



# **AlloyDB Omni for PostgreSQL - Transactional (OLTP) Benchmarking Guide**

July 2024

<b>Disclaimer</b>	<b>2</b>
<b>Overview</b>	<b>3</b>
Benchmarking Process	3
<b>Infrastructure Setup</b>	<b>5</b>
Provision the server and client VMs	5
Provision Server on GCE	5
Provision Client on GCE	8
<b>Install AlloyDB Omni</b>	<b>9</b>
Install Docker	9
Create an ext4 filesystem	10
Set up VM configurations	10
Start AlloyDB Omni	11
Allow access from the client VM	11
Update database configuration	12
Start AlloyDB Omni	13
<b>Setup of Benchmark Driver Machine (Client)</b>	<b>13</b>
Install PostgreSQL tools	13
Install Docker	14
<b>TPC-C Benchmark</b>	<b>15</b>
Benchmark configurations	15
Prerequisites	16
Initial Setup on Client Machine	16
Script to load TPC-C data	17
Running the TPC-C benchmark	18
Analyzing TPC-C Results	19
<b>TPC-B Benchmark</b>	<b>20</b>
Benchmark configurations	20
Load data	21
Run TPC-B	21
<b>Appendix 1: Observability</b>	<b>22</b>
<b>Appendix 2: Notes on performance benchmarking</b>	<b>24</b>
Benchmark Cleanup	24
Understanding system performance	24
CPU performance	25
Disk performance	25
Network latency	25

## Disclaimer

---

This AlloyDB Omni benchmark guide provides best practices for running an Online Transactional Processing (OLTP) benchmark. Your results may vary depending on several factors including, but not limited to the machine specifications of your AlloyDB Omni instance, type of client machine driving the benchmark, region, zone, and network bandwidth at the time of tests. Nothing in this user guide should be construed as a [promise](#) or [guarantee](#) about the results you'll derive from measuring the OLTP performance of AlloyDB Omni.

# Overview

---

AlloyDB Omni is a downloadable edition of AlloyDB, designed to run anywhere – in your data center, on your laptop, at the edge, and in any cloud. AlloyDB Omni has several components and features, such as state-of-the-art log and transaction management, dynamic memory management, and a built-in columnar engine. As a whole, these features enable high performance for your transactional (OLTP), analytical (OLAP), and hybrid (HTAP) workloads.

Relational database systems typically require a database administrator (DBA) to optimize them for benchmarking, which includes configuring the transaction log settings, establishing the right buffer pool sizes, and tweaking other important database parameters (flags) and characteristics. These settings also vary based on machine hardware.

During installation, AlloyDB Omni chooses settings that are likely to be optimal for the number of CPUs and memory on your system. It requires minimal to no tuning of flags at the database level to achieve high OLTP performance. Users may further adjust the settings to optimize performance for their specific workload.

This document describes step-by-step procedures and best practices to configure AlloyDB Omni, a client machine, and scripts to setup, load and run benchmarks. We will be running [HammerDB TPROC-C \(derived from TPC-C\)](#) and [Pgbench TPC-B like benchmarks](#) with different test parameters.

NOTE: Since HammerDB's TPROC-C implementation is a close variant of the official TPC-C benchmark, we will use the terms TPC-C and TPROC-C interchangeably throughout this user guide.

Similarly, we will use TPC-B and "TPC-B like" interchangeably throughout.

## Benchmarking Process

---

We'll go through the following steps to set up and run various OLTP benchmarks.

1. Configure AlloyDB Omni running on a Google Compute Engine (GCE) VM. We describe running on 2 different VM shapes:
  - a. N2D series: n2d-standard-8 and n2d-standard-16
  - b. N2 series: n2-highmem-8 and n2-highmem-16
2. Setup a separate benchmark driver client running on a GCE VM, where we will install benchmarking tools like HammerDB.
3. Run TPC-C like benchmark using HammerDB, and TPC-B like benchmark using Pgbench.

Unless otherwise specified, we used the following setup for performance benchmarking:

Component	Value
Machine Type	<u>N2D (AMD Milan or later)</u> n2d-standard-8 (8 vCPUs / 32GB) and n2d-standard-16 (16vCPU / 64GB)  Boot disk: 100 GB PD-SSD

	Data disk: 1 x 375GB local SSD  <u>N2 (Intel Icelake or later)</u> n2-highmem-8 (8 vCPU / 64 GB) n2-highmem-16 (16 vCPU / 128GB)  Boot disk: 100 GB PD-SSD Data disk: 4TB PD-SSD
Operating system	Ubuntu 22.04 (Linux kernel 6.5)
AlloyDB Omni Version	15.5.5
Region / Zone	us-central1 (lowa) / us-central1-f
Client VM – Machine Type	n2-standard-32 (32 vCPU/ 128GB) Boot disk: 128 GB, PD-SSD Ubuntu 22.04
Zone of Client VM	us-central1-f [same as AlloyDB Omni instance]
Connectivity	Private IP (same VPC)
Test tools	HammerDB-4.10 Pgbench Psql
Workloads	Benchmarks: TPC-C and TPC-B Database size: Smaller than memory and larger than memory Number of Clients: Low and high TPC-C benchmark and TPC-B benchmark

In your own testing, you can also run AlloyDB Omni on other platform configurations (as long as they meet [these system requirements](#)). Your benchmarking results will vary based on your specific hardware. Some crucial factors that can affect performance include the CPU model, number of cores/vCPUs, available memory, disk performance (IOPS and throughput), and network performance (latency and bandwidth) between the server and client.

