The science behind the report:

Three Microsoft Azure SQL Managed Instances offered better SQL Server performance and value than their Amazon RDS counterparts in our tests

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 Three Microsoft Azure SQL Managed Instances offered better SQL Server performance and value than their Amazon RDS counterparts in our tests.

We concluded our hands-on testing on April 21, 2022. 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 March 29, 2022 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.

When we created our test plan and began testing, we chose the fairest possible instance comparisons. First, we tested the TPROC-H workload and chose an AWS R5b instance (db.r5b.16xlarge). Then, we tested the complex OLTP workload (MSOLTPE), choosing another AWS R5b instance (db.r5b.4xlarge).

After completing discovery testing for these workloads, where we ensured we could run the tests properly and got an idea of relative performance, Amazon RD5 announced support for a new memory-optimized instance, an R6i instance that could be an upgrade to the R5b instances we were already using. Budget and timing did not permit us to restest on the newer instance type for the TPROC-H and MSOLTPE workloads. In that same release (https://aws.amazon.com/about-aws/whats-new/2022/03/amazon-rds-sql-server-supports-m6i-r6i-instances/), Amazon announced M6i instance support as well. Because we had not yet started TPROC-C testing, we opted to use the newly supported (and more comparable) M6i instances (db.m6i.32xlarge) for that testing.

Table 1: OLTP database performance, in transactions per minute (TPM) and new orders per minute (NOPM), that the instances achieved on the HammerDB TPROC-C benchmark. Higher numbers are better. Source: Principled Technologies.

TPROC-C	Azure SQL Managed Instance 80vCore Business Critical on Premium-series hardware	Amazon RDS for SQL Server db.m6i.32xlarge
TPM	826,687	334,228
NOPM	359,495	145,156

Table 2: OLTP database performance, in transactions per second (tps), that the instances achieved on the MSOLTPE benchmark. Higher numbers are better. Source: Principled Technologies.

MSOLTPE	Azure SQL Managed Instance 16vCore Business Critical on Premium Memory-Optimized-series hardware	Amazon RDS for SQL Server db.r5b.4xlarge
tps	571.52	102.36

Table 3: Data analytics completion times, in seconds, that the instances achieved on the HammerDB TPROC-H benchmark. Lower numbers are better. Source: Principled Technologies.

TPROC-H	Azure SQL Managed Instance 64vCore Business Critical on Premium Memory-Optimized-series hardware	Amazon RDS for SQL Server db.r5b.16xlarge
1 stream	669	3,687
8 streams	5,482	23,083

In Tables 4 through 6, we use pricing offered on each service's respective website on April 12, 2022. Azure region pricing is for the US South Central region, and AWS pricing in for the US East region. For TPROC-C and MSOLTPE workload results, we calculated the price per performance by simply dividing the monthly cost by the solution's results. Because TPROC-H results deal with time, where a small number is better, we inverted the performance results to match the conventions of the other two workloads. We then divided the monthly cost by the inverted performance. Note that price/performance metrics themselves use an arbitrary length of time for cost (we could also have calculated with the hourly rates of each solution), but give a ratio independent of time for the solutions.

Table 4: Price/performance comparison using the results from our TPROC-C tests.

	Amazon RDS for SQL Server m6i.32xlarge	Azure SQL Managed Instance (pay-as-you-go pricing) 80vCore Business Critical on Premium-series hardware	Azure SQL Managed Instance (Azure Hybrid Benefit pricing) 80vCore Business Critical on Premium-series hardware
Volume	2048GB 64K IOPS	2048GB 320K IOPS	2048GB 320K IOPS
Instance cost	$77,752.30	$46,450.86	$24,660.86
Volume cost	$13,312.00	$604.80	$604.80
Total cost	$91,064.30	$47,055.66	$25,265.66
Performance (TPM)	334,228	826,687	826,687
Price/performance	$0.2725	$0.0569	$0.0306
% Price/performance cost reduction	N/A	79.11%	88.78%

Table 5: Price/performance comparison using the results from our MSOLTPE tests.

	Amazon RDS for SQL Server **db.r5b.4xlarge**	**Azure SQL Managed Instance (pay-as-you-go pricing)** **16vCore Business Critical on Premium Memory-Optimized-series hardware**	**Azure SQL Managed Instance (Azure Hybrid Benefit pricing)** **16vCore Business Critical on Premium Memory-optimized-series hardware**
Volume	2048GB 43333 IOPS	2048GB 64K IOPS	2048GB 64K IOPS
Instance cost	$11,134.69	$11,285.07	$6,905.07
Volume cost	$9,178.6	$604.80	$604.80
Total cost	$20,313.29	$11,889.87	$7,509.87
Performance (tps)	102.36	571.52	571.52
Price/performance	$198.45	$20.80	$13.14
% Price/performance cost reduction	N/A	89.52%	93.38%

Table 6: Price/performance comparison using the results from our TPROC-H tests for single-stream testing.

	Amazon RDS for SQL Server **db.r5b.16xlarge**	**Azure SQL Managed Instance (pay-as-you-go pricing)** **64vCore Business Critical on Premium Memory-Optimized-series hardware**	**Azure SQL Managed Instance (Azure Hybrid Benefit pricing)** **64vCore Business Critical on Premium Memory-Optimized-series hardware**
Volume	4608GB 64K IOPS	4608GB 256K IOPS	4608GB 256K IOPS
Instance cost	$44,540.22	$45,140.28	$27,620.28
Volume cost	$13,952.00	$1,372.80	$1,372.80
Total cost	$58,492.22	$46,513.08	$28,993.08
Performance (seconds)	3,687	669	669
Price/inverted performance (normalized)	N/A	1.60	1.00
% Price/inverted performance cost reduction	N/A	90.3%	93.9%

System configuration information

For Amazon RDS for SQL Server instances, we chose Multi-AZ to match the high availability SLA that Azure SQL Managed Instance offers, which has four replicas (one primary and three secondary, one of which is read-only). Choosing Multi-AZ on Amazon RDS is a considerable price increase from other offerings, but wanted to match availability level along with hardware for the closest solution comparison.

