Replication is one-way only, and each replica is read-only. Replication operates asynchronously, so there's a lag between the time the data changes on the master and the point at which it appears in each replica.

Replicas can be in different regions from the master. You use replicas to place data close to the clients needing it, to reduce query latency. Cross-region replication also gives you a mechanism for handling regional disaster recovery.

[NOTE]

Cross-region replication is not available in the Basic performance tier.

Each replica is an instance of Azure Database for MySQL/PostgreSQL in its own right, but configured as read-only. If the connection to the master server is lost, or the master server is deleted, each replica becomes an independent read-write server. In this case, replicas are no longer synchronized with each other, so the data they hold might start to diverge.

[NOTE]

If you're using Azure Database for MySQL, read replicas are only available in the General Purpose and Memory Optimized pricing tiers. Additionally, read replicas aren't available in Azure Database for PostgreSQL, Hyperscale (Citus).

Creating replicas

The simplest way to add replicas to a server is through the **Replication** page for the server in the Azure portal. On this page, select **+ Add Replica**.

northwind101 - Replication
Azure Database for PostgreSQL server

Search (Ctrl+/) << Disable replication support **+ Add Replica** Delete Replica Stop Replication

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems

Settings
Connection security
Connection strings
Server parameters
Replication
Pricing tier
Properties
Locks

Master

NAME	PRICING TIER	LOCATION	STATUS
mpostgresql	Memory optimized, 2 vCore(s), 10...	UK South	Available

Replicas

NAME	PRICING TIER	LOCATION	STATUS
No results			

You'll be prompted for a name and location for the server. Apart from that, the other details for the replica, including the pricing tier, are set to the same as those used by the master. When the replica has been created, you can amend any settings for that server, including adjusting the pricing tier. However, make sure that each replica has sufficient resources available to handle the workload associated with receiving and storing the replicated data.

[NOTE]

If you're using the General Purpose or Memory Optimized pricing tiers, you must also enable replication support. You do this on the **Replication** page by selecting **Enable replication support**. The server will be restarted before you can continue.

When you've added a replica, it'll be shown on the **Replication** page. Depending on the size of the master and the amount of data in the databases, deployment and synchronization of each replica might take a significant amount of time.

Master			
NAME	PRICING TIER	LOCATION	STATUS
northwind101	Basic, 2 vCore(s), 50 GB	UK South	Available
Replicas			
NAME	PRICING TIER	LOCATION	STATUS
northwindreplica2	Basic, 2 vCore(s), 50 GB	UK South	Deploying
northwindreplica1	Basic, 2 vCore(s), 50 GB	UK South	Deploying

You reconfigure and resize a replica by selecting it on the **Replication** page.

If you prefer to use the Azure CLI, create replicas with the `az mysql/postgres server replica create` command:

```
az postgres server replica create \
  --name northwindreplica3 \
  --resource-group northwindrg \
  --source-server northwind101
```

Removing a replica

To remove a replica, select the replica on the **Replication** page, and select **Stop Replication**. The replica server will detach from the master and be converted into a read-write server instead. The replica won't be deleted, and you'll continue to be charged for the resources it consumes. If you need to delete the replica, use the **Delete Replica** command instead.

The Azure CLI provides the `az mysql/postgres server replica stop` command to halt replication and convert a replica into a read-write server. You then use the `az mysql/postgres server delete` command to delete the replica and free its resources.

Demonstration: Monitoring the sample database

In this demonstration, you will see how to use the features available in the Azure portal to monitor an Azure Database for PostgreSQL server. This demo will run an application that simulates a number of users performing concurrent operations against the Northwind database hosted by the server. You will create a chart that shows key performance metrics, and use Query Store to see the queries being run.

