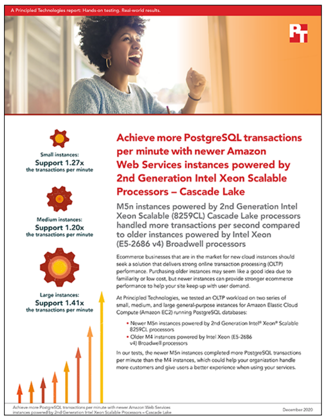
A Principled Technologies report: Hands-on testing. Real-world results.

Small instances: Support 1.27x the transactions per minute

Medium instances: Support 1.20x the transactions per minute

Large instances: Support 1.41x the transactions per minute

Achieve more PostgreSQL transactions per minute with newer Amazon Web Services instances powered by 2nd Generation Intel Xeon Scalable Processors – Cascade Lake

M5n instances powered by 2nd Generation Intel Xeon Scalable (8259CL) Cascade Lake processors handled more transactions per second compared to older instances powered by Intel Xeon (E5-2686 v4) Broadwell processors

Ecommerce businesses that are in the market for new cloud instances should seek a solution that delivers strong online transaction processing (OLTP) performance. Purchasing older instances may seem like a good idea due to familiarity or low cost, but newer instances can provide stronger ecommerce performance to help your site keep up with user demand.

At Principled Technologies, we tested an OLTP workload on two series of small, medium, and large general-purpose instances for Amazon Elastic Cloud Compute (Amazon EC2) running PostgreSQL databases:

- Newer M5n instances powered by 2nd Generation Intel® Xeon® Scalable 8259CL processors
- Older M4 instances powered by Intel Xeon (E5-2686 v4) Broadwell processors

In our tests, the newer M5n instances completed more PostgreSQL transactions per minute than the M4 instances, which could help your organization handle more customers and give users a better experience when using your services.

Achieve more PostgreSQL transactions per minute with newer Amazon Web Services instances powered by 2nd Generation Intel Xeon Scalable Processors – Cascade Lake

December 2020

The science behind the report:

Achieve more PostgreSQL transactions per minute with newer Amazon Web Services 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 Achieve more PostgreSQL transactions per minute with newer Amazon Web Services instances powered by 2nd Generation Intel Xeon Scalable Processors – Cascade Lake.

We concluded our hands-on testing on November 2, 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 29, 2020 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

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

2xlarge series (85WH/24Users)			
	m4	m5n	m5n advantage
Transactions per minute	291,729	369,397	1.27x
New orders per minute	126,892	160,603	1.27x

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

4xlarge series (240WH/32Users)			
	m4	m5n	m5n advantage
Transactions per minute	583,954	699,915	1.20x
New orders per minute	253,900	304,475	1.20x

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

16xlarge series (600WH/64Users)			
	m4	m5n	m5n advantage
Transactions per minute	1,427,001	2,016,463	1.41x
New orders per minute	620,588	876,660	1.41x

System configuration information

Table 4: Detailed information on the systems we tested.

System configuration information	m5n.2xlarge	m5n.4xlarge	m5n.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	10/30/2020	10/30/2020	10/30/2020
CSP/Region	us-east1-f	us-east1-f	us-east1-f
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	m5n.2xlarge	m5n.4xlarge	m5n.16xlarge
BIOS name and version	Amazon EC2 1.0, 10/16/2017	Amazon EC2 1.0, 10/16/2017	Amazon EC2 1.0, 10/16/2017
Operating system name and version/build number	CentOS 8.2	CentOS 8.2	CentOS 8.2
Date of last OS updates/ patches applied	10/29/2020	10/29/2020	10/29/2020
Processor			
Number of processors	1	1	2
Vendor and model	Intel® Xeon® Platinum 8259CL	Intel Xeon Platinum 8259CL	Intel Xeon Platinum 8259CL
Core count (per processor)	4	8	16
Core frequency (GHz)	2.50	2.50	2.50
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	up to 25Gb	up to 25Gb	75Gb
Storage bandwidth per VM instance	up to 4,750 Mbps	4,750 Mbps	13,600 Mbps

System configuration information	m5n.2xlarge	m5n.4xlarge	m5n.16xlarge
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	1,000	1,000	1,000
Drive information (speed, interface, type)	gp2	gp2	gp2
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	3,000	3,000	3,000
Drive information (speed, interface, type)	gp2	gp2	gp2
Local storage (log drive)			
Number of drives	1	1	1
Drive size (GB)	500	500	500
Drive information (speed, interface, type)	gp2	gp2	gp2
Network adapter			
Vendor and model	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter	Amazon Elastic Network Adapter
Number and type of ports	1x 25Gb	1x 25Gb	1x 75Gb

Table 5: Detailed information on the systems we tested.