AlloyDB Omni is 100% compatible with PostgreSQL. To aid in doing performance comparisons between AlloyDB Omni and PostgreSQL, this document also shows you the small changes that you need to make in order to run the same benchmarks on PostgreSQL. These changes are shown in a box with the PostgreSQL icon as follows:



Instructions for PostgreSQL will show up in a box like this.

# Infrastructure Setup

---

## Provision the server and client VMs

---

**Note:** The next section describes how to provision the VMs through the GCP cloud console. You may skip this section if running on your own hardware.

### Provision Server on GCE

1. Create or select your GCP project: Go to <https://console.cloud.google.com> and select your project from the drop down menu or create a new one.
2. Follow these links on the portal: “Products and Solutions” → “All Products” → “Compute Engine”.
3. Click on the following button to create an instance to run AlloyDB Omni.



4. Choose a name for your server VM, and select your desired region and zone.

A screenshot of the GCP VM instance configuration form. It shows the following fields:

- Name \***: A text input field containing 'omni-server-16vcpu' and a help icon.
- Labels ?**: A section with a '+ ADD LABELS' button.
- Region \***: A dropdown menu showing 'us-central1 (Iowa)' with a help icon. Below it, the text 'Region is permanent' is displayed.
- Zone \***: A dropdown menu showing 'us-central1-a' with a help icon. Below it, the text 'Zone is permanent' is displayed.

5. Under "Machine Configuration", select your desired machine type:
  - a. For N2D: Select "N2D" for "Series", and "n2d-standard-8" or "n2d-standard-16" for the "Machine type".
  - b. For N2: Select "N2" for "Series", and "n2-highmem-8" or "n2-highmem-16" for the "Machine type".

## Machine configuration

General purpose
  Compute optimized
  Memory optimized
  Storage optimized
  NEW
  GPUs

Machine types for common workloads, optimized for cost and flexibility

Series	Description	vCPUs	Memory	Platform
<input type="radio"/> N4	Flexible & cost-optimized	2 - 80	4 - 640 GB	Intel Emerald Rap
<input type="radio"/> C3	Consistently high performance	4 - 102	8 - 1,536 GB	Intel Sapphire Rap
<input type="radio"/> C3D	Consistently high performance	4 - 360	8 - 2,880 GB	AMD Genoa
<input type="radio"/> E2	Low cost, day-to-day computing	0.25 - 32	1 - 128 GB	Based on ava
<input type="radio"/> N2	Balanced price & performance	2 - 128	2 - 864 GB	Intel Cascade
<input checked="" type="radio"/> N2D	Balanced price & performance	2 - 224	2 - 896 GB	AMD EPYC
<input type="radio"/> T2A	Scale-out workloads	4 - 140	4 - 102 GB	AMD EPYC
<input type="radio"/> T2D	Scale-out workloads	4 - 140	4 - 102 GB	AMD EPYC
<input type="radio"/> N1	Balanced price & performance	0.25 - 96	0.6 - 624 GB	Intel Skylake


**Machine type**

Choose a machine type with preset amounts of vCPUs and memory that suit most workloads. Or, you can create a custom machine for your workloads' particular needs. [Learn more](#)

n2d-standard-16 (16 vCPU, 8 core, 64 GB memory)

Memory

64 GB



vCPU

16 (8 cores)

6. For best performance, expand "Advanced Configurations":
  - a. For N2D: Select "AMD Milan or later"
  - b. For N2: Select "Intel Icelake or later"

You may leave the other options empty, as they are not needed.

CPU platform

AMD Milan or later ▼ ?

vCPUs to core ratio ▼ ?

Visible core count ▼ ?

[^ ADVANCED CONFIGURATIONS](#)

7. Under "Boot disk", ensure you are using a Ubuntu 22.04 image, on a "SSD persistent disk", and 100 GB provisioned for the boot disk.

## Boot disk ?

name	omni-server-16vcpu
disk type	new SSD persistent disk
disk size	100 GB
schedule ?	No schedule selected
disk type ?	Free
OS	Ubuntu 22.04 LTS

Save

- Under "Observability - Ops Agent", select "Install Ops Agent for Monitoring and Logging". This agent helps gather system metrics during the benchmark run.

## Observability - Ops Agent ?

Monitor your system through collection of logs and key metrics.

- Install Ops Agent for Monitoring and Logging ←

- Under "Advanced options" -> "Disks":
  - If using local SSDs: click "+ Add Local SSD". Then, under "Disk capacity", select the "375 GB (1 \* 375 GB)" option.

### Disk capacity

You can create multiple Local SSD disks at the same time. Local SSD options may vary by machine type and number of vCPUs selected

Disk capacity \*  
375 GB (1 \* 375 GB) ?

### Properties

Total size	375 GB
Device name ?	local-ssd-0
Encryption type	Google-managed

- If using PD-SSDs: click "+ Add New Disk". Then, under "Disk settings", select "SSD persistent disk" for "Disk type", and enter "4096" for "Size".

