The science behind the report:

Realize better value and performance migrating from Azure Database for PostgreSQL – Single Server to Flexible Server with AMD EPYC

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 Realize better value and performance migrating from Azure Database for PostgreSQL – Single Server to Flexible Server with AMD EPYC.

We concluded our hands-on testing on January 30, 2024. 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 December 21, 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.

HammerDB testing – 16vCPU configurations	Flexible Server	Single Server	Improvement with Flexible Server
NOPM	30,605	6,490	4.71x
TPM	70,458	15,006	4.69x
CPU utilization	52.4%	22.1%	-
IOPS	37.7K	13.2K	2.85x

Table 2: Total monthly cost estimates for Azure Database for PostgreSQL – Flexible Server and Azure Database for PostgreSQL – Single Server on January 17, 2024, using the Azure Pricing Calculator.

Service category	Service type	Region	Description	Estimated monthly cost
Flexible Server				
Databases	Azure Database for PostgreSQL	East US	Flexible Server Deployment, General Purpose Tier, 1 D16ads_v5(16 vCores) x 730 Hours, 1 TB Premium SSD v2 Storage, 51,200 IOPS, 0 GB Additional Backup storage with LRS, without High availability	$2,172.72
Single Server				
Databases	Azure Database for PostgreSQL	East US	Single Server Deployment, General Purpose Tier, 1 Gen 5 (16 vCore) x 730 Hours, 6,656 GB Storage, 0 GB Additional Backup storage - LRS redundancy	$1,788.608

System configuration information

Table 3: Detailed information on the instances we tested.

Configuration information	Azure Database for PostgreSQL – Flexible Server instance	Azure Database for PostgreSQL – Single Server instance
Tested by	Principled Technologies	Principled Technologies
Test date	1/19/2024	1/19/2024
CSP / region	East US (Zone 1)	East US (Zone 1)
Workload and version	HammerDB 4.9	HammerDB 4.9
Workload specific parameters	5,000 warehouses, pg_partition=true, pg_storedprocs=true, pg_allwarehouses=true, 10 minutes warmup, 10 minutes run time	5,000 warehouses, pg_partition=true, pg_storedprocs=true, pg_allwarehouses=true, 10 minutes warmup, 10 minutes run time
Iterations and result choice	Three runs, median	Three runs, median
Instance type	Standard_D16ads_v5	GP_G5_16
PostgreSQL version	PostgreSQL 16	PostgreSQL 11
Date of last OS updates/patches applied	12/21/2023	12/21/2023
Processor		
Number of vCPUs on instance	16	16
Cloud instance memory		
Total memory in instance (GB)	64	80 (5GB per vCPU)
Data drive		
Number of drives	1	1
Drive size (GB)	1,024	6,656
Drive information (speed, interface, type)	Premium SSD Version2, 51,200 IOPS	Premium SSD, 20,000 IOPS

How we tested

We deployed two Azure Database instances for PostgreSQL: a Flexible Server instance (Standard_D16ads_v5) with a 1TB Premium SSD version2 volume configured with 51,200 IOPS and throughput of 768 MBps, and a Single Server instance (GP_Gen5_16) with a 6.5TB volume that allowed us to utilize the instance's maximum disk IOPS of 20,000. We compared the OLTP performance using the TPROC-C workload from Hammerdb 4.9. We created a tpcc database with 5,000 warehouses with partition and storedprocs enabled. We ran the TPROC-C workload using 64 virtual users with 10 minutes warmup time and 10 minutes runtime. On the database server side, we used all out of the box server parameters for both Flexible Server and Single Server instances.

Setting up PostgreSQL testing 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 drop-down menu.
    b. Choose your Resource group from the drop-down menu.
    c. Name the Virtual Machine.
    d. Choose your Region from the dropdown menu. We used East US.
    e. Under Availability options, choose Availability zone.
    f. Choose your Availability zone. We used Zone 1.
    g. Choose Ubuntu 20.04 LTS under the Image drop-down menu.
    h. Under VM architecture, select x64.
    i. Leave Azure Spot instance set to No.
    j. Select the instance size you wish to use; we used Standard D32ds_v5.
    k. Keep the Authentication type set to SSH public key.
    l. Choose a Username and Key pair name, and select generate a new key pair.
    m. Leave Public inbound ports set to Allow selected ports.
    n. For Select inbound ports, choose SSH (22).
    o. 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 drop-down 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 PostgreSQL – Flexible Server instance

