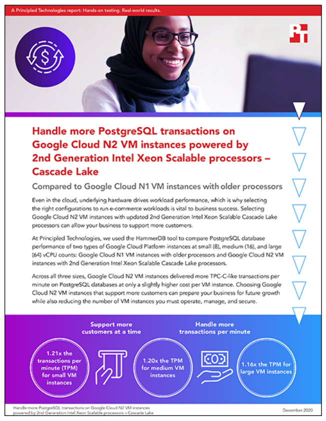
A Principled Technologies report: Hands-on testing. Real-world results.

Handle more PostgreSQL transactions on Google Cloud N2 VM instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake

Compared to Google Cloud N1 VM instances with older processors

Even in the cloud, underlying hardware drives workload performance, which is why selecting the right configurations to run e-commerce workloads is vital to business success. Selecting Google Cloud N2 VM instances with updated 2nd Generation Intel Xeon Scalable Cascade Lake processors can allow your business to support more customers.

At Principled Technologies, we used the HammerDB tool to compare PostgreSQL database performance of two types of Google Cloud Platform instances at small (8), medium (16), and large (64) vCPU counts: Google Cloud N1 VM instances with older processors and Google Cloud N2 VM instances with 2nd Generation Intel Xeon Scalable Cascade Lake processors.

Across all three sizes, Google Cloud N2 VM instances delivered more TPC-C-like transactions per minute on PostgreSQL databases at only a slightly higher cost per VM instance. Choosing Google Cloud N2 VM instances that support more customers can prepare your business for future growth while also reducing the number of VM instances you must operate, manage, and secure.

Support more customers at a time

Handle more transactions per minute

1.21x the transactions per minute (TPM) for small VM instances

1.20x the TPM for medium VM instances

1.16x the TPM for large VM instances

Handle more PostgreSQL transactions on Google Cloud N2 VM instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake

December 2020

The science behind the report:

Handle more PostgreSQL transactions on Google Cloud N2 VM instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake

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 Handle more PostgreSQL transactions on Google Cloud N2 VM instances powered by 2nd Generation Intel Xeon Scalable processors – Cascade Lake.

We concluded our hands-on testing on October 28, 2020. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on October 26, 2020 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

Table 1: Small VM instance results. Source: Principled Technologies.

n-standard-8 series (85WH/24Users)			
	n1	n2	n2 advantage
Transactions per minute	336,987	406,414	1.21x
New orders per minute	146,491	176,670	1.21x

Table 2: Medium VM instance results. Source: Principled Technologies.

n-standard-16 series (240WH/32Users)			
	n1	n2	n2 advantage
Transactions per minute	623,568	747,191	1.20x
New orders per minute	271,163	324,923	1.20x

Table 3: Large VM instance results. Source: Principled Technologies.

n-standard-64 series (600WH/64Users)			
	n1	n2	n2 advantage
Transactions per minute	1,605,344	1,864,632	1.16x
New orders per minute	697,985	810,631	1.16x

System configuration information

Table 4: Detailed information on the n2 VM instances we tested.

System configuration information	n2-standard-8	n2-standard-16	n2-standard-64
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	10/28/2020	10/28/2020	10/28/2020
CSP/Region	us-east1-b	us-east1-b	us-east1-b
Workload & version	HammerDB v3.3 TPC-C-like	HammerDB v3.3 TPC-C-like	HammerDB v3.3 TPC-C-like
WL specific parameters	85 warehouses vm.nr_hugepages = 12288	240 warehouses vm.nr_hugepages = 28672	600 warehouses vm.nr_hugepages = 102400
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	n2-standard-8	n2-standard-16	n2-standard-64
BIOS name and version	Google Google, 01/01/2011	Google Google, 01/01/2011	Google Google, 01/01/2011
Operating system name and version/build number	CentOS 8.2	CentOS 8.2	CentOS 8.2
Date of last OS updates/ patches applied	10/26/2020	10/26/2020	10/26/2020
Processor			
Number of processors	1	1	2
Vendor and model	Intel® Xeon® Platinum 82XXCL	Intel Xeon Platinum 82XXCL	Intel Xeon Platinum 82XXCL
Core count (per processor)	4	8	16
Core frequency (GHz)	2.80	2.80	2.80
Stepping	7	7	7
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64
Memory module(s)			
Total memory in system (GB)	32	64	256
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	32	64	256
General hardware			
Storage: Network or direct-attached	Network attached	Network attached	Network attached
Network bandwidth per VM instance	16 Gbps	32 Gbps	32 Gbps
Storage bandwidth per VM instance	400 MB/s	400 MB/s	400 MB/s

