The science behind the report:

Microsoft Azure Database for MySQL delivered better performance and lower price per performance than competing Amazon Web Services (AWS) and Google Cloud solutions

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Microsoft Azure Database for MySQL delivered better performance and performance per performance than competing Amazon Web Services (AWS) and Google Cloud solutions.

We concluded our hands-on testing on October 13, 2023. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on October 11, 2023 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Results of our testing.

	Azure Database for MySQL	Amazon Aurora™ MySQL	Amazon RDS for MySQL	Google Cloud™ SQL for MySQL
5-min runtime				
128 threads				
TPS	9,969.88	8,703.15	6,607.71	4,404.33
Avg. latency (ms)	12.84	14.71	19.37	29.06
256 threads				
TPS	10,251.60	9,140.24	8,370.71	5,712.49
Avg. latency (ms)	24.97	28.00	30.58	44.80
10-min runtime				
128 threads				
TPS	9,996.74	8,565.33	7,101.09	4,714.79
Avg. latency (ms)	12.80	14.94	18.02	27.15
256 threads				
TPS	10,324.25	9,101.28	8,532.13	5,955.9
Avg. latency (ms)	24.79	28.12	30.00	42.98

System configuration information

Table 2: Detailed information on the systems we tested.

System configuration information	Azure Database for MySQL – Flexible Server instance	Amazon Aurora instance
Tested by	Principled Technologies	Principled Technologies
Test date	10/12/2023	10/11/2023
CSP / region	West US 3 (Zone 1)	us-west-2d
Workload and version	Sysbench 1.0.18	Sysbench 1.0.18
Workload specific parameters	20 tables, 20M rows per table, 128 and 256 threads, 5 and 10 minute tests, 70/30 R/W ratio OLTP	20 tables, 20M rows per table, 128 and 256 threads, 5 and 10 minute tests, 70/30 R/W ratio OLTP
Iterations and result choice	Three runs, median	Three runs, median
Server platform	Standard_E32ds_v5	db.r6i.8xlarge
SQL version	MySQL 8.0.32	Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28)
Date of last OS updates/patches applied	10/11/2023	10/11/2023
Processor		
Number of vCPU	32	32
Vendor and model	Intel® Xeon® Platinum 8370C	Intel Xeon Ice Platinum 8375C
Core count (per processor)	32	32
Core frequency (GHz)	2.80	2.90
Stepping	6	6
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Memory module(s)		
Total memory in system (GB)	256	256
Data drive		
Number of drives	1	1
Drive size (GB)	256	Unavailable
Drive information (speed, interface, type)	Auto-IOPS	I/O-Optimized

Table 3: Detailed information on the systems we tested.

System configuration information	Amazon RDS instance	Google Cloud MySQL Instance
Tested by	Principled Technologies	Principled Technologies
Test date	10/12/2023	10/13/2023
CSP / region	us-west-2d	us-west3-b
Workload and version	Sysbench 1.0.18	Sysbench 1.0.18
Workload specific parameters	20 tables, 20M rows per table, 128 and 256 threads, 5 and 10 minute tests, 70/30 R/W ratio OLTP	20 tables, 20M rows per table, 128 and 256 threads, 5 and 10 minute tests, 70/30 R/W ratio OLTP
Iterations and result choice	Three runs, median	Three runs, median
Server platform	db.r6i.8xlarge	db-perf-optimized-N-32
SQL version	MySQL 8.0.34	MySQL 8.0.34
Date of last OS updates/patches applied	10/11/2023	10/11/2023
Processor		
Number of vCPU	32	32
Vendor and model	Intel Xeon Ice Platinum 8375C	undisclosed
Core count (per processor)	32	undisclosed
Core frequency (GHz)	2.90	undisclosed
Stepping	6	undisclosed
Hyper-threading	Yes	undisclosed
Turbo	Yes	undisclosed
Memory module(s)		
Total memory in system (GB)	256	256
Data drive		
Number of drives	1	1
Drive size (GB)	800	2,500
Drive information (speed, interface, type)	io1, 40,000 IOPS	SSD, 60,000 IOPS

How we tested

For the Azure solution, we deployed a 32vCore Business Critical Azure Database for MySQL - Flexible Server instance (Standard_E32ds_v5) with a 256GB volume. We enabled Auto-IOPS and storage autogrowth.