Disk settings

Disk type \*  
SSD persistent disk

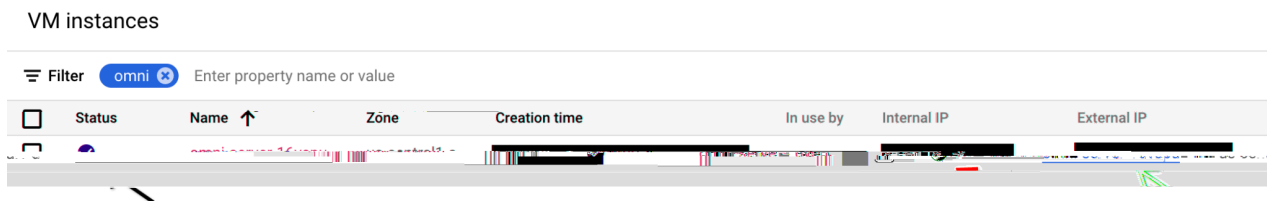
COMPARE DISK TYPES

Size \*  
4096 GB ?  
Provision between 10 and 65,536 GB

Click "Save" at the bottom to confirm your choices.



10. Now click "Create" at the bottom of the create instance page, and a new VM will begin to be provisioned for you. Wait until the VM is fully created, which will be indicated by a green check mark under the "Status" column.



## Provision Client on GCE

To run the OLTP benchmarks, you will require a client machine with enough processing power. We need to provision a client machine that is powerful enough that it is not the bottleneck in a benchmark.

We use an **n2-standard-32** machine (32 vCPUS, 128 GB memory) with 128 GB SSD persistent disk (pd-ssd) as the client in this document. We use Ubuntu 22.04 as the OS for the client.

**Important:** The client must be provisioned in the same region, zone, and VPC as AlloyDB Omni's primary instance. Benchmarking tools directly access the AlloyDB Omni instance over private IP. This setup reduces network latency between the server and client.

Below is a screenshot of the client machine we provisioned for the purpose of this benchmarking guide.

Basic information	
Name	[REDACTED]
Instance Id	[REDACTED]
Description	None
Type	Instance
Status	✓ Running
Creation time	Jul 15, 2024, 9:04:04 AM UTC-07:00
Zone	us-central1-a ← Same zone as Alloy
Instance template	None
In use by	None
Reservations	Automatically choose
Labels	None
Tags ?	[REDACTED]
Deletion protection	Disabled
Confidential VM service ?	Disabled
Preserved state size	0 GB
Machine configuration	
Machine type	n2-standard-32
CPU platform	Intel Cascade Lake
Architecture	x86/64
vCPUs to core ratio	—
Custom visible cores	—
Display device	Disabled Enable to use screen capturing and recording tools
GPU	None
Resource policies	[REDACTED]

## Install AlloyDB Omni

### Install Docker

SSH to the server VM:

```
gcloud compute ssh --zone "<primary zone>" "<server machine name>" --project "<google-project>"
```

AlloyDB Omni is packaged as a Docker image. First, install docker:

```
sudo apt update -y
sudo apt install -y docker.io
sudo usermod -aG docker $USER # This requires a re-login to take effect
```

Exit the SSH session, and re-login. Test that docker is installed by running:

```
docker run --rm hello-world
```

The following message shows Docker is installed successfully and can be run by the current user:

Hello from Docker! This message shows that your installation appears to be working correctly.

Download the AlloyDB Omni image:

```
docker pull google/alloydbomni:15
```



To run PostgreSQL, download its image from [Docker Hub](#) instead:

```
docker pull postgres:15
```

## Create an ext4 filesystem

The disks are attached to the VM as raw devices. To use them, we have to create a filesystem on top of the devices.

By default, the devices are listed as `/dev/nvme0n1` (for local SSDs) and `/dev/sdb` (for PD-SSDs) on the GCE VM.

```
# For local SSD
export DEVICE="/dev/nvme0n1"

# For PD-SSD
export DEVICE="/dev/sdb"
```

Format an `ext4` filesystem on the virtual device:

```
sudo mkfs.ext4 -m 0 -F -E lazy_itable_init=0,lazy_journal_init=0 ${DEVICE}
```

Create a data directory for AlloyDB Omni:

```
mkdir /home/$USER/alloydb-data
```

Mount the filesystem onto this directory:

```
sudo mount --make-shared -o noatime,discard,errors=panic ${DEVICE} /home/$USER/alloydb-data
```

## Set up VM configurations

**Swap:** AlloyDB Omni uses swap to efficiently manage memory on the VM. For optimal performance, you should set swap to roughly 20% of available memory. To enable swap, run:

```
export SWAPSIZE=$(awk '/MemTotal/ { printf int($2 * 0.2 /1024/1024) }' /proc/meminfo)
sudo -E fallocate -l ${SWAPSIZE}G /swapfile
sudo chmod 600 /swapfile
```

```
sudo mkswap /swapfile
sudo swapon /swapfile
```

**Huge pages:** Enable huge pages on the VM by running:

```
sudo docker run --rm --privileged google/alloydbomni:15 setup-host
```



The `setup-host` script is just a convenience script to set up huge pages. You may run the above step, even if you are benchmarking PostgreSQL instead.