Table 7: Detailed information on the intances we tested with the TPROC-C workload.

System configuration information	**Amazon RDS for SQL Server** **db.m6i.32xlarge**	**Azure SQL Managed Instance** **80vCore Business Critical on Premium-series hardware**
Tested by	Principled Technologies	Principled Technologies
Test date	04/06/2022	04/08/2022
CSP / region	us-east-1 (Multi-AZ)	US South Central
Workload and version	HammerDB v4.2 TPROC-C	HammerDB v4.2 TPROC-C
Workload specific parameters	13,000 warehouses, 96 virtual users, Use All Warehouses, timed test	13,000 warehouses, 256 virtual users, Use All Warehouses, timed test
Iterations and result choice	Three runs, median	Three runs, median
Server platform	db.m6i.32xlarge	80vCore Business Critical Premium Non Memory-Optimized
SQL version	SQL Server 2019 Enterprise Edition	SQL Server 2019 Enterprise Edition
Date of last OS updates/patches applied	04/06/2022	04/08/2022
Processor		
Number of vCPU	128	80
Vendor and model	Intel® Xeon® Platinum 8375C	Intel Xeon Platinum 8370C
Core count (per processor)	32	32
Core frequency (GHz)	2.90	2.80
Stepping	6	6
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Memory module(s)		
Total memory in system (GB)	512	560
General hardware		
Storage: Network or direct attached / instance	Network attached	Direct attached
Storage bandwidth / instance (Gbps)	40	Not documented
Data drive		
Number of drives	1	1
Drive size (GB)	2,048	2,048
Drive information	io1, 64,000 IOPS	Ultra, 320,000 IOPS

Table 8: Detailed information on the instances we tested using the MSOLTPE workload.

System configuration information	Amazon RDS for SQL Server db.r5b.4xlarge	Azure SQL Managed Instance 16vCore Business Critical on Premium Memory-Optimized-series hardware
Tested by	Principled Technologies	Principled Technologies
Test date	03/30/2022	03/29/2022
CSP / region	us-east-1 (Multi-AZ)	US South Central
Workload and version	MSTPCE 1.14.0-1048	MSTPCE 1.14.0-1048
Workload specific parameters	80,000 customers, 4 drivers, 500 users, NoPacing, hit=240 UE	80,000 customers, 4 drivers, 500 users, NoPacing, hit=240 UE
Iterations and result choice	Three runs, median	Three runs, median
Server platform	db.r5b.4xlarge	16vCore Business Critical Premium Memory-Optimized
SQL version	SQL Server 2019 Enterprise Edition	SQL Server 2019 Enterprise Edition
Date of last OS updates/patches applied	03/30/2022	03/29/2022
Processor		
Number of vCPU	16	16
Vendor and model	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8370C
Core count (per processor)	24	32
Core frequency (GHz)	2.50	2.80
Stepping	7	6
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Memory module(s)		
Total memory in system (GB)	128	217.6
General hardware		
Storage: Network or direct attached / instance	Direct attached	Direct attached
Storage bandwidth / instance (Gbps)	10	Not documented
Data drive		
Number of drives	1	1
Drive size (GB)	2,048	2,048
Drive information	io1, 43,333 IOPS	Ultra, 64,000 IOPS

Table 9: Detailed information on the instances we used for TPROC-H testing.

System configuration information	Amazon RDS for SQL Server db.r5b.16xlarge	Azure SQL Managed Instance 64vCore Business on Critical Premium Memory-Optimized-series hardware
Tested by	Principled Technologies	Principled Technologies
Test date	04/05/2022	04/06/2022
CSP / region	us-east-1 (Multi-AZ)	US South Central
Workload and version	HammerDB v4.2 TPROC-H	HammerDB v4.2 TPROC-H
Workload specific parameters	3000 scale, 1 and 8 users, MAXDOP 0	3000 scale, 1 and 8 users, MAXDOP 0
Iterations and result choice	Three runs, median	Three runs, median
Server platform	db.r5b.16xlarge	64vCore Business Critical Premium Memory-Optimized
SQL version	SQL Server 2019 Enterprise Edition	SQL Server 2019 Enterprise Edition
Date of last OS updates/patches applied	04/05/2022	04/06/2022
Processor		
Number of vCPU	64	64
Vendor and model	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8370C
Core count (per processor)	24	32
Core frequency (GHz)	2.50	2.80
Stepping	7	6
Hyper-threading	Yes	Yes
Turbo	Yes	Yes
Memory module(s)		
Total memory in system (GB)	512	870.4
General hardware		
Storage: Network or direct attached / instance	Direct attached	Direct attached
Storage bandwidth / instance (Gbps)	40	Not documented
Data drive		
Number of drives	1	1
Drive size (GB)	4,560	4,560
Drive information	io1, 64,000 IOPS	Ultra, 256,000 IOPS

How we tested

TPROC-C testing

For the AWS solution, we used an RDS db.m6i.32xlarge instance with a 2TB io1 volume configured to support up to 64,000 IOPS, the maximum available IOPS for io1 volumes. For the Azure solution, we used an 80vCore Business Critical Premium Series SQL Managed Instance (MI) configured with a 2TB volume, with 320,000 IOPS via the Azure SQL Managed Instance scaling factor of 4,000 IOPS per vCore. We compared the OLTP performance from the TPROC-C workload from HammerDB 4.2. We used a 13,000-warehouse database size with 96 virtual users driving the tests on AWS, and 256 virtual users driving the tests on Azure. These user counts represent configurations that produced the best TPM results for each of the respective solutions during tuning runs.