For the Amazon Aurora and Amazon RDS solutions, we used a db.r6i.8xlarge instance. For Aurora, we set the storage to I/O-Optimized and enabled storage autogrowth. For RDS, we attached an 800GB io1 volume that supported up to 40,000 IOPS, the maximum available IOPS for the instance type.

For Google Cloud MySQL, we deployed a 32vCore instance on the Enterprise Plus SKU (db-perf-optimized-N-32), with a 2,500GB SSD volume with storage autogrowth enabled. This increased volume size allowed us to utilize the instance's maximum disk IOPS of 60,000 and throughput of 1200 MBps.

However, each of these services provided additional data-caching local SSDs, which allowed these instances to exceed their stated per-instance limits.

We compared the OLTP performance from sysbench 1.0.18. We sized the database to have 20 tables with 20 million rows each, running the 70/30 read/write workload at 128 and 256 threads on each of the managed database services (Azure Database for MySQL - Flexible Server, Amazon Aurora, Amazon RDS, and Google Cloud MySQL).

While testing, we used mostly out-of-the-box parameters with a few exceptions that we set to match across the different offerings. For all solutions, we set `max_prepared_stmt_count` to 655,350, enabled `log_bin` (this parameter is ON by default when users enable automated backups and point-in-time-restores), and set `binlog_format = ROW`. We set the `innodb_redo_log_capacity` to 536,870,912 where applicable (Amazon Aurora does not allow users to change this setting, and Google Cloud MySQL uses the pre-MySQL 8.0.31 setting `innodb_log_file_size` in conjunction with `innodb_log_file_count`. For the Google Cloud solution, the `innodb_log_file_count` is 2, so we set the `innodb_log_file_size` to 268,435,456 to give a product equal to the `innodb_log_redo_capacity` (specified above). Additionally, we set the `innodb_buffer_pool_size` to the maximum allowed value on each service: 75 percent of the total memory capacity for Azure and both Amazon offerings, and 72 percent of the total memory capacity for Google Cloud.

Setting up the MySQL Flexible Server environment on Azure

Creating the client VM

1. Log into the Azure Portal, and navigate to the Virtual Machines service.
2. To open the Add VM wizard, click Add.
3. On the Basics tab, set the following:
    a. Choose your Subscription from the dropdown menu.
    b. Choose your Resource group from the dropdown menu.
    c. Name the Virtual Machine.
    d. Choose your Region from the dropdown menu. We used West US 3.
    e. Under Availability options, choose your Availability zone. We used Zone 1.
    f. Under the Image dropdown menu, choose Ubuntu 20.04 LTS.
    g. Under VM architecture, select x64.
    h. Leave Azure Spot instance set to No.
    i. Select the instance size you wish to use; we used Standard D32ds_v5.
    j. Keep the Authentication type set to SSH public key.
    k. Choose a Username and Key pair name, and select generate a new key pair.
    l. Leave Public inbound ports set to Allow selected ports.
    m. For Select inbound ports, choose SSH (22).
    n. Click Next: Disks.
4. On the Disks tab, set the following:
    a. For the OS disk size, select 30GB.
    b. For the OS disk type, choose Standard SSD from the dropdown menu.
    c. Leave the default Encryption type.
    d. Click Next: Networking.
5. On the Networking tab, set the following:
    a. Leave all settings as default.
    b. Ensure the checkbox for accelerated networking is enabled.
    c. Click Next: Management.

6. On the Management tab, leave all defaults.
7. On the Advanced tab, leave all defaults.
8. On the Tags tab, add any tags you wish to use.
9. On the Review + create tab, review your settings, and click Create.