System configuration information	n2-standard-8	n2-standard-16	n2-standard-64
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	500	500	500
Drive information (speed, interface, type)	Standard Persistent Disk	Standard Persistent Disk	Standard Persistent Disk
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	500	500	500
Drive information (speed, interface, type)	SSD Persistent Disk	SSD Persistent Disk	SSD Persistent Disk
Local storage (log drive)			
Number of drives	1	1	1
Drive size (GB)	500	500	1,000
Drive information (speed, interface, type)	SSD Persistent Disk	SSD Persistent Disk	SSD Persistent Disk
Network adapter			
Vendor and model	Google VirtIO Ethernet Adapter	Google VirtIO Ethernet Adapter	Google VirtIO Ethernet Adapter
Number and type of ports	1x 100Gb	1x 100Gb	1x 100Gb

Table 5: Detailed information on the n1 VM instances we tested.

System configuration information	n1-standard-8	n1-standard-16	n1-standard-64
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	10/28/2020	10/28/2020	10/28/2020
CSP/Region	us-east1-b	us-east1-b	us-east1-b
Workload & version	HammerDB v3.3 TPC-C-like	HammerDB v3.3 TPC-C-like	HammerDB v3.3 TPC-C-like
WL specific parameters	85 warehouses vm.nr_hugepages = 12288	240 warehouses vm.nr_hugepages = 28672	600 warehouses vm.nr_hugepages = 102400
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	n1-standard-8	n1-standard-16	n1-standard-64
BIOS name and version	Google Google, 01/01/2011	Google Google, 01/01/2011	Google Google, 01/01/2011
Operating system name and version/build number	CentOS 8.2	CentOS 8.2	CentOS 8.2
Date of last OS updates/ patches applied	10/26/2020	10/26/2020	10/26/2020

System configuration information	n1-standard-8	n1-standard-16	n1-standard-64
Processor			
Number of processors	1	1	1
Vendor and model	N/A	N/A	N/A
Core count (per processor)	4	8	32
Core frequency (GHz)	2.20	2.20	2.20
Stepping	0	0	0
Hyper-Threading	Yes	Yes	Yes
Turbo	Yes	Yes	Yes
Number of vCPU per VM	8	16	64
Memory module(s)			
Total memory in system (GB)	30	60	240
NVMe memory present?	No	No	No
Total memory (DDR+NVMe RAM)	32	64	256
General hardware			
Storage: Network or direct-attached	Network attached	Network attached	Network attached
Network bandwidth per VM instance	16 Gbps	16 Gbps	16 Gbps
Storage bandwidth per VM instance	400 MB/s	400 MB/s	400 MB/s
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	500	500	500
Drive information (speed, interface, type)	Standard Persistent Disk	Standard Persistent Disk	Standard Persistent Disk
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	500	500	500
Drive information (speed, interface, type)	SSD Persistent Disk	SSD Persistent Disk	SSD Persistent Disk
Local storage (log drive)			
Number of drives	1	1	1
Drive size (GB)	500	500	1,000
Drive information (speed, interface, type)	SSD Persistent Disk	SSD Persistent Disk	SSD Persistent Disk
Network adapter			
Vendor and model	Google VirtIO Ethernet Adapter	Google VirtIO Ethernet Adapter	Google VirtIO Ethernet Adapter
Number and type of ports	1x 100Gb	1x 100Gb	1x 100Gb

How we tested

Testing overview

For this project, we tested Google Cloud VM instances featuring older Intel E5_v4 processors vs. newer 2nd Generation Xeon processors. We ran a TPC-C-like workload on PostgreSQL on the GCP VM instances to show the performance increase in terms of transactions per minute on OLTP databases that customers can expect to see using the newer instance series vs. the older.

Using our methodology to aid your own deployments

While the methodology below describes in great detail how we accomplished our testing, it is not a deployment guide. However, because we include many basic installation steps for operating systems and testing tools, reading our testing methodology may help with your own installation.

Creating the Centos 8 baseline image

This section contains the steps we took to create our baseline image.