MSOLTPE testing

For the AWS solution, we used an RDS db.r5b.4xlarge instance with a 2TB io1 volume configured to support up to 43,333 IOPS, the instance IOPS limit for r5b.4xlarge. For the Azure solution, we used a 16vCore Business Critical Premium Series Memory-Optimized SQL Managed Instance configured with a 2TB volume, with 64,000 IOPS via the Azure SQL MI scaling factor of 4,000 IOPS per vCore. We compared the OLTP performance from the MSOLTPE derived from TPC-E 1.14.0. We used an 800,000-customer database size with four drivers and 500 users driving the tests on each solution.

TPROC-H testing

For the AWS solution, we used an RDS db.r5b.16xlarge instance with a 4.5TB io1 volume configured to support up to 64,000 IOPS, the maximum available IOPS for io1 volumes. For the Azure solution, we used a 64vCore Business Critical Premium Series Memory-Optimized SQL Managed Instance configured with a 4.5TB volume, with 256,000 IOPS via the Azure SQL MI scaling factor of 4,000 IOPS per vCore. We compared the OLAP performance from the TPROC-H workload from HammerDB 4.2. We used a 3,000-scale database size, running each test with one user and eight users subsequently on each solution.

Setting up an Amazon RDS environment

Creating an S3 bucket

1. Log into AWS, and navigate to the AWS Management Console.
2. Click S3.
3. Click Create Bucket.
4. Name the bucket, and choose the region you wish to work in.
5. Add a tag if desired.
6. Leave all other options as default, and click Create Bucket. This bucket will be used to host backup files for native restore.

Creating the SQL Server 2019 Enterprise Option Group

1. Log into AWS, and navigate to the AWS Management Console.
2. Click RDS.
3. In the left pane, click Option groups.
4. Click Create Group.
5. Give the group a name and description.
6. Under engine, choose the same version of SQL as that which will be used for the RDS instance under test. We used sqlserver-ee 15.00 (i.e., SQL Server 2019 Enterprise Edition).
7. Click Create.
8. On the Option groups page, select the newly created Option group, and click Add option.
9. On the next screen, from the drop-down menu, select SQLSERVER_BACKUP_RESTORE.
10. Select Yes to Create a new role, and give the role a name.
11. Under S3 bucket, select the bucket created in the previous step.
12. Under Apply Immediately, select Yes.
13. Click Add Option. RDS instances you create with this group will now be able to perform native restores from S3.
14. On the Option groups page, select the same Option group created previously, and click Add option.
15. From the drop-down menu, select TRANSPARENT_DATA_ENCRYPTION.
16. Under Schedule for adding option, select Immediately.
17. Click Add option. You can now encrypt databases using TDE for RDS instances created with this group.

Troubleshooting the native restore role [optional]

If you have issues getting the native restore to work using the steps above, you may need to add permissions policies for the SQLSERVER_BACKUP_RESTORE role you created in the previous steps. Complete to following steps to do so:

1. Log into AWS, and navigate to the AWS Management Console.
2. Click IAM.
3. In the left pane, click Roles.
4. Click the role you created while adding the SQLSERVER_BACKUP_RESTORE option to the Option group.
5. Click Add permissions, and click Attach policies.
6. Select AWSBackupServiceRolePolicyForBackup and AWSBackupServiceRolePolicyForRestores, and click Attach policies.

Creating the RDS SQL 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 type, select Microsoft SQL Server.
6. Under Database management type, leave it set to Amazon RDS.
7. Under Edition, select SQL Server Enterprise Edition.
8. Leave the Version as the newest SQL Server 2019 15.00 version.
9. Under Templates, leave Production selected.
10. Under Settings, give the DB instance a unique name.
11. Under Credentials Settings, choose a Master username and password for logging into the SQL Server instance.
12. Under DB instance class, choose the instance type you wish to deploy. See Table 10 for the instance types we chose.
13. Under Storage type, select Provisioned IOPS SSD (io1).
14. Set the allocated storage to be large enough to host your database. See Table 10 for the volume sizes we chose.
15. Set the provisioned IOPS to your desired value. See Table 10 for the Provisioned IOPS values we chose.
16. Uncheck the Enable storage autoscaling checkbox.
17. Under Multi-AZ deployment, select Yes (Mirroring / Always On).
18. Under Connectivity, leave VPC, Subnet group, and Public access all set to defaults.
19. Under VPC security group, select Create new, and give the security group a name.
20. Scroll down past Microsoft SQL Server Windows Authentication, and click to expand the Additional configuration tab.
21. Under Option group, select the Option group created in the previous section.
22. Leave Encryption enabled.
23. Select any other personal preferences in the Additional configuration section, and click Create database.