1. Log into the Azure Portal, and navigate to the Azure Database for PostgreSQL Flexible Server 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 you wish to work in. We used East US.
5. For the PostgreSQL version, select 16.
6. For Workload type, select Production (Small / Medium size)
7. Under Compute + storage, click Configure server.
    a. Compute tier: General Purpose
    b. Compute processor: AMD
    c. Compute size: Standard_D16ads_v5
    d. Storage type: Premium SSD v2
    e. Storage size (in GiB): 1024
    f. IOPS: 51200
    g. Throughput: 768
    h. Storage Auto-growth: Disabled
    i. Enable high availability: Disabled
    j. Backup retention period (in days): 7
    k. Geo-redundancy: Disabled
    l. To continue, click Save.
8. Choose your Availability zone. We used Zone 1.
9. Under Authentication method, select PostgreSQL authentication only, and enter a username and password.
10. Click Next: Networking.
11. On the Networking tab, set the following:
    a. Connectivity method: Public access (allowed IP addresses).
    b. Add a firewall rule with your Ubuntu client IP.
    c. Click Next: Security.
12. On the Security tab, leave all defaults.
13. On the Tags tab, add any tags you wish to use.
14. On the Review + create tab, review your settings, and click Create.

Creating the Azure Database for PostgreSQL – Single Server instance

1. Log into Azure Cloud Shell using your account credentials.
2. Create a single server instance:

```
az postgres server create --resource-group <resource group name> --name <single server instance
name> --location eastus --admin-user <admin username> --admin-password <admin password> --sku-name
GP_Gen5_16 --storage-size 6815744 --version 11
```

3. Create a firewall rule:

```
az postgres server firewall-rule create --resource-group <resource group name> --server <single
server instance name> --name <firewall rule name> --start-ip-address <IP of the client VM> --end-ip-
address <IP of the client VM>
```

Configuring the 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 (HammerDB and PostgreSQL client library):

```
sudo apt install gnupg2 wget vim
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
curl -fsSL https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/postgresql.gpg
    sudo apt update
sudo apt install postgresql-16
wget https://github.com/TPC-Council/HammerDB/releases/download/v4.9/HammerDB-4.9-Linux.tar.gz
tar -xzvf HammerDB-4.9-Linux.tar
```

Creating the TPC-C-like database schema

1. SSH into the Ubuntu client VM, and navigate to the HammerDB directory.
2. Create a pg_tpcc_buildschema.tcl script file:

```
#!/bin/tclsh
puts "SETTING CONFIGURATION"
dbset db pg
dbset bm TPC-C
diset connection pg_host <flexible server FQDN | single server FQDN>
diset connection pg_port 5432
diset connection pg_sslmode prefer
set vu 32
set warehouse 5000
diset tpcc pg_count_ware $warehouse
diset tpcc pg_num_vu $vu
diset tpcc pg_superuser <super username>
diset tpcc pg_superuserpass <super user password>
diset tpcc pg_defaultdbase postgres
diset tpcc pg_user <pg username>
diset tpcc pg_pass <pg user password>
diset tpcc pg_dbase tpcc
diset tpcc pg_tspace pg_default
diset tpcc pg_storedprocs true
diset tpcc pg_partition true
puts "SCHEMA BUILD STARTED"
buildschema
puts "SCHEMA BUILD COMPLETED"
```

3. Build a TPCC database schema:

```
./hammerdbcli auto pg_tpcc_buildschema.tcl
```

Running the tests

1. SSH into the Ubuntu client VM, and navigate to the HammerDB folder:
2. Create a pg_tpcc_run.tcl file:

```
dbset db pg
dbset bm TPC-C

diset connection pg_host <flexible server FQDN | single server FQDN>
diset connection pg_port 5432
diset connection pg_sslmode prefer

diset tpcc pg_driver timed
diset tpcc pg_rampup 10
diset tpcc pg_duration 10
diset tpcc pg_num_vu 64
diset tpcc pg_count_ware 5000
diset tpcc pg_user pt
diset tpcc pg_pass Password1!
diset tpcc pg_storedprocs true
diset tpcc pg_total_iterations 10000000
diset tpcc pg_allwarehouse true

loadscript
vuset vu 64
vuset logtotemp 1
vucreate
vurun
vudestroy
```