Note that `sudo` is required for both the "Swap" and "Huge pages" steps above, as they configure OS level settings.

## Start AlloyDB Omni

Start AlloyDB Omni with the following command (replace `[PASSWORD]` with an appropriate password):

```
export PASSWORD=[PASSWORD]
docker run --detach \
  --name alloydb-omni \
  -e POSTGRES_PASSWORD=${PASSWORD} \
  -e PGDATA=/var/lib/postgresql/data/pgdata \
  -v "/home/$USER/alloydb-data":/var/lib/postgresql/data \
  -v /dev/shm:/dev/shm \
  --ulimit=nice=-20:-20 \
  -p 5432:5432 \
  google/alloydbomni:15
```



To start PostgreSQL instead, replace the last line with `postgres:15`.

You may further customize the installation by following the steps in [Customize your AlloyDB Omni installation](#).

## Allow access from the client VM

First, run the following command to get a shell inside the container:

```
docker exec -it alloydb-omni bash
```

Inside the shell, set an environment variable with your client's internal IP address. This can be found in the GCP console UI:

VM instances									
Filter <input type="text"/> Enter property name or value <span>✕</span>									
<input type="checkbox"/>		Status	Name ↑	Zone	Creation time	Machine type	In use by	Internal IP	External IP
<input type="checkbox"/>			<input type="text"/>	us-central1-a	Jul 15, 2024, 3:32:19 PM	n2-standard-32		10.128.0.94 <a href="#">(nic0)</a>	35.238.33.217 <a href="#">(nic0)</a>

```
export CLIENT_IP=[CLIENT IP]
```

Then run:

```
sed -i '/host all all all scram-sha-256/i host all all '${CLIENT_IP}'/32 trust'
$PGDATA/pg_hba.conf
```

This adds a line near the end of `pg_hba.conf` like this:

```
# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all trust
# IPv4 local connections:
host all all 127.0.0.1/32 trust
# IPv6 local connections:
host all all ::1/128 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all trust
host replication all 127.0.0.1/32 trust
host replication all ::1/128 trust
host all all 1.2.3.4/32 trust # <-- ENTRY FOR CLIENT
host all all all scram-sha-256
```

**NOTE:** In this guide, we use the "trust" setting to simplify the benchmarking setup. However, note that "trust" bypasses password protection, and should not be used for a production instance.

## Update database configuration

Still inside the bash shell, run the following command to set the configurations for the database. Run the following command to append these settings to the end of the file:

```
cat << EOF >> $PGDATA/postgresql.conf
huge_pages=on
max_connections=2000
EOF

# Only required for AlloyDB Omni 15.5.5 and earlier.
# Later versions include these settings out of the box.
lux_wal_writer_batch_size=0
enable_google_adaptive_autovacuum=off
```



If you are setting configurations for PostgreSQL, also add the following settings, which are generic rule-of-thumb settings when running PostgreSQL:

#### PostgreSQL (33% buffer cache)

```
shared_buffers=21436MB
effective_cache_size=25724MB
```

Run `exit` to exit the shell.

## Start AlloyDB Omni

Now we may restart AlloyDB Omni to pick up the updated configurations:

```
docker restart alloydb-omni
```

If you do not see any errors, that means AlloyDB Omni is running. Verify by connecting to the database locally:

```
docker exec -it alloydb-omni psql -h localhost -U postgres -c "SELECT 1"
```

You should see the output:

```
?column?
-----
         1
(1 row)
```

## Setup of Benchmark Driver Machine (Client)

This section will guide you through the steps of configuring the client machine, where we will install benchmarking tools such as HammerDB.

Connect to the client machine using the “`gcloud compute ssh`” command.

Sample command:

```
gcloud compute ssh --zone "<primary zone>" "<client machine name>" --project "<google-project>"
```

### Install PostgreSQL tools

The following commands install `psql` which is used to connect to AlloyDB Omni. It also installs `pgbench` which we will be using for the TPC-B benchmark.

```
# Import the repository signing key
```

```

sudo apt install -y curl ca-certificates
sudo install -d /usr/share/postgresql-common/pgdg
sudo curl -o /usr/share/postgresql-common/pgdg/apt.postgresql.org.asc --fail
https://www.postgresql.org/media/keys/ACCC4CF8.asc

# Create the repository configuration file
sudo sh -c 'echo "deb [signed-by=/usr/share/postgresql-common/pgdg/apt.postgresql.org.asc]
https://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" >
/etc/apt/sources.list.d/pgdg.list'

# Update the package lists
sudo apt update -y

# Install tools for PostgreSQL 15
sudo apt -y install postgresql-15

# By default, a Postgres database is launched. But we don't need that here.
sudo systemctl stop postgresql
sudo systemctl disable postgresql

```

Now ensure that it works and you are able to connect to the AlloyDB Omni. Use the “Private IP” address of your AlloyDB Omni instance.

```

export SERVER_IP=[Private IP of AlloyDB Omni instance]
psql -h ${SERVER_IP} -U postgres -c "SELECT 1"

```

## Install Docker

For this benchmarking guide, we will use the HammerDB test driver. HammerDB is packaged via Docker, so we will install Docker on the client machine like above.