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 drop-down wizard, click Launch instance.
4. In the search window, type Windows Server, and press Enter.
5. Next to Microsoft Windows Server 2019 Base on Quick Start, click Select.
6. On Choose Instance Type, select your instance type, and click Next: Configure Instance Details. We used m5n.4xlarge for TPROC-C and TPROC-H, and m5n.2xlarge for MSOLTPE.
7. On Configure Instance, set the following:
    a. Number of instances: 1
    b. Purchasing option: Leave unchecked
    c. Network: Default VPC
    d. Subnet: Choose the region that your RDS instance is in.
    e. Auto-assign Public IP: Enable
    f. Placement Group: Leave unchecked
    g. Capacity Reservation: Open
    h. Domain join directory: No Directory
    i. IAM role: None
    j. Shutdown behavior: Stop
8. Click Next: Add Storage

9. On Add Storage, set the following:
    a. Size: 50GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
10. Click Next: Add Tags
11. On Add Tags, add any tags you wish to use. Click Next: Configure Security Group.
12. On Configure Security Group, select the group created during the RDS instance deployment.
13. On Review, click Launch.
14. Choose the appropriate option for the key pair, and click Launch Instances.

Setting up the Azure SQL Managed Instance environment

Creating an Azure Blob container

1. Log into the Azure Portal, and navigate to the Storage accounts service.
2. Click Create.
3. Select your Subscription and Resource group.
4. Enter an account name, and select the region you wish to work in.
5. Select your preferred Performance tier and Redundancy, and click Next: Advanced. We used Standard and Geo-redundant storage (GRS).
6. On the Advanced tab, select any personal preferences, and click Next: Networking. We disabled Cross-tenant replication and used the Hot access tier.
7. On the Networking tab, leave all defaults, and click Next: Data protection.
8. On the Data protection tab, select your Data protection preferences, and click Next: Encryption. We used the default settings.
9. On the Encryption tab, elect your Encryption preferences, and click Next: Tags. We used the default settings.
10. On the Tags tab, add any desired tags, and click Review + create.
11. Review your selections, and click Create.
12. On the Storage accounts page, select your newly created storage account.
13. In the left pane, select Containers.
14. To launch the Container creation wizard, click +Container.
15. Give the container a name, and click Create. This container will be used to host backup files for native restore.

Creating the SQL Managed Instance

1. Log into the Azure Portal and navigate to the SQL Managed Instance service.
2. Click Create.
3. Select your Subscription and Resource group.
4. Enter a name for the instance, and select the region you wish to work in. We used (US) South Central.
5. Under Compute + storage, click Configure Managed Instance.
6. Under Service tier, select Business Critical.
7. Under Hardware generation, select your preferred hardware series. See Table 10 for the generations we used for each instance.
8. Under vCores, select your desired number of vCores. See Table 10 for the number of vCores we used for each instance.
9. Use the slider to specify the amount of Storage in GB to use. See Table 10 for the volume size we used for each instance.
10. Choose your desired Backup storage redundancy, and click Apply. We used Geo-redundant backup storage.
11. Under Authentication, leave Use SQL authentication selected, and enter credentials for logging into the Azure SQL MI.
12. Click Next: Networking.
13. On the Networking tab, leave all defaults, and click Next: Security.
14. On the Security tab, leave all defaults, and click Next: Additional settings.
15. On the Additional settings tab, set your time zone and maintenance window to your desired values, and click Next: Tags. We used (UTC-05:00) Eastern Time.
16. On the Tags tab, enter in any desired tags, and click Review + create.
17. Review your settings, and click Create.

Configuring the Managed Instance Virtual Network for Client Connection

1. Log into the Azure Portal, and navigate to the Resource groups service.
2. Select the Resource group containing your SQL MI Virtual network (vnet).
3. Select the virtual network for your MI.
4. In the left pane, select Subnets.
5. Click + Subnet.
6. Give the subnet a name, and leave the rest of the options default.
7. To create the client subnet, click OK.

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. From the drop-down menu, choose your Subscription.
    b. From the drop-down menu, choose your Resource group.
    c. Name the Virtual Machine.
    d. From the drop-down menu, choose your Region.
    e. Leave the Availability options set to No infrastructure redundancy required.
    f. From the Image drop-down menu, choose Windows Server 2019 Datacenter.
    g. Leave Azure Spot instance set to No.
    h. Select the instance size you wish to use; we used Standard D16ds_v4 for TPROC-C and TPROC-H, and Standard D8ds_v4 for MSOLTPE.
    i. Choose a new Username and Password for the Administrator account.
    j. Leave Public inbound ports set to Allow selected ports.
    k. For Select inbound ports, choose SSH (22).
4. On the Disks tab, set the following:
    a. From the drop-down menu, for the OS disk type, choose Standard SSD.
    b. Leave the default Encryption type.
5. On the Networking tab, set the following:
    a. From the drop-down menu, choose the Virtual network subnet you created in the Configuring the Managed Instance Virtual Network for Client Connection section.
    b. To create a new Public IP, select Create New.
    c. Leave the rest of the settings at defaults.
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.

Configuring Microsoft Windows Server 2019 client (AWS and Azure solutions)

Configuring Windows Server 2019

1. Open Server Manager, and click Local Server.
2. Disable IE Enhanced Security Configuration.
3. Change the time zone to your local time zone.
4. Change the name of your server, and when prompted, reboot.
5. Open Server Manager again, and click Local Server.
6. Click to run updates.
7. Run updates, rebooting when prompted, until the server shows no new updates to install.

Installing Microsoft ODBC 17 Driver for SQL Server

1. In a browser, navigate to https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver15.
2. Download the newest version of Microsoft ODBC Driver 17 for SQL Server (x64). We used 17.9.1.1.
3. Double-click the msodbcsql.msi to open the installer, and click Next.
4. Click I accept the terms in the license agreement, and click Next.
5. Click Next.
6. Click Install.