The main tasks in this demonstration are:

1. Run a workload and capture metrics
2. Examine queries using Query Store

Setup

This demonstration assumes that you have completed the demonstration from module 3; it uses the Azure Database for PostgreSQL server and database created by that demonstration. If you haven't completed that demonstration, perform the following setup steps:

1. Open an Azure Cloud Shell window. Make sure that you are running the **Bash** shell.
2. In the Cloud shell, clone the repository that holds 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-workshop
```

3. Move to the `workshop/migration_samples/setup/postgresql` folder.

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

4. 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_northwind.sh northwind[nnn] [resource_group_name] [location]
```

5. When prompted for the password for the `northwindadmin` user, type **Pa55w.rdDemo**.
6. Wait for the script to complete before continuing. It will finish with the message **Setup Complete**.

Task 1: Run a workload and capture metrics

1. In the Azure portal, go to the page for your Azure Database for PostgreSQL server.
2. Under **Settings**, click **Connections strings**.
3. On the **Connection strings** page, copy the **ADO.NET (Ngpsql)** connection string to the clipboard.
4. Start the Azure Cloud shell.
5. If you haven't already done so, run the following command to clone the repository containing the scripts and sample databases.

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

6. Move to the *workshop/migration_samples/code/postgresql/NorthwindSoakTest* folder.

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

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

```
code App.config
```

8. Replace the value of the **ConnectionString** app setting with the connection string from the clipboard. In this string, replace the text **{your_database}** with **azurenorthwind**, and replace **{your_password}** with **Pa55w.rdDemo**.
9. Save the changes and close the editor.
10. Run the app:

```
dotnet run
```

The application simulates 101 concurrent users performing a series of queries against the Northwind database. Allow the app to continue running as you perform the next steps.

11. Return to the page for your server in the Azure portal.

12. Under **Monitoring**, click **Metrics**.

13. On the chart page, add the following metric:

Property	Value
Resource	northwind[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

14. Click **Add metric**, and add the following metric:

Property	Value
Resource	northwind[nnn]
Metric Namespace	PostgreSQL server standard metrics
Metric	CPU percent

Property	Value
Aggregation	Avg

15. Click **Add metric**, and add the following metric:

Property	Value
Resource	northwind[nnn]
Metric Namespace	PostgreSQL server standard metrics
Metric	Memory percent
Aggregation	Avg