First, install docker:

```

sudo apt install -y docker.io
sudo usermod -aG docker $USER # This requires a re-login to take effect

```

Exit the SSH session, and re-login. Test that docker is installed by running:

```

docker run --rm hello-world

```

The following message shows Docker is installed successfully and can be run by the current user:

Hello from Docker! This message shows that your installation appears to be working correctly.

Run the following command to download the HammerDB image:

```

docker pull tpcorg/hammerdb:postgres

```

## TPC-C Benchmark

[HammerDB](#) is a popular benchmarking tool that includes a [TPC-C](#) benchmark implementation for evaluating the performance of OLTP systems. HammerDB's TPC-C implementation allows users to simulate a workload similar to the TPC-C benchmark, including a mix of transactions that mimic the behavior of a wholesale supplier environment. HammerDB measures the system's performance in terms of transactions per minute (TPM) and generates reports that include detailed statistics and performance metrics. Additionally, HammerDB supports customization of the benchmark parameters, allowing users to adjust the database size, the number of warehouses, and other workload characteristics to simulate different scenarios.

This section provides a comprehensive guide on how to execute the HammerDB TPC-C benchmark to gauge the performance of the AlloyDB Omni database system.

### Benchmark configurations

These tables show the benchmark configurations that we used, and the results we obtained in our internal runs. Please note that the results may fluctuate across different runs. Nevertheless, your results should generally align with the findings we have achieved.

On n2d-standard instances:

DataSet size	#vCPUs / RAM (GB)	#of Warehouses	# of Virtual Users	Throughput (TPM)
Small (< RAM)	8 vCPUs, 32GB	144	8	204139
	8 vCPUs, 32GB	144	128	299822
	16 vCPUs, 64GB	288	16	195353
	16 vCPUs, 64GB	288	256	436412
Big (> RAM)	8 vCPUs, 32GB	800	8	125777
	8 vCPUs, 32GB	800	128	190117
	16 vCPUs, 64GB	1600	16	121513
	16 vCPUs, 64GB	1600	256	220611

On n2-highmem instances:

DataSet size	#vCPUs / RAM (GB)	#of Warehouses	# of Virtual Users	Throughput (TPM)
Small (< RAM)	8 vCPUs, 64GB	288	8	146491
	8 vCPUs, 64GB	288	128	356427
	16 vCPUs, 128GB	576	16	210814



	16 vCPUs, 128GB	576	256	903690
Big (> RAM)	8 vCPUs, 64GB	1600	8	66821
	8 vCPUs, 64GB	1600	128	224327
	16 vCPUs, 128GB	3200	16	109398
	16 vCPUs, 128GB	3200	256	456404

## Prerequisites

- A. You need to run the following steps from a client (driver) machine. Ensure that you have completed the setup steps listed in the "[Setup of Benchmark Driver Machine \(Client\)](#)" section (especially installation of the HammerDB utility).
- B. **Cleanup:** If you are running multiple benchmarks in succession, ensure you follow the "[Benchmark Cleanup](#)" section before doing your subsequent run.

## Initial Setup on Client Machine

Create a `hammerdb/` directory for HammerDB configuration scripts.

```
mkdir hammerdb
cd hammerdb
```

Then create `setup.env` file by running the following:

```
export SERVER_IP=1.2.3.4 # Private IP of the AlloyDB primary instance
export PGPORT=5432 # Postgres default port address. You do not need to change it unless
you use non-default port address.
export NUM_WAREHOUSE=576 # Number of TPC-C warehouses to load. This determines the overall
database size.
export NUM_USERS=256 # Number of users for running the benchmark.

cat << EOF > setup.env
PGHOST=${SERVER_IP}
PGPORT=${PGPORT}
NUM_WAREHOUSE=${NUM_WAREHOUSE}
NUM_USERS=${NUM_USERS}
EOF
```

Edit the generated `setup.env` file and change all the **highlighted** parameter values to those that are suitable to your test setup.

## Script to load TPC-C data

---

In the context of the TPC-C benchmark, a "load step" refers to the process of populating the benchmark database with initial data before running the actual performance test.

During this step, the database is populated with a specified number of warehouses, customers, and other entities according to the TPC-C specifications. The purpose of the load step is to create a realistic workload for the performance test, and to ensure that the test results are comparable across different systems.

After the load step is completed, the database is pre-populated with a defined set of initial data, and ready to be used for the TPC-C benchmark test.

Follow the steps below to load the TPC-C database:

1. Create **build-tpcc.tcl** file by running the following:

```
cat << EOF > build-tpcc.tcl
# CONFIGURE PARAMETERS FOR TPCC BENCHMARK
# -----
dbset db pg
dbset bm tpc-c

# CONFIGURE POSTGRES HOST AND PORT
# -----
diset connection pg_host $::env(PGHOST)
diset connection pg_port $::env(PGPORT)

# CONFIGURE TPCC
# -----
diset tpcc pg_superuser postgres
diset tpcc pg_user tpcc
diset tpcc pg_dbase tpcc

# SET NUMBER OF WAREHOUSES AND USERS TO MANAGE EACH WAREHOUSE
# THIS IMPORTANT METRIC ESTABLISHES THE DATABASE SCALE/SIZE
# -----
diset tpcc pg_count_ware $::env(NUM_WAREHOUSE)
diset tpcc pg_num_vu 10

# LOG OUTPUT AND CONFIGURATION DETAILS
# -----
vuset logtotemp 1
print dict

# CREATE AND POPULATE DATABASE SCHEMA
# -----
buildschema

vudestroy
```

```
quit
```

```
EOF
```