Installing HammerDB 4.2

1. In a browser, navigate to https://github.com/TPC-Council/HammerDB/releases.
2. Download the HammerDB-4.2-Win-x64-Setup.exe executable.
3. Double-click the .exe file, choose English, and click OK.
4. Click Yes.
5. Click Next.
6. Choose a destination location, and click Next.
7. Click Next.
8. Click Finish.

Installing and setting up the MSOLTPE benchmark kit

1. On the client instance, copy over the MSTPCE.1.14.0-1048.7z folder.
2. Unzip the folder.
3. Inside the MSTPCE.1.14.0-1048 folder, navigate to the BenchCraft folder.
4. Double-click vcredist_x64.exe, and install the package.
5. Double-click vcredist_x86.exe, and install the package.
6. Double-click BenchCraftSetup, and click Next.
7. Accept the terms, and click Next.
8. Click Install.
9. Once the installation has completed, navigate back to the parent directory.
10. Right-click the Start_MDAC_SUT_Servers.cmd file, and click Edit.
11. Fill the script in with the following information:

```
START EGen\SUTServers_With_SQL_Auth\SUT_CE_Server.exe -s [SQL_server_endpoint] -d [database_name]
-c MDAC -p 38100 -m MEE1 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_CE_Server.exe -s [SQL_server_endpoint] -d [database_name]
-c MDAC -p 38200 -m MEE2 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_CE_Server.exe -s [SQL_server_endpoint] -d [database_name]
-c MDAC -p 38300 -m MEE3 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_CE_Server.exe -s [SQL_server_endpoint] -d [database_name]
-c MDAC -p 38400 -m MEE4 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_MEE_Server.exe -s [SQL_server_endpoint] -d [database_
name] -c MDAC -p 39100 -m MEE1 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_MEE_Server.exe -s [SQL_server_endpoint] -d [database_
name] -c MDAC -p 39200 -m MEE2 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_MEE_Server.exe -s [SQL_server_endpoint] -d [database_
name] -c MDAC -p 39300 -m MEE3 -u [SQL_server_user] -w [SQL_server_password]
START EGen\SUTServers_With_SQL_Auth\SUT_MEE_Server.exe -s [SQL_server_endpoint] -d [database_
name] -c MDAC -p 39400 -m MEE4 -u [SQL_server_user] -w [SQL_server_password]
```

12. Save and close the file.

Configuring BenchCraft and creating a test file

In this section, we detail the steps to create a four-driver, 500-user test file with the NoPacing parameter set.

1. On the client instance, click Start, and open BenchCraft.
2. Click the Params tab, and right-click→New.
3. Right-click the new Parameter Set, and click Modify.
4. Name the Parameter Set NoPacing, and set the Max Txn Rate to 39999.
5. Click the Drivers tab, and right-click→New UE.
6. Under Parameter Set, choose the newly created NoPacing set.
7. Under Machine Name, enter the FQDN or IP address of the client instance hosting BenchCraft.
8. Enter a location and unique name for the Log File Name. We used C:\BenchCraft-Logs\UE_1.log.
9. Set the size of the target DB to 800000 customers.
10. For the first UE driver, check Run Data Maintenance and Run Checkpoint. For subsequent users, leave these boxes unchecked.
11. Set the Connect Rate to 60 and the Start Rate to 0.
12. Under Additional options, enter `hit=240`.
13. Click OK.
14. Repeat steps 5 through 13 three more times for a total of four UE drivers, being sure to increment the Log File Name each time.
15. In the Drivers tab, right-click→New MEE.
16. Under Machine Name, enter the FQDN or IP address of the client instance hosting BenchCraft.
17. Enter a location and unique name for the Log File Name. We used C:\BenchCraft-Logs\MEE_1.log.
18. Under SUT server to connect, enter the FQDN of the client instance hosting BenchCraft, followed by the port specified in the Start_MDAC_SUT_Servers.cmd file above (i.e., for MEE1, use [client-instance-FQDN]:39100).
19. Click OK.
20. Repeat steps 15 through 19 three more times for a total of four MEE drivers, making sure to increment the Log File Name and SUT port number each time.
21. Click the Users tab.
22. Double-click DRIVER1.
23. Set the number of users to 125.
24. Under SUT Front-End Server, enter the FQDN of the client instance hosting BenchCraft, followed by the port specified in the Start_MDAC_SUT_Servers.cmd file above (i.e., for UE1, use [client-instance-FQDN]:38100).
25. Repeat steps 22 through 24 for DRIVER2, DRIVER3, and DRIVER4, making sure to increment the port number each time.
26. Click File→Save As, and give your test file a name.
27. To create an .xml file containing your test profile configuration, click Save.

Creating, backing up, and uploading the test databases

The following sections detail how we created TPROC-C and TPROC-H database backups on a local client-server Microsoft Windows Server 2019 pair in our datacenter. Microsoft provided us with the 80,000 customer TPC-E-like database backup, and we detail the creation steps they provided here.

Creating the TPROC-C database

1. Open SQL Server Management Studio.
2. Right-click Databases→New Database.
3. Name the database. We named ours tpcc.
4. Navigate to the Filegroups tab, and click Add Filegroup.
5. Name the new filegroup, and set it to Default.
6. Navigate to the Files tab, and to add data files, click Add. We used four additional files.
7. Pre-grow the data files to 350GB each.
8. Pre-grow the mdf file and log file to 300GB each.
9. Name the new database files.
10. In the Options tab, set the Recovery Model to FULL., and click OK to begin the creation process.