Creating the PostgreSQL baseline image VM instance

1. Log into Google Cloud and click on Go to console.
2. Click Compute engine, then click VM instances.
3. Click Create.
4. In the left window, select New VM instance.
5. Add the following information:
    a. Name: Name your VM
    b. Labels: projectname:default study:1
    c. Region: us-east1 (South Carolina)
    d. Zone: us-east1-b
    e. Machine Configuration:
    f. Machine family: General-purpose
    g. Series: E2
    h. Machine type: e2-micro
    i. CPU platform: Automatic
    j. Keep Turn on display device unchecked
    k. Keep Confidential VM Service and Container unchecked
6. Boot Disk, click Change.
    a. Operating System: CentOS
    b. Version: CentOS 8
    c. Boot disk type: Standard persistent disk
    d. Size: 50GB
7. Click Select
    a. Identity and API access: App Engine default service account.
    b. Firewall: Check Allow HTTP traffic and Allow HTTPs traffic.
8. Click Create.

Configuring Centos 8 and installing PostgreSQL

1. Login to the postgresql instance via ssh.
2. Disable SELINUX:

```
sudo sed -I 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

3. Update Centos.

```
sudo dnf update
```

4. Disable transparent hugepages persistently by adding "transparent_hugepage=never" to the end of the GRUB_CMDLINE_LINUX option in /etc/default/grub and running the following command to rebuild the grub.cfg file:

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

5. Reboot
6. Create XFS filesystems on the data and log volumes:

```
sudo mkfs.xfs /dev/<disk1>
sudo mkfs.xfs /dev/<disk2>
```

7. Create a postgres user:

```
sudo useradd postgres
```

8. Create a password for the postgres user:

```
sudo passwd postgres
```

9. Download and install the postgres repository:

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-
repo-latest.noarch.rpm
```

10. Disable the existing postgres module that comes with the kernel build:

```
sudo dnf -qy module disable postgresql
```

11. Install Postgresql 13:

```
sudo dnf install -y postgresql13-server
```

12. Create a directory for the log volume.

```
sudo mkdir /var/lib/pgsql/13/log
```

13. Mount the data and log volumes to the data and log directories:

```
sudo mount -t xfs -O defaults,noatime /dev/<disk1> /var/lib/pgsql/13/data
sudo mount -t xfs -O defaults,noatime /dev/<disk2> /var/lib/pgsql/13/log
```

14. Add the mount entries to /etc/fstab.
15. Change the ownership of the postgres user home folder and the postgres database location.

```
sudo chown -R postgres:postgres /home/postgres
sudo chown -R postgres:postgres /var/lib/pgsql
```

16. Switch to the postgres user:

```
su - postgres
```

17. Edit the .bash_profile file to have the following:

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
# User specific environment and startup programs
[ -f /etc/profile ] && source /etc/profile
PGDATA=/var/lib/pgsql/13/data
export PGDATA
PATH=$PATH:$HOME:/usr/bin:/usr/local/bin:/usr/pgsql-13/bin
export PATH
LD_LIBRARY_PATH=/usr/pgsql-13/lib
export LD_LIBRARY_PATH
```

18. Set the environment variables:

```
source .bash_profile
```

19. Initialize the database:

```
initdb -D $PGDATA
```

20. Exit to the user with root privileges.
21. Shut down the instance:

```
sudo poweroff
```

Configuring Centos 8 and installing HammerDB 3.3

1. Log into the HammerDB instance via ssh.
2. Disable SELINUX:

```
sudo sed -I 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

3. Update Centos:

```
sudo dnf update
```

4. Add postgres user:

```
sudo useradd postgres
```

5. Create password for postgres user:

```
sudo passwd postgres
```

6. Download and install the PostgreSQL repository:

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-
repo-latest.noarch.rpm
```

7. Disable the PostgreSQL module that comes with the kernel build:

```
sudo dnf -qy module disable postgresql
```

8. Install the PostgreSQL 13 client:

```
sudo dnf install postgresql13
```

9. Download HammerDB 3.3:

```
sudo wget https://github.com/TPC-Council/HammerDB/releases/download/v3.3/HammerDB-3.3-Linux.tar.gz
```

10. Move the HammerDB package to the postgres user home folder:

```
sudo mv sudo mv HammerDB-3.3-Linux.tar.gz /home/postgres/
```

11. Change the ownership on the postgres home folder:

```
sudo chown -R postgres:postgres /home/postgres
```

12. Switch to the postgres user:

```
su - postgres
```

13. Untar the HammerDB package:

```
tar xfv HammerDB-3.3-Linux.tar.gz
```

14. Add the LD Library Path to the .bash_profile:

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
# User specific environment and startup programs
LD_LIBRARY_PATH=/usr/pgsql-13/lib
export LD_LIBRARY_PATH
```

15. Set the environment variables:

```
source .bash_profile
```

16. Exit to the user with root privileges.
17. Shut down the instance:

```
sudo poweroff
```

Creating a snapshot of your baseline VM

1. Log into Google Cloud and click on Go to console.
2. Click on Compute engine, then click Snapshots
3. Click the Create snapshot button at the top of the page.
4. Enter a snapshot Name.
5. Optionally, enter a Description of the snapshot.
6. Select the Source disk from the drop-down menu.
7. Determine your snapshot storage location.
8. Under Location, select whether you want to store your snapshot in a Multi-regional location or a Regional location. We chose Regional.
9. Select which specific region or multi-region that you want to use. To use the region or multi-region that is closest to your source disk, select Based on disk's location (default). We chose us-east1.
10. Add labels (projectname:default study:1)
11. Leave everything else default.
12. Click Create to create the snapshot.

Creating your image with the baseline snapshot

1. Log into Google Cloud and click on Go to console.
2. Click on Compute engine, then click Images.
3. Click the Create image button at the top of the page.
4. Specify the Name of your image.
5. Specify the Source from which you want to create an image. In our case, we used the snapshot created in the previous step.
6. Specify the Location at which to store your image. We chose us-east1.
7. Specify a family if desired.
8. Enter a description if desired.
9. Add labels (projectname:default study:1)
10. Leave default encryption choice.
11. Click Create to create the image.

Creating a standalone boot disk from the custom image

1. Log into Google Cloud and click on Go to console.
2. Click on Compute engine, then click Disks.
3. Name the disk.
4. Under Type, select Standard persistent disk.
5. Select your region and zone.
6. Choose No schedule under Snapshot schedule.
7. Choose Image as your Source type.
8. Choose the image created in the previous step as your Source image.
9. Choose a size for the boot disk. Note that estimated performance improves with increasing disk size.
10. Leave encryption as Google-managed key.
11. Add your project labels.
12. Click Create.

Creating the VMs under test

To create an instance, you must first have a template. The steps below will walk you through the creation of an instance from a template.

1. Log into Google Cloud and click on Go to console.
2. Click on Compute engine, then click VM instances.
3. Click the Create instance button at the top of the page.
4. In the left window, select New VM instance.
5. Add the following information:
    a. Name: Name your VM.
    b. Labels: projectname:default study:1
    c. Region: us-east1 (South Carolina)
    d. Zone: us-east1-b
    e. Machine Configuration:
    f. Machine family: General-purpose
    g. Series: N1
    h. Machine type: n1-standard-16
    i. CPU platform: Intel Broadwell or higher
    j. Keep Turn on display device unchecked.
    k. Keep Confidential VM Service and Container unchecked.
    l. Boot Disk, click Change.
        - Select the Existing disks tab
        - Choose the disk you created in the previous step and click Select
    m. Identity and API access: App Engine default service account.
    n. Firewall: Check Allow HTTP traffic and Allow HTTPs traffic.
    o. Click Management, security, disks, networking, sole tenancy
        - Click the Disks tab
        - Click Add new disk
        - Optionally, enter a name and description for the disk
        - Choose the disk type. We chose Standard persistent disk.
        - Choose a size in GB.
        - Click Add new disk
        - Optionally, enter a name and description for the disk
        - Choose the disk type. We chose SSD persistent disk.
        - Choose a size in GB
        - Click Add new disk
        - Optionally, enter a name and description for the disk
        - Choose the disk type. We chose SSD persistent disk.
        - Choose a size in GB
        - Leave the rest default and click Done.
    p. Click Create.

Configuring PostgreSQL on the VMs under test

In this section, we list the various PostgreSQL settings that we changed and the steps to do so.