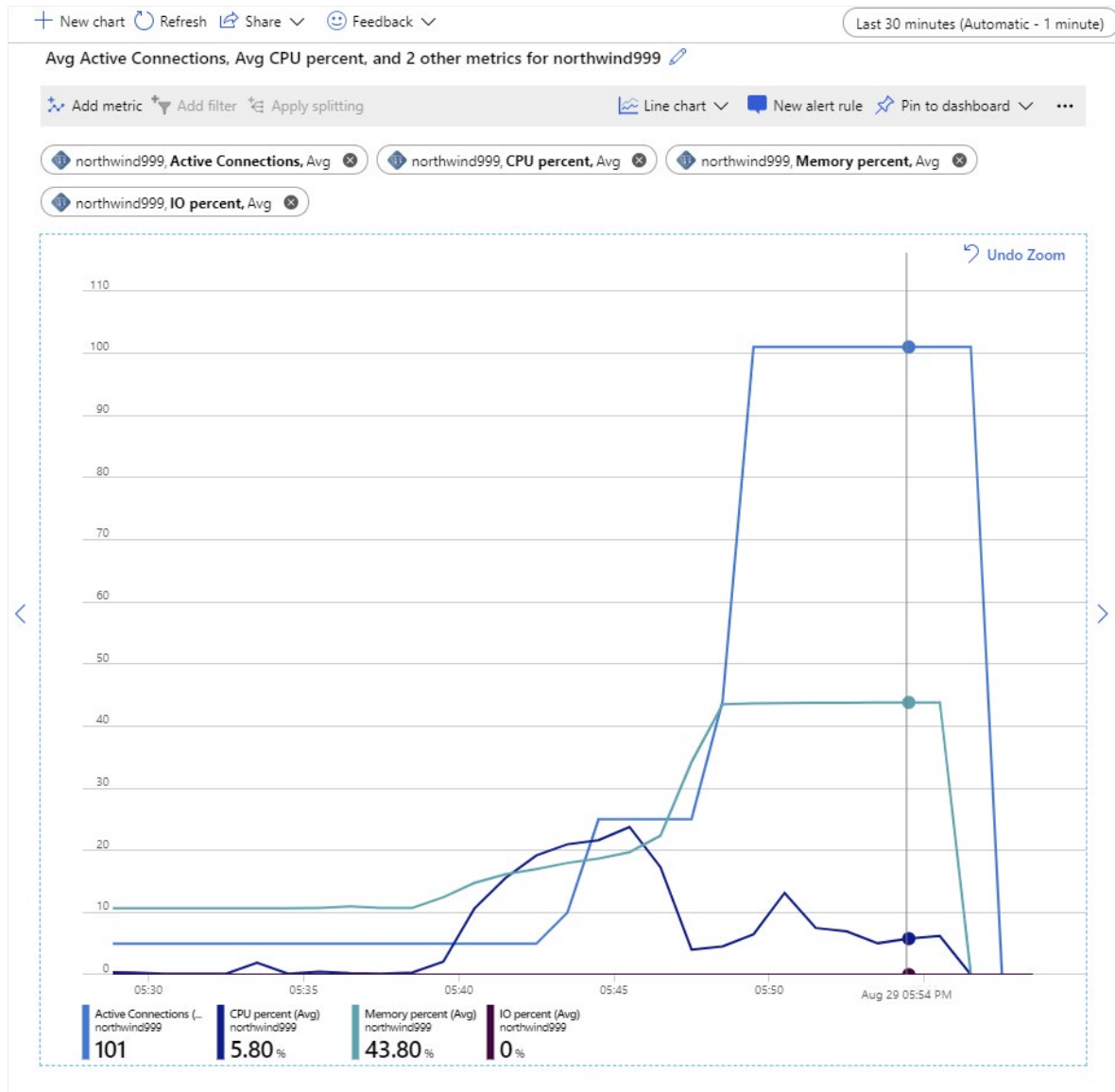
16. Click **Add metric**, and add the following metric:

Property	Value
Resource	northwind[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.

17. Set the time range for the chart to **Last 30 minutes**.

Allow the application to continue running for a few minutes; there will be a lag before the metrics start to reflect the load placed on the system. After a short time, you should see that the number of active connections levels off at 101; this is the number of users simulated by the test app. You should also notice that the Memory percent metric rises and then levels out at around 45%, and the IO percent remains close to 0. All the connections are querying the same data, which is being cached in memory. Once the data has been read in, each query incurs very little disk IO.