Creating the Azure Database for MySQL - Flexible Server Instance

1. Log into the Azure Portal, and navigate to the Azure Database for MySQL flexible servers service.
2. Click Create, and select Flexible Server.
3. Select your Subscription and Resource group.
4. Enter a name for the instance, and select the region in which you wish to work. We used West US 3.
5. For the MySQL version, select 8.0.
6. For the Workload type, select Tier 1 Business Critical Workloads.
7. Under Compute + storage, click Configure server.
    a. Compute tier: Business Critical
    b. Compute processor: Intel
    c. Compute size: Standard_E32ds_v5
    d. Storage size (in GiB): 256
    e. IOPS: Auto scale IOPS
    f. Storage Auto-growth: Enabled
    g. Enable high availability: Disabled
    h. Backup retention period (in days): 7
    i. Geo-redundancy: Disabled
8. Click Save to continue.
9. Choose your Availability zone. We used Zone 1.
10. Under Authentication method, select MySQL authentication only, and enter a username and password.
11. Click Next: Networking.
12. On the Networking tab, set the following:
    a. Connectivity method: Public access (allowed IP addresses) and Private endpoint
    b. Public access: enabled
    c. Add a firewall rule with your Ubuntu client IP
    d. Click Next: Security
13. On the Security tab, leave all defaults.
14. On the Tags tab, add any tags you wish to use.
15. On the Review + create tab, review your settings, and click Create.

Setting up RDS environment on AWS

Creating the MySQL 8.0 Parameter Group

1. Log into AWS, and navigate to the AWS Management Console.
2. Click RDS.
3. In the left pane, click Parameter groups.
4. Click Create parameter group.
5. Under Parameter group family, select mysql8.0.
6. Under Type, select DB Parameter Group.
7. Give the group a name and description.
8. Click Create.
9. Click into your newly created parameter group, and click Edit in the top right corner.
10. Search for and change the following parameters:

```
binlog_format = ROW
innodb_redo_log_capacity = 536870912
max_prepared_stmt_count = 655350
```

11. Click Save Changes.

Creating the client instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, and to open the Launch Instance in the dropdown wizard, click Launch instance.
4. Name the instance, and add any tags you wish to use.
5. Under Quick Start, click Ubuntu.
6. Under Amazon Machine Image (AMI), select Ubuntu Server 20.04 LTS (HVM), SSD Volume Type.
7. Keep the Architecture set to x86.
8. Under Instance Type, select m6i.8xlarge.
9. Under key pair (login), click Create key pair:
    a. Enter a key pair name.
    b. Leave RSA and .pem selected, and click Create key pair.
10. Next to Network settings, click Edit:
    a. Leave the default VPC selected.
    b. Choose your subnet. We selected us-west-2d.
    c. Leave Auto-assign public IP enabled.
    d. Under Firewall (security groups), select Create security group.
    e. Name the group, and give it a description.
    f. Ensure that there is an inbound rule for ssh over TCP on port 22, and click Configure storage.
11. Set the root volume to 30GiB on gp2 storage, and click Launch Instance.

Editing the Security Group inbound rules

1. Log into AWS, and navigate to the AWS Management Console.
2. Click EC2.
3. In the left pane, select Security Groups.
4. Click into your newly created Security group.
5. Click Edit inbound rules.
6. Click Add rule, and set the following:
    a. Type: All traffic
    b. Source: Custom→Security Groups→[your security group]
7. Click Save rules.