3. Run the test with 5,000 warehouses, a 10-minute warmup time, and a 10-minute run time:

```
./hammerdb_cli auto pg_tpcc_run.tcl
```

4. Log into psql and drop the tpcc database.
5. Recreate the tpcc database:

```
./hammerdbcli auto pg_tpcc_buildschema.tcl
```

6. Repeat steps 3 through 5 twice more, and collect the median results.

Migrating a database from Single Server to Flexible Server

Here we describe steps for creating the target Flexible Server, completing prerequisite tasks for the migration, and performing the migration itself.

For a full list of migration best practices, please see the following link:
https://learn.microsoft.com/en-us/azure/postgresql/migrate/best-practices-seamless-migration-single-to-flexible.

Creating the target Azure Database for PostgreSQL – Flexible Server instance

1. Log into the Azure Portal and navigate to the Azure Database for PostgreSQL – Flexible Server Portal.
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 you wish to work in. We used East US.
5. For the PostgreSQL version, select 11. [Note: You can upgrade this to a newer version post-migration.]
6. For Workload type, select Production (Small / Medium size).
7. Under Compute + storage, click Configure server.
    a. Compute tier: General Purpose
    b. Compute processor: AMD
    c. Compute size: Standard_D16ads_v5
    d. Storage type: Premium SSD v2
    e. Storage size (in GiB): 1024
    f. IOPS: 20,000
    g. Throughput: 384
    h. Storage Auto-growth: Disabled
    i. Enable high availability: Disabled
    j. Backup retention period (in days): 7
    k. Geo-redundancy: Disabled
    l. To continue, click Save.
8. Choose your Availability zone. We used Zone 1.
9. Under Authentication method, select PostgreSQL authentication only, and enter a username and password.
10. Click Next: Networking.
11. On the Networking tab, set the following:
    a. Connectivity method: Public access (allowed IP addresses).
    b. Add a firewall rule with your Ubuntu client IP.
    c. Click Next: Security.
12. On the Security tab, leave all defaults.
13. On the Tags tab, add any tags you wish to use.
14. On the Review + create tab, review your settings, and click Create.

Completing migration prerequisites

For a full list of migration prerequisites, please see the following link:
https://learn.microsoft.com/en-us/azure/postgresql/migrate/concepts-single-to-flexible#migration-prerequisites.

1. Ensure that both the Flexible Server and Single Server instances have public access turned on with firewall rules (we set these during instance creation).
2. Allow-list required extensions: Log into the Ubuntu client, and connect to the tpcc database in the PostgreSQL instance.
3. Use the following command to list all extensions being used on your Single Server database:

```
select * from pg_extension;
```

4. Check if the list contains any of the following extensions:
    a. PG_CRON
    b. PG_HINT_PLAN
    c. PG_PARTMAN_BGW
    d. PG_PREWARM
    e. PG_STAT_STATEMENTS
    f. PG_AUDIT
    g. PGLOGICAL
    h. WAL2JSON

If yes, follow the next steps. Note: Our db did not require any additional allow-listing before migration.

5. Navigate to the Flexible Server instance Server Parameters blade, and search for the shared_preload_libraries parameter.
6. Select the list of above extensions used by your Single Server database to this parameter, and select Save.

Migrating the database with the Single to Flexible Migration tool

1. Navigate to the target Flexible Server instance, and click the Migration blade.
2. Click Migrate from Single Server.
3. Select Validate and Migrate, and click Next.
4. Select the source Single Server instance, enter its admin-user password, and click Next.
5. Select the target Flexible Server instance, enter its admin-user password, and click Next.
6. Select the database to be migrated, and click Next. We used the same HammerDB-generated 5000wh tpcc db used in performance testing.
7. Review, and click Start. Once the validation is completed, migration will be triggered if all checks are in Succeeded or Warning state.

Note: Along with data migration, user/role information, ownership of database objects, and permissions of database objects also migrate to the target server.

Read the report at https://facts.pt/Y72pat3 ▶

This project was commissioned by Microsoft.