Populating the TPROC-C database

1. Open HammerDB, and click Options→Benchmark.
2. Choose MSSQL Server and TPROC-C.
3. Expand SQL Server→TPROC-C→Schema Build.
4. Double-click Options.
5. Input the IP of the SUT in the SQL Server field.
6. Select 13000 warehouses and 16 virtual users.
7. Click OK.
8. Double-click Build.
9. To back up the database, see the Backing up the database into multipart backup files section below.

Creating the MSOLTPE database files and tables

To see the scripts referenced here, access the zip file located at <link to come>.

1. Open SQL Server Management Studio.
2. Create the empty database files by running the Create_Database_4-Files.sql T-SQL script.
3. Create the MSOLTPE scaling tables and appropriate check constraints by running the Create_Tables_Scaling.sql and Create_Check_Constraints_Scaling.sql T-SQL scripts.
4. Create the MSOLTPE growing tables and appropriate check constraints by running the Create_Tables_Growing.sql and Create_Check_Constraints_Growing.sql T-SQL scripts.
5. Create the MSOLTPE fixed tables and appropriate check constraints by running the Create_Tabes_Fixed.sql and Create_Check_Constraints_Fixed.sql T-SQL scripts.
6. Install the MSOLTPE stored procedures and create the Data Maintenance Audit Table using the T-SQL scripts in the DML section.
7. Update the SQL Server options to prepare for bulk data load by running the Database_Options_1.sql T-SQL script.

Creating and loading the MSOLTPE database files

Note: The setup process utilized by Microsoft dynamically generates EGenLoader cmd files based on the number of cores on the host machine and the number of customers (i.e., database size). We created the EGenLoader cmds and bulk insert cmds referred to here on a 40vCore host for a DB size of 800,000 customers.

1. If using the MSTPCE.1.14.0-1048 kit, skip directly to step 9.
2. On the client instance, navigate to https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp.
3. Download the newest version of TPC-E Tools.
4. Extract the zip file.
5. Download and install the Community edition of Visual Studio: https://visualstudio.microsoft.com/downloads/.
6. Open Visual Studio, and click Open a Project or Solution.
7. Navigate to the folder containing the EGen.sln solution, and load it.
8. Build the EGenLoader.vcproj project in Visual Studio.
9. Use the EGenLoaderInstance[#].cmd scripts to create the flat files. Note: If you built the EGenLoader exectuable yourself, update the directories pointed to in the scripts accordingly.
10. After the flat files are generated, use the BCP_[#] cmds to load the data into the database tables created in the previous section.
11. Convert the NI_ITEM column to a varbinary(max) using the Convert_NI_ITEM_Data.sql T-SQL script. Note: The EGenLoader code, when generating flat files, generates the NI_ITEM column as a text field. The TPC-E specification requires that the NI_ITEM column be defined as a LOB (varbinary(max) in SQL Server terms).

Creating the MSOLTPE database Indexes and foreign key constraints

1. Open SQL Server Management Studio.
2. Create the indexes on the MSOLTPE fixed tables by running the Create_Indexes_Fixed_Tables.sql T-SQL script.
3. Create the indexes on the MSOLTPE growing tables by running the Create_Indexes_Growing_Tables.sql T-SQL script.
4. Create the indexes on the MSOLTPE scaling tables by running the Create_Indexes_Scaling_Tables.sql T-SQL script.
5. Create the foreign key constraints by running the Create_FK_Constraints.sql T-SQL script.
6. Setup the post-load database options by running the Database_Options_2.sql T-SQL script.
7. To back up the database, see the Backing up the database into multipart backup files section below.

Creating the TPROC-H database and its tables

1. Open SQL Server Management Studio.
2. Right-click Databases→New Database.
3. Name the database. We named ours tpch.
4. Click on the Filegroups tab, and click Add Filegroup.
5. Name the new filegroup, and set it to Default. We named ours DATA_FG.
6. On the General tab, click Add to create 32 additional data files.
7. Name each file, and set the initial size for each new file to 100GB. Make sure all new files are on the new filegroup you created.
8. On the Options tab, set the Recovery Model to Full.
9. To start the build, click OK.
10. TPC specifies eight specific tables with specific columns: Part, Supplier, PartSupp, Customer, Nation, Lineitem, Region, and Orders. To run a SQL query to create these tables in your new database, run the script we provide below.

Generating the TPROC-H data, loading it into the database, and creating Indexes

In this step, we used a tool called dbgen to create the flat files for the 3000-scale database that we then loaded into the empty database.

1. Download the dbgen tool from GitHub: https://github.com/electrum/tpch-dbgen.
2. Unzip the file to a volume with roughly 3.5TB capacity.
3. Download and install the Community edition of Visual Studio: https://visualstudio.microsoft.com/downloads/
4. Navigate to the dbgen folder, and build the tpch.vcproj project in Visual Studio.
5. Navigate to the debug folder inside the dbgen folder, and shift right-click in the folder. Click Open PowerShell Window here.
6. In PowerShell, test to make sure dbgen.exe works by typing `./dbgen.exe -h`. If the help menu pops up, continue. If you get an error, try copying the dbgen.exe file to the dbgen folder, opening PowerShell there, and running the command again.
7. Once you have dbgen.exe working properly, create the 3000 scale files with the following command:

```
./dgben.exe -s 3000
```

8. Once the files have finished building, you can load them into the database. Open SQL Server Management Studio and run bulk insert commands to load the files into the tables. See below for the commands we used.
9. When the data has finished loading, run the index creation query script given below.
10. To back up the database, see the Backing up the database into multipart backup files section below.