Creating the RDS MySQL instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click RDS.
3. Under Create database, click Create database.
4. Leave the database creation method as Standard create.
5. Under Engine options, select MySQL.
6. Under Engine Version, choose your MySQL major and minor version. We selected 8.0.34.
7. Under Templates, select Dev/Test.
8. Under Availability and durability, leave Single DB instance selected.
9. Name the instance.
10. Select a Master username and password.
11. Under Instance configuration, select Memory optimized classes, and choose db.r6i.8xlarge.
12. Under Storage, select Provisioned IOPS SSD (io1) with 800GB and 40,000 provisioned IOPS (the instance maximum).
13. Leave storage autoscaling enabled.
14. Under Networking, leave Compute resource, Network type, VPC, and DB subnet group with default settings.
15. Enable Public access.
16. Choose existing VPC security group, and select the previously created group.
17. Select your Availability zone. We used us-west-2d.
18. Leave Database authentication and Monitoring options set to default.
19. Under Additional configuration, set database parameter group to the group that you created in the Creating the MySQL 8.0 Parameter Group section.
20. Leave Backup, Encryption, and Maintenance options set to default, and click Create Database.

Setting up Aurora environment on AWS

Note: we used the same AWS client for both RDS and Aurora, so we did not repeat the instructions for creating an AWS client in this section.

Creating the Amazon Aurora 3.04.0 Parameter Groups (instance and cluster)

1. Log into AWS, and navigate to the AWS Management Console.
2. Click RDS.
3. In the left pane, click Parameter groups.
4. Click Create parameter group.
5. Under Parameter group family, select aurora-mysql8.0.
6. Under Type, select DB Parameter Group.
7. Enter a name and description for the group.
8. Click Create.
9. Click into your newly created instance parameter group, and click Edit in the top right corner.
10. Search for and change the following parameter:

```
max_prepared_stmt_count = 655350
```

11. Click Save Changes.
12. Repeat steps 4 and 5.
13. Under Type, select DB Cluster Parameter Group.
14. Enter a name and description for the group.
15. Click Create.
16. Click into the newly created cluster parameter group, and click Edit in the top right corner.
17. Search for and change the following parameters:

```
binlog_format = ROW
max_prepared_stmt_count = 655350
```

18. Click Save Changes.

Creating the Aurora (MySQL Compatible) instance

1. Log into AWS, and navigate to the AWS Management Console.
2. Click RDS.
3. Under Create database, click Create database.
4. Leave the database creation method as Standard create.
5. Under Engine options, select Aurora (MySQL Compatible).
6. Under Available versions, choose your Aurora major and minor version. We selected Aurora MySQL 3.04.0 (compatible with MySQL 8.0.28), the latest available version at the time of testing.
7. Under Templates, select Dev/Test.
8. Name the instance.
9. Select a Master username and password.
10. Under Cluster storage configuration, select Aurora I/O-Optimized.
11. Under Instance configuration, select Memory optimized classes, and choose db.r6i.8xlarge.
12. Under Multi-AZ deployment, leave Don't create an Aurora Replica selected.
13. Under Networking, leave Compute resource, Network type, VPC, and DB subnet group with default settings.
14. Enable Public access.
15. Choose existing VPC security group, and select the previously created group.
16. Select your Availability zone. We used us-west-2d.
17. Leave Database authentication and Monitoring options set to default.
18. Under Additional configuration, set DB cluster parameter group and DB parameter group to the respective groups created in the Creating the Amazon Aurora 3.04.0 Parameter Groups section.
19. Leave Backup, Encryption, and Maintenance options set to default, and click Create Database.

Setting up Google Cloud MySQL environment on Google Cloud

Creating the client instance

1. Log into Google Cloud, and navigate to the Console.
2. Click the Navigation menu, and select Compute Engine→VM instances.
3. Click Create Instance.
4. Name the instance.
5. Select your region and zone. We used us-west3-b.
6. Under Machine configuration, select N2.
7. Under Machine type, choose n2-standard-32.
8. Under Boot disk, click Change, and set the following options:
    a. Operating system: Ubuntu
    b. Version: Ubuntu 20.04 LTS
    c. Boot disk type: SSD persistent disk
    d. Size (GB): 30
9. Leave Service account and Access scopes set to default.
10. Under Firewall, select Allow HTTP traffic and Allow HTTPS traffic.
11. Leave the remaining options default, and click Create.