2. Execute the load command as shown below and wait for the command to finish. During this command, you may run `docker logs -f build-tpcc` to follow its progress.

```
docker run \  
  --detach \  
  --name build-tpcc \  
  --env-file setup.env \  
  -v $PWD/build-tpcc.tcl:/build-tpcc.tcl \  
  tpcorg/hammerdb:postgres ./hammerdbcli auto /build-tpcc.tcl
```

## Running the TPC-C benchmark

---

In this step, we will initiate the actual TPC-C performance test. The TPC-C benchmark will be executed using the populated database from the load step. The benchmark generates a series of transactions that simulate a typical business environment, including order entry, payment processing, and inventory management. The workload is measured in "transactions per minute" (**TPM**), which represents the number of complete business transactions that the system can handle in one minute.

The run step is designed to stress the database system under realistic conditions and provide a standard way of measuring performance that can be compared across different database systems. Vendors and customers widely use the results of the TPC-C benchmark to evaluate the performance of different database systems and hardware configurations.

The following script will run the TPC-C benchmark for about 1 hour after approximately 10 minutes of warm up.

1. Create `run-tpcc.tcl` script by running the following:

```
cat << EOF > run-tpcc.tcl  
dbset db pg  
dbset bm tpc-c  
  
# CONFIGURE PG HOST and PORT  
# -----  
diset connection pg_host $::env(PGHOST)  
diset connection pg_port $::env(PGPORT)  
  
# CONFIGURE TPCC DB  
# -----  
diset tpcc pg_superuser postgres  
diset tpcc pg_user postgres
```

```

diset tpcc pg_dbase tpcc

# BENCHMARKING PARAMETERS
# -----
diset tpcc pg_driver timed
diset tpcc pg_rampup 10
diset tpcc pg_duration 60
diset tpcc pg_vacuum false
diset tpcc pg_partition false
diset tpcc pg_allwarehouse true
diset tpcc pg_timeprofile true
diset tpcc pg_connect_pool false
diset tpcc pg_dritasnap false
diset tpcc pg_count_ware $::env(NUM_WAREHOUSE)
diset tpcc pg_num_vu 1

loadscript
print dict
vuset logtotemp 1
vuset vu $::env(NUM_USERS)
vucreate
vurun
quit
EOF

```

2. Run the script as follows:

```

docker run \
  --detach \
  --name run-tpcc \
  --env-file setup.env \
  -v $PWD/run-tpcc.tcl:/run-tpcc.tcl \
  tpcorg/hammerdb:postgres ./hammerdbcli auto /run-tpcc.tcl

```

Similar to before, you may run `docker logs -f run-tpcc` to follow the progress.

3. Now wait for the `run-tpcc.sh` script to finish. The script will take approximately 1.5 hours to complete (pg\_rampup = 10mins, pg\_duration = 60mins, and time spent initializing and terminating the HammerDB workers).

## Analyzing TPC-C Results

---

In the context of the TPC-C benchmark, **NOPM** and **TPM** are performance metrics used to measure the performance of a database system. **NOPM** stands for "New Orders Per Minute" and measures the number of new order transactions that the system can handle in one minute. The New Order transaction is one of the most important transactions in the TPC-C benchmark and involves creating a new order for a customer.

**TPM** stands for "Transactions Per Minute" and measures the total number of completed business transactions that the system can handle in one minute. This includes not only **New Order** transactions but also **Payment**, **Delivery**, **Order Status**, and other types of transactions defined in the TPC-C benchmark.

In general, TPM is considered to be the primary performance metric for the TPC-C benchmark, as it provides an overall measure of the system's ability to handle a realistic workload. However, NOPM can also be a useful metric for systems that are heavily focused on processing new orders, such as e-commerce or retail systems.

At the end of the run, you can extract the performance results from the logs of `run-tpcc`:

```
$ docker logs run-tpcc |& grep NOPM
Uuser 1:TEST RESULT : System achieved 95033 NOPM from 218572 PostgreSQL TPM
```

From the sample output, we can see that the performance is 218572 TPM.

## TPC-B Benchmark

[pgbench](#) is a simple program, provided by PostgreSQL, for running benchmark tests. It is simpler to run, and hence more convenient. However, in our experience, the workload generated from `pgbench` can be less representative of real-world usage.

### Benchmark configurations

These tables show the benchmark configurations that we used, and the results we obtained in our internal runs. Please note that the results may fluctuate across different runs. Nevertheless, your results should generally align with the findings we have achieved.