Backing up the database into multipart backup files

1. Open SQL Server Management Studio.
2. Right-click the database you wish to back up, and click Tasks→Back up….
3. Ensure that the database you wish to back up is the selected Database, and that the Backup type is Full.
4. Under destination, click Add...
5. Under file name, type in or select your destination folder, and enter the name of the first backup file. Use the convention [db_backup_name]_1.bak.
6. Click OK.
7. Repeat steps 4 through 6 for each additional backup file, incrementing the number by one each time. We split our backups into eight files each.

Uploading the databases to AWS S3

1. On the server hosting the database, download the AWS CLI installer from https://awscli.amazonaws.com/AWSCLIV2.msi.
2. Double-click the installer to start the setup wizard, and click Next.
3. Check the License Agreement box, and click Next.
4. Click Next.
5. Click Install.
6. Once the AWS CLI installation has completed, open a command prompt, and run the following command:

```
aws configure
```

7. Meanwhile, open a browser, and navigate to the AWS Management Console.
8. Click IAM.
9. Under Quick Links in the right pane, click My security credentials.
10. To expand the tab, click Access keys.
11. Click Create New Access Key, and make note of both the Access Key ID and Secret Access Key.
12. Back in the command prompt, enter the Access Key ID, and press Enter.
13. Enter the Secret Access Key, and press Enter.
14. Enter your preferred Default region, and press Enter. We used us-east-1.
15. Enter your default output format. We left this value default.
16. Within your command prompt window, navigate to the folder where the database backup files are located:

```
cd [backup file folder location]
```

17. Run the following command to upload the first part of your multipart backup file to the S3 bucket you created in the Creating an S3 bucket section:

```
aws s3 cp [db_backup_name]_1.bak s3://[bucket_name]/
```

18. Open a new command window for each backup file, and repeat steps 16 and 17 for each, incrementing the number by one each time, for a parallel multipart upload.

Uploading the database to Azure Blob

1. On the server hosting the database, download the Azure CLI installer from https://aka.ms/installazurecliwindows.
2. Double-click the azure-cli-[version].msi to start the setup wizard, and click Next.
3. Check the License Agreement box, and click Install.
4. Once the Azure CLI installation has completed, open a command prompt, and run the following command:

```
az login
```

5. Once your default browser loads the Azure sign-in page, log in with your Microsoft account credentials.
6. Set your Subscription to the correct value by running the following command:

```
az account set --subscription "[subscription_name]"
```

7. Navigate to the folder where the database backup files are located:

```
cd [backup file folder location]
```

8. Run the following command to upload the first part of your multipart backup file to the container created in the Creating an Azure Blob container section:

```
az storage azcopy blob upload -c [container_name] --account_name [account_name] -s [file_name_on_
server]_1.bak -d [file_name_on_Azure_blob]_1.bak
```

9. Open a new command window for each backup file, and repeat steps 6 and 7 for each file, incrementing the number by one each time, for a parallel multipart upload.

Restoring and configuring the databases on the SUT (AWS and Azure solutions)

Restoring the database on RDS Instance from AWS S3

1. Log into the client instance.
2. Open SQL Server Management Studio (SSMS).
3. Connect to the RDS instance by putting in the endpoint FQDN and credentials set during deployment.
4. Click New Query, and enter the following query to restore from a multipart backup file:

```
exec msdb.dbo.rds_restore_database
@restore_db_name='[db_name]';
@s3_arn_to_restore_from='arn:aws:s3:::[bucket-name]/[db_backup_name]_*';
```

5. To check the progress of the restore task, run the following command in a New Query window:

```
exec msdb.dbo.rds_task_status;
```

Enabling database features on AWS

1. After the restore has completed, run the following queries:
2. To enable QueryStore, run:

```
ALTER DATABASE [db_name]
SET QUERY_STORE = ON (OPERATION_MODE = READ_WRITE);
```

3. To enable Accelerated Database Recovery (ADR), run:

```
ALTER DATABASE [db_name] SET ACCELERATED_DATABASE_RECOVERY = ON;
GO
```

4. To enable Transparent Data Encryption (TDE), first run the following to determine the correct RDSTDECertificate to use:

```
USE [master]
GO
SELECT name FROM sys.certificates WHERE name LIKE 'RDSTDECertificate%'
GO
```

5. To create a Database Encryption Key (DEK), run: (Where [RDSTDECertificateName] is the name of the certificate found in the previous step.)

```
USE [db_name]
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE [RDSTDECertificateName]
```

6. To enable encryption on the database, run:

```
ALTER DATABASE [db_name]
SET ENCRYPTION ON
GO
```

7. To check the status of the encryption process, use the following command:

```
USE [master]
GO
SELECT name FROM sys.databases WHERE is_encrypted = 1
GO
SELECT db_name(database_id) as DatabaseName, * FROM sys.dm_database_encryption_keys
GO
```

Creating a snapshot on AWS

1. Log into AWS, and navigate to the AWS Management Console.
2. Click RDS.
3. Click Snapshots.
4. In the Manual tab, click Take Snapshot.
5. Select your RDS instance, name the snapshot, and click Create Snapshot.

Restoring the database on SQL Managed Instance from Azure Blob

1. Log into the client VM.
2. Open SQL Server Management Studio (SSMS).
3. Connect to the RDS instance by putting in the endpoint FQDN and credentials set during deployment.
4. Right-click Databases, and click Restore Database.
5. To select the source of the backup file , click the ellipsis (…) next to Device.
6. Leave Backup media type set to URL, and click Add.
7. Next to Azure storage container, click Add.
8. Click Sign In.
9. Sign in using your Microsoft Account credentials.
10. Select your subscription, storage account, and blob container.
11. Click Create Credential, and click OK.
12. In the left pane, expand the folder structure to show the folder where the backup files are located. Select all the backup files from the multipart backup file, and click OK.
13. Specify the destination database name, and click OK to begin the restore process.