Creating the Google Cloud MySQL Instance

1. Log into Google Cloud, and navigate to the Console.
2. Click the Navigation menu, and select SQL.
3. Click Create Instance.
4. Click Choose MySQL.
5. Name the instance, and set a password for the root user.
6. Under Database version, select MySQL 8.0.
7. Click Show Minor Versions, and select MySQL 8.0.34 from the dropdown menu.
8. Under Choose a Cloud SQL edition, select EnterprisePlus.
9. Under preset edition, select Development.
10. Select your region. We used us-west3.
11. Leave Zonal availability set to Single zone.
12. Click Specify zones and select your AZ. We used us-west3-b.
13. Click Machine configuration, and select 32 vCPU, 256GB.
14. Leave data cache enabled.
15. Under Storage type, select SSD.
16. Under Storage capacity, select custom, and input 2,500GB to utilize the maximum IOPS and throughput for the instance.
17. Leave automatic storage increase enabled.
18. Under Connections, ensure that Public IP is checked.
19. Under Authorized networks, click Add a Network, then add a rule with your client VM IP.
20. Under Data Protection, leave Automatic daily backups and point-in-time recovery enabled, and disable deletion protection.
21. Under Flags, add the following flags:

```
innodb_buffer_pool_size = 197912092999
innodb_log_file_size = 268435456
max_prepared_stmt_count = 655350
```

22. Under Labels, add any labels you wish to use.
23. Click Create Instance.

Configuring Ubuntu 20.04 LTS client (all solutions)

Configuring Ubuntu 20.04 LTS client

1. SSH into the Ubuntu client VM.
2. Run updates and upgrade:

```
sudo apt-get update
sudo apt-get upgrade -y
```

3. Install required packages (Sysbench and MySQL client tools):

```
sudo apt-get install sysbench -y
sudo apt-get install mysql-client -y
```

Creating and preparing the test database (all solutions)

Creating the database

1. SSH into the Ubuntu client VM.
2. Connect to the MySQL instance:

```
mysql -h [mysql-instance-name] -u [username] -p
```

3. Enter your password.
4. Create a database called `testdb`:

```
CREATE DATABASE testdb;
```

5. Exit MySQL:

```
exit;
```

Preparing the database

1. Navigate to the sysbench folder:

```
cd /usr/share/sysbench
```

2. Run the sysbench prepare statement with 20 tables and 20 million rows per table:

```
date ; sysbench oltp_write_only.lua --tables=20 --table-size=20000000 --threads=20 --time=720000
--mysql-host=[mysql-instance-name] --mysql-db=testdb --mysql-user=[username] --mysql-
password='[password]' --mysql-port=3306 --report-interval=10 --percentile=99 prepare ; date
```

3. After completing the data insertion, wait 20 minutes for background tasks to complete.

Running the tests

4. On the client system, navigate to the sysbench folder:

```
cd /usr/share/sysbench
```

5. Run the sysbench 70/30 read/write workload for 30 minutes at 256 threads as a warmup:

```
sysbench oltp_read_write.lua --tables=20 --table-size=20000000 --db-driver=mysql --threads=256
--time=1800 --mysql-host=[mysql-instance-name] --mysql-db=testdb --mysql-user=[username] --mysql-
password='[password]' --mysql-port=3306 --report-interval=1 --percentile=99 run | tee ~/warmup.log
```

6. Reboot the client VM.
7. Return to the sysbench folder:

```
cd /usr/share/sysbench
```

8. Set the ulimit to two times the amount of threads to be tested:

```
ulimit -n =[threads x2]
```

9. Run the 70/30 read/write workload for five minutes at 128 and 256 threads, and for 10 minutes at the same thread sizes. Complete steps 3 through 5 after each test at a particular thread size and runtime.
10. Complete step 6 two more times, and collect the median results.

Read the report at https://facts.pt/z5OCeh2 ▶

This project was commissioned by Microsoft.

Principled Technologies®

Facts matter.®