On n2d-standard instances:

DataSet size	#vCPUs / RAM (GB)	Scale factor	Clients	Throughput (TPS)
Small (< RAM)	8 vCPUs, 32GB	800	8	4886
	8 vCPUs, 32GB	800	128	13259
	16 vCPUs, 64GB	1600	16	8993
	16 vCPUs, 64GB	1600	256	21442
Big (> RAM)	8 vCPUs, 32GB	8500	8	3632
	8 vCPUs, 32GB	8500	128	10208
	16 vCPUs, 64GB	17000	16	6611
	16 vCPUs, 64GB	17000	256	12539

On n2-highmem instances:

DataSet size	#vCPUs / RAM (GB)	Scale factor	Clients	Throughput (TPS)
Small (< RAM)	8 vCPUs, 64GB	1600	8	5339
	8 vCPUs, 64GB	1600	128	12940
	16 vCPUs, 128GB	3200	16	9507
	16 vCPUs, 128GB	3200	256	22637
Big (> RAM)	8 vCPUs, 64GB	17000	8	2408
	8 vCPUs, 64GB	17000	128	3383
	16 vCPUs, 128GB	34000	16	3012
	16 vCPUs, 128GB	34000	256	8410

### Load data

To load the initial dataset, you pick a "scale factor". The larger the scale factor, the larger the database size.

First, specify some parameters:

```
export SERVER_IP=1.2.3.4 # Private IP of the AlloyDB primary instance
export SCALE_FACTOR=100 # Scale factor for pgbench
```

Then, to load the initial dataset:

```
pgbench -h ${SERVER_IP} -U postgres postgres --initialize --scale=${SCALE_FACTOR}
```

### Run TPC-B

We first run pgbench for 10 mins (warm-up phase), then 1 hour (actual run phase). The results from the run phase will be used.

First, specify some parameters:

```
export SERVER_IP=1.2.3.4 # Private IP of the AlloyDB primary instance
export CLIENTS=16 # Number of parallel clients
```

Warm-up phase:

```
pgbench -h ${SERVER_IP} -U postgres postgres --time=600 --protocol=prepared --client=${CLIENTS}
--jobs=${CLIENTS}
```

Actual run phase:

```
pgbench -h ${SERVER_IP} -U postgres postgres --time=3600 --protocol=prepared --client=${CLIENTS}
```

```
--jobs=${CLIENTS}
```

At the end of the run, pgbench will print an output like:

```
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 8500
query mode: prepared
number of clients: 64
number of threads: 64
duration: 3600 s
number of transactions actually processed: 34028870
latency average = 6.770 ms
latency stddev = 17.611 ms
tps = 9452.335211 (including connections establishing)
tps = 9452.470670 (excluding connections establishing)
```

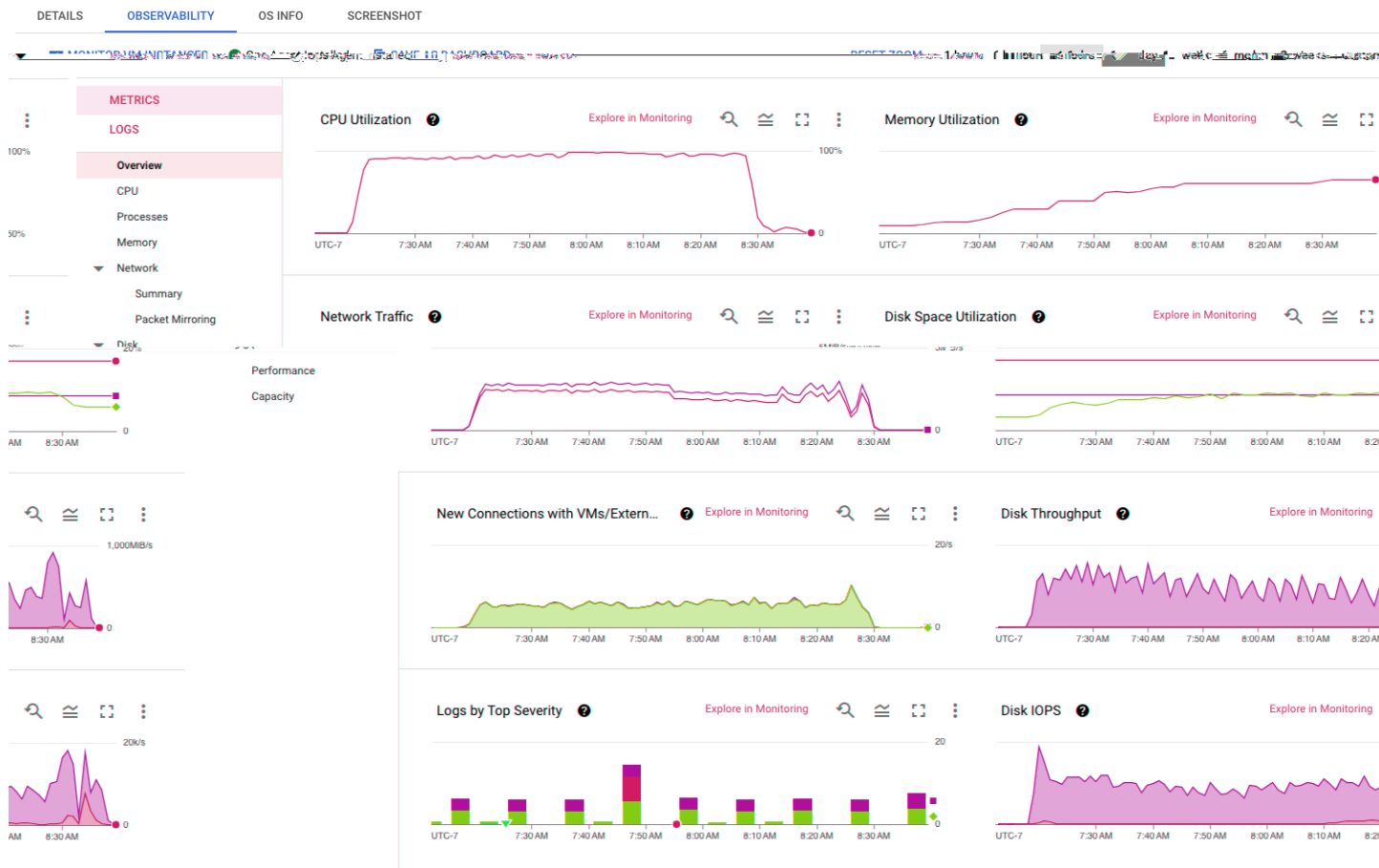
The final line (`tps = ...`) reports the TPS (transactions per second) that pgbench was able to perform on the database.