Setting the number of Huge Pages

1. Log into the postgres instance you're working with via ssh.
2. Edit the /etc/sysctl.conf file and add the following line to set the number of hugepages.

```
vm.nr_hugepages = <hugepage_size>
```

3. Reboot

Configuring and starting the database

1. Log into the PostgreSQL instance via ssh.
2. Switch to the postgres user.

```
su - postgres
```

3. Edit the pg_hba.conf file in $PGDATA to add hammerdb client connection info.

```
host all all <IP_ADDRESS>/<PREFIX> trust
```

4. Edit the postgresql.conf file in $PGDATA.
5. Start the database.

```
pg_ctl -D $PGDATA start
```

6. Log into psql.

```
psql
```

7. Change the default password.

```
\password
```

8. Log out of psql.

```
\q
```

Creating the database schema with HammerDB

1. Log into the HammerDB instance via ssh.
2. Switch to the postgres user.

```
su - postgres
```

3. Navigate to the HammerDB directory.

```
cd HammerDB-3.3
```

4. Start hammerdbcli.

```
./hammerdbcli
```

5. Set the following variables.

```
dbset db pg
dbset bm tpc-c
diset connection pg_host <IP_ADDRESS>
diset tpcc pg_count_ware <DB_SIZE>
diset tpcc pg_num_vu 8
diset tpcc pg_superuserpass <Password>
```

6. Build the schema

```
buildschema
```

Backing up the database

1. Log into the PostgreSQL instance.
2. Switch to the postgres user.

```
su - postgres
```

3. Shutdown the database.

```
pg_ctl -D $PGDATA stop
```

4. Copy the database directory to the backup directory.

```
cp -R $PGDATA /var/lib/pgsql/13/backups
```

Moving the write ahead log to the log volume and starting the database

1. Move the WAL to the log volume.

```
mv $PGDATA/pg_wal /var/lib/pgsql/13/log
```

2. Create a symbolic link pointing to the WAL.

```
ln -s /var/lib/pgsql/13/log/pg_wal $PGDATA/pg_wal
```

3. Start the database.

```
pg_ctl -D $PGDATA start
```

Running the tests

In this section, we list the steps to run the HammerDB TPC-C-like test on the VMs under test.

1. Log into the HammerDB instance via ssh.
2. Switch to the postgres user:

   ```
   su - postgres
   ```

3. Navigate to the HammerDB directory:

   ```
   cd HammerDB-3.3
   ```

4. Start hammerdbcli:

   ```
   ./hammerdbcli
   ```

5. Set the following variables:

   ```
   dbset db pg
   dbset bm tpc-c
   diset connection pg_host <IP_ADDRESS>
   diset tpcc pg_count_ware <db_size>
   diset tpcc pg_superuserpass <Password>
   diset tpcc pg_total_iterations 10000000
   diset tpcc pg_rampup 15
   diset tpcc pg_duration 30
   diset tpcc pg_driver timed
   ```

6. Load the driver script

   ```
   loadscript
   ```

7. Configure the virtual users to run the test:

   ```
   vuset vu <num_virtual_users>
   vuset vu logtotemp 1
   ```

8. Create the virtual users:

   ```
   vucreate
   ```

9. Run the test:

   ```
   vurun
   ```

10. After the test is complete destroy the virtual users:

    ```
    vudestroy
    ```

11. After destroying the virtual users, log into the PostgreSQL instance and restore the database:
12. Reboot the instance and rerun the test 2 more times for a total of 3 runs.

Restoring the database

We used the following script to restore the database in between each run:

```
#!/bin/bash
pg_ctl -D $PGDATA stop
rm -rf $PGDATA/*
rm -rf /var/lib/pgsql/13/log/*
cp -R /var/lib/pgsql/13/backups/data/* $PGDATA/
mv $PGDATA/pg_wal /var/lib/pgsql/13/log
ln -s /var/lib/pgsql/13/log/pg_wal $PGDATA/pg_wal
pg_ctl -D $PGDATA start
```

Warehouse configurations

85-warehouse PostgreSQL config

```
listen_addresses = '*'
port = 5432
max_connections = 256
shared_buffers = 20480MB
huge_pages = on
temp_buffers = 1024MB
work_mem = 1024MB
maintenance_work_mem = 512MB
autovacuum_work_mem = -1
max_stack_depth = 5MB
dynamic_shared_memory_type = posix
max_files_per_process = 4000
effective_io_concurrency = 32
wal_level = minimal
synchronous_commit = off
wal_buffers = 512MB
checkpoint_timeout = 1h
max_wal_size = 1GB
min_wal_size = 80MB
checkpoint_completion_target = 1
checkpoint_warning = 0
max_wal_senders = 0
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_messages = error
log_min_error_statement = error
log_line_prefix = '%m [%p] '
log_timezone = 'America/New_York'
autovacuum = off
datestyle = 'iso, mdy'
timezone = 'America/New_York'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
max_locks_per_transaction = 64
max_pred_locks_per_transaction = 64
```