Task 2: Examine queries using Query Store

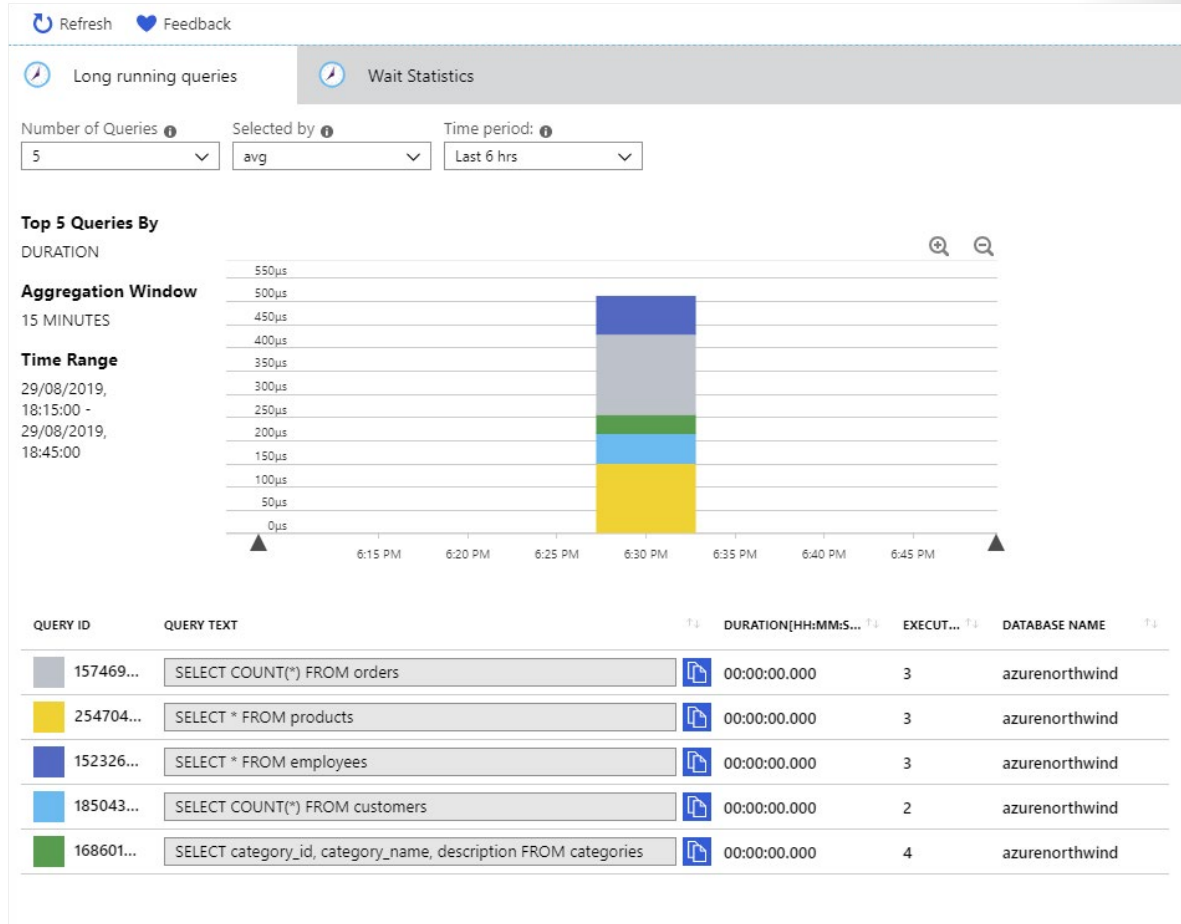
1. In the Azure portal, on the page for your server, under **Settings**, click **Server parameters**, and set the following server parameters to the values shown in the table, and then click **Save**

Parameter	Value
pg_qs.query_capture_mode	ALL
pgms_wait_sampling.query_capture_mode	ALL