Enabling database features on Azure

1. After the restore has completed, run the following queries:
2. To enable QueryStore, run:

```
ALTER DATABASE [db_name]
SET QUERY_STORE = ON (OPERATION_MODE = READ_WRITE);
```

3. To enable Accelerated Database Recovery (ADR), run:

```
ALTER DATABASE [db_name] SET ACCELERATED_DATABASE_RECOVERY = ON;
GO
```

4. To enable Transparent Data Encryption (TDE), run:

```
ALTER DATABASE [db_name]
SET ENCRYPTION ON;
GO
```

Performing the TPROC-C test on Windows Server 2019

1. On the client system, start HammerDB.
2. Select Options→Benchmark.
3. Set the database server to SQL Server, and set the workload to TPROC-C.
4. Open the Options panel for the Driver Script: SQL Server→TPROC-C→Driver Script→Options.
5. Enter the Endpoint FQDN of the system under test in the SQL Server field.
6. Change the ODBC Driver to ODBC Driver 17 for SQL Server.
7. Choose SQL Server Authentication, and change the SQL Server User Password to the password you chose during SQL Server setup.
8. Choose Timed Driver Script.
9. Change the Total Transactions per User to 1000000000.
10. Enter 15 for the Rampup Time, and 30 for the Test Duration.
11. Select Use All Warehouses.
12. Click OK.
13. Open the Options panel for the Virtual Users: SQL Server→TPROC-C→Virtual User→Options.
14. Use 96 Virtual Users for AWS, or 256 Virtual Users for Azure.
15. Select the following: Show Output, Log Output to Temp, and Use Unique Log Name.
16. Click OK.
17. Click Create Users.
18. Click the green arrow to begin the test.
19. Once the test is complete, save the HammerDB results text file and AWS / Azure metrics output.
20. To repeat on AWS: Delete the RDS instance and restore from snapshot, and repeat steps 1 through 18 of the Performing the test on Windows Server 2019 section.
21. To repeat on Azure: Delete the SQL Managed Instance, then create a new instance following the steps in the Creating the AWS RDS Instance section. Repeat steps 1 of the Restoring the database on SQL Managed Instance from Azure Blob section through step 18 of the Performing the test on Windows Server 2019 section.
22. Run the test three times, and report the median run.

Performing the TPC-E test on Windows Server 2019

1. On the client system, open BenchCraft.
2. In the MSTPCE folder, double-click the Start_MDAC_SUT_Servers.cmd command file.
3. In BenchCraft, click Open.
4. Select the .xml file you created in the Configuring BenchCraft and creating a test file section.
5. Click Launch.
6. Once the respective windows for each UE and MEE have launched, click Start. The users should begin connecting, but not yet become active.
7. Once all 500 users have connected, click Rates.
8. Change the start rate to a value between 10 and 60, depending on how quickly you wish to ramp up.
9. Let the test run for 90 minutes.
10. Once the test has run its course, click Pause.
11. Once the users have all been paused, click Stop.
12. To collect results for the test, click Tools→Sort/Merge.
13. Ensure that the correct logs are selected (they should populate automatically), and name your output file. Click OK.
14. Once the Sort/Merge is completed, click Tools→Transaction Report.
15. Truncate the Start and End times by 15 minutes to create a window of one hour. Name the output file, and click OK.
16. To create a Transaction Step Report, click Tools→Transaction Step Report.
17. Truncate the Start and End times by 15 minutes to create a window of one hour. Name the output file, and click OK.
18. Repeat the test two more times for a total of three runs, and record the median run.

Performing the TPROC-H test on Windows Server 2019

1. On the client system, start HammerDB.
2. Select Options→Benchmark.
3. Set the database server to SQL Server, and set the workload to TPROC-H.
4. Open the Options panel for the Driver Script: SQL Server→TPROC-H→Driver Script→Options.
5. In the SQL Server field, enter the Endpoint FQDN of the system under testChange the ODBC Driver to ODBC Driver 17 for SQL Server.
6. Set the scale to 3000, MAXDOP to 0, and check the box for Clustered Columnstore. Click OK.
7. Expand Virtual User, and double-click Options.
8. For the power test, choose one user.Check the boxes for Show Output, Log Output to Temp, and Use Unique Log Name.
9. Click OK.
10. Double-click Create users.
11. To begin the run, click Start.
12. Once the power test has finished, record the results, and click Stop.
13. Double-click Virtual User→Options.
14. Set the number of users to 8 for the throughput test.
15. Double-click Create users.
16. To begin the run, click Start.
17. Once the throughput test has finished, record the results, and click Stop.
18. Repeat the test two more times for a total of three runs at each user count, and record the median run.

Table 10: Specifications for the instances we used in our testing.

Instance type	80vCore Business Critical Premium	m6i.32xlarge	64vCore Business Critical Premium Memory-Optimized	r5b.16xlarge	16vCore Business Critical Premium Memory-Optimized	r5b.4xlarge
CSP	Azure	AWS	Azure	AWS	Azure	AWS
Workload	TPROC-C	TPROC-C	TPROC-H	TPROC-H	MSOLTPE	MSOLTPE
vCores	80	128	64	64	16	16
Memory	560	512	870.4	512	217.6	128
Volume size (GB)	2,048	2,048	4,608	4,608	2,048	2,048
Volume IOPS	320,000	64,000	256,000	64,000	64,000	43,333

Read the report at https://facts.pt/GZUp6xk ▶

This project was commissioned by Microsoft.

Principled Technologies®

Facts matter.®