240-warehouse PostgreSQL config

```
listen_addresses = '*'
port = 5432
max_connections = 256
shared_buffers = 51200MB
huge_pages = on
temp_buffers = 2048MB
work_mem = 2048MB
maintenance_work_mem = 1024MB
autovacuum_work_mem = -1
max_stack_depth = 5MB
dynamic_shared_memory_type = posix
max_files_per_process = 4000
effective_io_concurrency = 32
wal_level = minimal
synchronous_commit = off
wal_buffers = 1024MB
checkpoint_timeout = 1h
max_wal_size = 1GB
min_wal_size = 80MB
checkpoint_completion_target = 1
checkpoint_warning = 0
max_wal_senders = 0
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_messages = error
log_min_error_statement = error
log_line_prefix = '%m [%p] '
log_timezone = 'America/New_York'
autovacuum = off
datestyle = 'iso, mdy'
timezone = 'America/New_York'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
max_locks_per_transaction = 64
max_pred_locks_per_transaction = 64
```

600-warehouse PostgreSQL config

```
listen_addresses = '*'
port = 5432
max_connections = 1000
shared_buffers = 204800MB
huge_pages = on
temp_buffers = 4096MB
work_mem = 4096MB
maintenance_work_mem = 1024MB
autovacuum_work_mem = -1
max_stack_depth = 7MB
dynamic_shared_memory_type = posix
max_files_per_process = 4000
effective_io_concurrency = 32
wal_level = minimal
synchronous_commit = off
wal_buffers = 512MB
checkpoint_timeout = 1h
max_wal_size = 5GB
min_wal_size = 80MB
checkpoint_completion_target = 1
checkpoint_warning = 0
max_wal_senders = 0
effective_cache_size = 128GB
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_messages = error
log_min_error_statement = error
log_line_prefix = '%m [%p] '
log_timezone = 'America/New_York'
autovacuum = off
datestyle = 'iso, mdy'
timezone = 'America/New_York'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
max_locks_per_transaction = 64
max_pred_locks_per_transaction = 64
```

Determining CPU vulnerability mitigation

We used dnf to install the spectre-meltdown-checker and then ran the following command on each VM to determine the Intel processor mitigation settings that Google employs:

```
grep . /sys/devices/system/cpu/vulnerabilites/*
```

N1 VM instance

```
/sys/devices/system/cpu/vulnerabilities/itlb_multihit:Not affected
/sys/devices/system/cpu/vulnerabilities/l1tf:Mitigation: PTE Inversion
/sys/devices/system/cpu/vulnerabilities/mds:Mitigation: Clear CPU buffers; SMT Host state unknown
/sys/devices/system/cpu/vulnerabilities/meltdown:Mitigation: PTI
/sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Mitigation: Speculative Store Bypass
disabled via prctl and seccomp
/sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: usercopy/swapgs barriers and __user
pointer sanitization
/sys/devices/system/cpu/vulnerabilities/spectre_v2:Mitigation: Full generic retpoline, IBPB:
conditional, IBRS_FW, STIBP: conditional, RSB filling
/sys/devices/system/cpu/vulnerabilities/srbds:Not affected
/sys/devices/system/cpu/vulnerabilities/tsx_async_abort:Mitigation: Clear CPU buffers; SMT Host state
unknown
```

N2 VM instance

```
/sys/devices/system/cpu/vulnerabilities/itlb_multihit:Not affected
/sys/devices/system/cpu/vulnerabilities/l1tf:Not affected
/sys/devices/system/cpu/vulnerabilities/mds:Mitigation: Clear CPU buffers; SMT Host state unknown
/sys/devices/system/cpu/vulnerabilities/meltdown:Not affected
/sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Mitigation: Speculative Store Bypass
disabled via prctl and seccomp
/sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: usercopy/swapgs barriers and __user
pointer sanitization
/sys/devices/system/cpu/vulnerabilities/spectre_v2:Mitigation: Enhanced IBRS, IBPB: conditional, RSB
filling
/sys/devices/system/cpu/vulnerabilities/srbds:Not affected
/sys/devices/system/cpu/vulnerabilities/tsx_async_abort:Mitigation: Clear CPU buffers; SMT Host state
unknown
```

Read the report at http://facts.pt/0Q5h9xl ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®