Setting these parameters captures the performance data held by Query Store

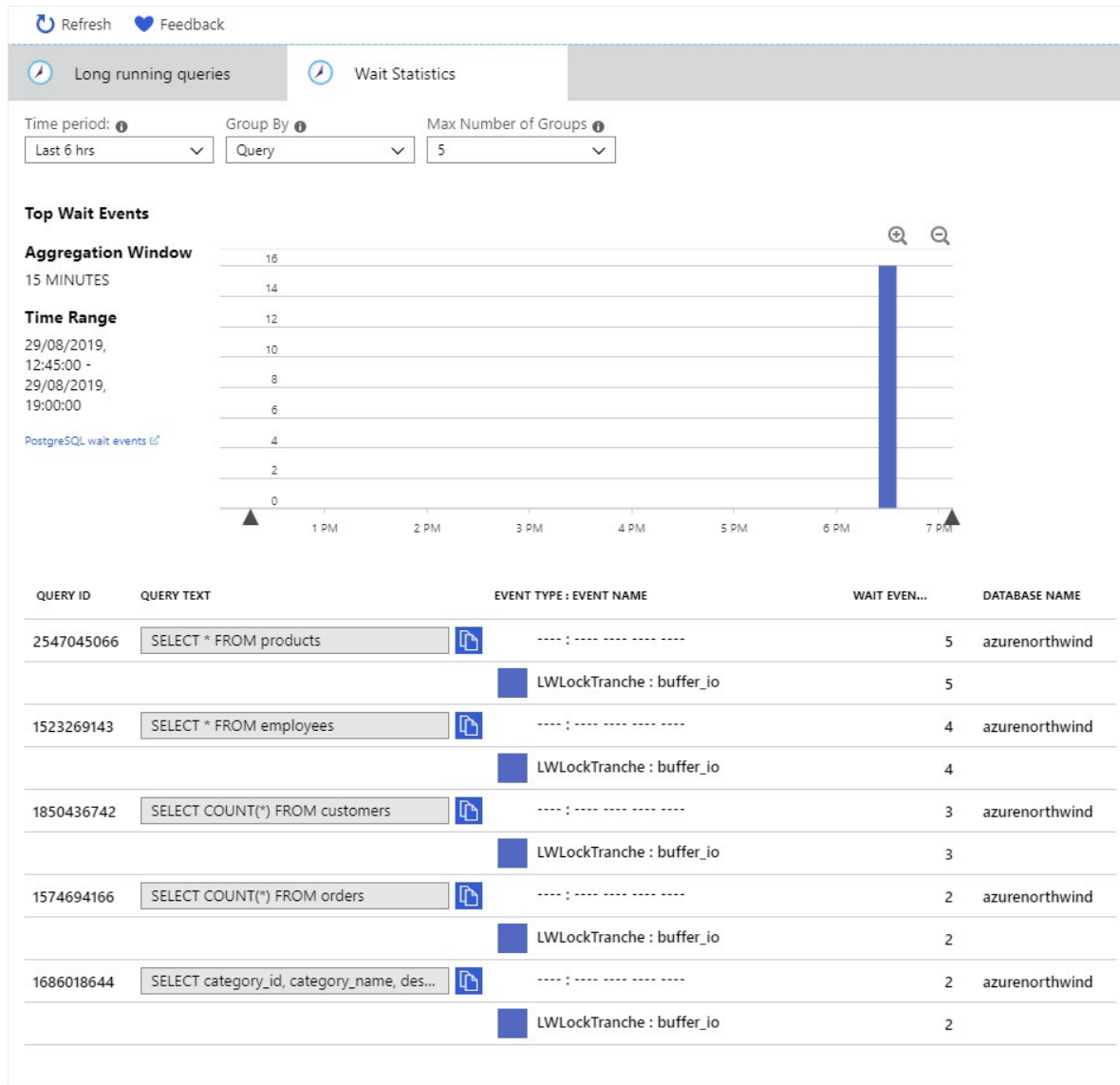
2. Allow the test app to continue running for a couple more minutes.
3. In the Azure portal, under **Intelligent performance**, click **Query Performance Insight**.
4. On the **Long running queries** tab, set the **Time period** to **Last 6 hrs**, and then zoom in on the current burst of activity shown in the chart.

- Examine the queries below the chart. These are the queries performed by the test application. The color associated with each query corresponds to the sections in the bar in the chart. The chart gives you a good comparison of the relative time spent executing each query.
- Click each section of the bar in the chart. The cumulative times for the corresponding query will appear.



- Click the **Wait Statistics** tab.
- Set the **Time period** to **Last 6 hrs**, and set **Group By** to **Query**.

The chart should show that only one type of wait event occurred; **LWLockTranche : buffer_io**, and each event has only occurred a small number of times. This type of wait event occurs when a query is waiting for IO on a data page. In this case, the IO occurs the first time each page is read from disk; it is then cached in memory. The **memory percent** metrics shown in the previous task indicate that there is sufficient memory available to cache all the data being read by the application. If memory was starting to run low, you would likely see far more of these wait events.



9. Return to the Cloud Shell, and press Enter to stop the test application.

Lesson 2: Summary

In this lesson, you saw how to:

- Monitor database and server performance in Azure.
- Configure server parameters.
- Trace activity in a database.
- Track query performance.
- Configure read replicas.