## Appendix 1: Observability

---

To further understand the behavior of the database system, you can use the GCE monitoring page to monitor important system metrics, such as CPU usage, memory usage, etc. This monitoring information can be found by navigating to the "Compute Engine -> VM instances -> Instance" page and/or navigating to the **Observability** page on <https://console.cloud.google.com>.

For instance, the below picture shows the CPU/Memory/Disk/Network metrics of a GCE instance during the TPC-C run.



If you are running AlloyDB Omni on other hardware, you can use the `iostat` program to check real time CPU/IO stats. (If you get an error `iostat: command not found`, install the program with `sudo apt install -y sysstat`.)

```
iostat -m 10
```

This will print statistics about the I/O devices every 10 seconds, e.g.:

```
$ iostat -m 10
Linux 6.1.0-21-cloud-amd64 (omni-server-16vcpu)          06/26/24          _x86_64_          (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.17    0.57   0.26   0.18    0.00   98.82

Device            tps    MB_read/s    MB_wrtn/s    MB_dscd/s    MB_read    MB_wrtn    MB_dscd
sda                805.89         0.08         21.04         1.50         517       138491       9895
...
```

For details about the output of `iostat`, please refer to its [documentation](#).



## Appendix 2: Notes on performance benchmarking

---

### *Benchmark Cleanup*

---

This step is important if you are planning to execute multiple benchmarks in succession. Performing a proper cleanup between each benchmark is a critical prerequisite for accurate and reliable benchmarking results. This includes deleting previous benchmark data (i.e. benchmark database), and rebooting the AlloyDB Omni instance (that clears caches at database and operating systems level) before running another benchmark. A proper benchmark cleanup ensures that residual effects from previous benchmarks do not affect the performance measurements of the new benchmark. It also helps to ensure consistency and repeatability of the benchmark results, which is essential for making meaningful comparisons between different systems or identifying areas for optimization in hardware, software, or configuration.

Follow the URL <https://cloud.google.com/compute/docs/instances/stop-start-instance> to learn more about how to reboot a GCE VM.

To drop the previous benchmark database, you can use the following psql command from the client machine.

```
export SERVER_IP=[Private IP of AlloyDB Omni instance]
psql -h $SERVER_IP -U postgres -c "DROP DATABASE IF EXISTS [database_name];"
```

You may also need to remove docker containers left behind from earlier runs:

```
docker container prune
```

### *Understanding system performance*

---

Since AlloyDB Omni can be run on many different environments, it is important to know that the transaction performance is highly dependent on CPU/Memory/IO/Network latency.

1. When most data fits in memory, it is a CPU bound workload, and more CPUs will get more transaction performance.
2. When most data can not fit in memory, it becomes an IO bound workload, more disk IOPS/throughput will get more transaction performance. IO latency is also important for OLTP workload, when a transaction commits, it needs to flush WAL to disk before commit, so IO latency is directly related to commit latency.
3. Query latency is affected by network latency between client and server communication. It is recommended to have the client and server located in the same local network or same zone for benchmarking purposes.

Before benchmarking, It is useful to be able to characterize system performance of the hardware. In this section, we list down some commands that can be used to measure:

1. Performance of the CPU
2. Performance of the disk
3. Network latency between client and server

## CPU performance

CPU performance can be measured by sysbench benchmark, see <https://github.com/akopytov/sysbench> for installation instructions.

Use the following command to measure cpu performance:

```
sysbench cpu --cpu-max-prime=10000 --threads=[Number of vCPUs] run
```

## Disk performance

Fio can be used to measure disk performance.

Use the following commands to measure IOPS, throughput and latency.

IOPS

```
fio --time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 --name=iops_test  
--filename=/mnt/disks/pgsql/fio_test --bs=8k --iodepth=256 --size=4G --readwrite=randrw  
--rwmixread=25 --verify=0 --group_reporting=1
```

Write Throughput

```
fio --name=write_throughput --filename=/mnt/disks/pgsql/fio_test --numjobs=16 --size=4G  
--time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 --verify=0 --bs=256k  
--iodepth=256 --rw=randwrite --group_reporting=1
```

Latency

```
fio --time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 --name=latency_test  
--filename=/mnt/disks/pgsql/fio_test --bs=256k --iodepth=1 --size=4G --readwrite=randwrite  
--verify=0
```

## Network latency

Ping can be used to measure network latency.

```
ping [IP address] -c 100
```