Knowledge Check

Multiple choice

How do you change the server parameters for Azure Database for MySQL?

- Edit the my.cnf for the server using a text editor, then restart the server.
- Modify the server log settings for the server in the Azure portal.
- Use the Server parameters page for the server in the Azure portal.
- You can't change server parameters. The server is managed entirely by Azure.
- Connect to the server using MySQL Workbench and use the System Variables page to modify the server settings.

Multiple choice

You're using Query Performance Insight to track the performance of queries running on your server. You notice that the wait statistics for one query show a significant number of lengthy IO waits. What might cause this?

- This could be due to a poorly designed query, an inefficient join operation, or a full table scan incurred because of a missing index.
- This could be caused by a high volume of network activity.
- This might be the result of connection throttling, if the server is attempting to handle a large number of concurrent requests.
- This might be the result of deploying too many databases on the same server, causing IO contention.
- This could be caused by other long running transactions locking the data on disk.

Module summary

Module Summary

In this module, you learned how to configure security for your Azure Database for MySQL/PostgreSQL server. You saw how the server automatically backs up data, and how you can restore a server after a failure. You learned how to monitor the performance of your server, and how to trace database activity and track the performance of queries running against the databases on your server. Finally, you saw how to configure read replication, to distribute read-heavy workloads and reduce read-latency across servers.

Takeaways

In this module you learned how to:

- Protect your server, and recover after a failure.
- Monitor and tune your server.

Module Lab Information

Answers

Multiple choice

You're writing a script that uses the Azure CLI to create and configure an Azure Database for MySQL service. You need to allow access to the service for other Azure services.

Which IP address range should you specify in the `az mysql server firewall-rule create` command?

- You don't need to create a firewall rule because all Azure services will automatically have access to the server.
- Set the `start-ip-address` parameter to `0.0.0.0`, and the `end-ip-address` parameter to `255.255.255.255`.
- Set the `start-ip-address` and `end-ip-address` parameters to `0.0.0.0`.
- Set the `start-ip-address` and `end-ip-address` parameters to `255.255.255.255`.
- Set the `start-ip-address` and `end-ip-address` parameters to the range defined by the subnet that contains your resources.

Explanation

Set the `start-ip-address` and `end-ip-address` parameters to `0.0.0.0`. Setting the `start-ip-address` parameter to `0.0.0.0`, and the `end-ip-address` parameter to `255.255.255.255` will also work, but is dangerous as it permits traffic from all IP addresses, effectively disabling the firewall.

Multiple choice

Restoring an Azure Database for PostgreSQL server from a backup overwrites the existing data.

True or false?

- True.
- False.

Explanation

Restore operations—*point-in-time* and *geo-restore*—both create a new server that contains the restored data. The original server, and data, are left intact.

Multiple choice

How do you change the server parameters for Azure Database for MySQL?

- Edit the `my.cnf` for the server using a text editor, then restart the server.
- Modify the server log settings for the server in the Azure portal.
- Use the Server parameters page for the server in the Azure portal.
- You can't change server parameters. The server is managed entirely by Azure.
- Connect to the server using MySQL Workbench and use the System Variables page to modify the server settings.

Explanation

The Server parameters page in the Azure portal enables you to change server parameters.

Multiple choice

You're using Query Performance Insight to track the performance of queries running on your server. You notice that the wait statistics for one query show a significant number of lengthy IO waits.

What might cause this?

- This could be due to a poorly designed query, an inefficient join operation, or a full table scan incurred because of a missing index.
- This could be caused by a high volume of network activity.
- This might be the result of connection throttling, if the server is attempting to handle a large number of concurrent requests.
- This might be the result of deploying too many databases on the same server, causing IO contention.
- This could be caused by other long running transactions locking the data on disk.

Explanation

The most likely cause is an inefficient query and possibly a missing index. In this case, the query might be attempting to read all of the data from a large table into memory. Modifying the query to retrieve fewer rows based on an index lookup will enable better caching of the results.