System configuration information	m4.2xlarge	m4.4xlarge	m4.16xlarge
Tested by	Principled Technologies	Principled Technologies	Principled Technologies
Test date	10/30/2020	10/30/2020	10/30/2020
CSP/Region	us-east1-f	us-east1-f	us-east1-f
Workload & version	HammerDB v3.3 TPC-C-Like	HammerDB v3.3 TPC-C-Like	HammerDB v3.3 TPC-C-Like
WL specific parameters	vm.nr_hugepages = 12288	vm.nr_hugepages = 28672	vm.nr_hugepages = 102400
Iterations and result choice	3 runs, median	3 runs, median	3 runs, median
Server platform	m4.2xlarge	m4.4xlarge	m4.16xlarge
BIOS name and version	Xen 4.2.amazon, 8/24/2006	Xen 4.2.amazon, 8/24/2006	Xen 4.2.amazon, 8/24/2006
Operating system name and version/build number	CentOS 8.2	CentOS 8.2	CentOS 8.2
Date of last OS updates/ patches applied	10/29/2020	10/29/2020	10/29/2020

System configuration information	m4.2xlarge	m4.4xlarge	m4.16xlarge
Processor			
Number of processors	1	1	2
Vendor and model	Intel Xeon CPU E5-2686 v4	Intel Xeon CPU E5-2686 v4	Intel Xeon CPU E5-2686 v4
Core count (per processor)	4	8	16
Core frequency (GHz)	2.30	2.30	2.30
Stepping	1	1	1
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	Direct-attached	Direct-attached	Direct-attached
Network bandwidth per VM instance	High	High	10Gb
Storage bandwidth per VM instance	1,000 Mbps	2,000 Mbps	10,000 Mbps
Local storage (OS)			
Number of drives	1	1	1
Drive size (GB)	1,000	1,000	1,000
Drive information (speed, interface, type)	gp2	gp2	gp2
Local storage (data drive)			
Number of drives	1	1	1
Drive size (GB)	3,000	3,000	3,000
Drive information (speed, interface, type)	gp2	gp2	gp2
Local storage (log drive)			
Number of drives	1	1	1
Drive size (GB)	500	500	500
Drive information (speed, interface, type)	gp2	gp2	gp2
Network adapter			
Vendor and model	Intel 82599	Intel 82599	Amazon Elastic Network Adapter
Number and type of ports	1x 10Gb	1x 10Gb	1x 25Gb

How we tested

Testing overview

For this project, we tested AWS instances featuring older Intel E5_v4 processors vs. newer 2nd Generation Xeon processors. We ran a TPC-C-like workload on PostgreSQL on the AWS 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.

Create the PostgreSQL baseline image VM

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Launch instance, then Launch instance in the dropdown to open the Launch Instance wizard.
4. In the search window, enter Centos 8 and press Enter.
5. On the AWS Marketplace tab, click the Select button next to Centos 8 base by Amazon Web Services.
6. On the Choose Instance Type tab, select t2.medium, then click Next: Configure Instance Details.
7. On the Configure Instance tab, set the following:
    a. Number of instances: 1
    b. Purchasing option: Leave unchecked
    c. Network: Default VPC
    d. Subnet: Choose the region you're working 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 the Add Storage tab, set the following:
    a. Size: 30GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
    e. Click Add New Volume
    f. Size: 20GB
    g. Volume Type: gp2
    h. Delete on Termination: Checked
    i. Encryption: Not Encrypted
10. Click Add New Volume.
    a. Size: 20GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
11. Click Next: Add Tags.
12. On the Add Tags tab, leave defaults
13. Click Next: Configure Security Group
14. On the Configure Security Group tab, set the following:
    a. Leave defaults.
    b. Click Review and Launch.
15. On the Review Tab, click Launch.
16. Choose the appropriate option for the key pair, then click Launch Instances.

Create the HammerDB client baseline image VM

1. Log into AWS and navigate to the AWS Management Console.
2. Click on EC2
3. Click Launch instance, then Launch instance in the dropdown to open the Launch Instance wizard.
4. In the search window, enter Centos 8 and press enter.
5. On the AWS Marketplace tab, click the Select button next to Centos 8 base by Amazon Web Services.
6. On the Choose Instance Type tab, select t2.medium, then click "Next: Configure Instance Details".
7. On the Configure Instance tab, set the following:
    a. Number of instances: 1
    b. Purchasing option: Leave unchecked
    c. Network: Default VPC.
    d. Subnet: Choose the region you're working 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 the Add Storage tab, set the following:
    a. Size: 30GB
    b. Volume Type: gp2
    c. Delete on Termination: Checked
    d. Encryption: Not Encrypted
10. Click Next: Add Tags.
11. On the Add Tags tab, leave defaults
12. Click Next: Configure Security Group
13. On the Configure Security Group tab, set the following:
    a. Leave defaults.
    b. Click Review and Launch.
14. On the Review Tab, click Launch.
15. Choose the appropriate option for the key pair, then click Launch Instances.

Configure Centos 8 and install PostgreSQL

1. Log into 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 Centos user.
21. Shut down the instance:

```
sudo poweroff
```

Configure Centos 8 and install 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 PostgreSQL user home folder:

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

11. Change the ownership on the PostgreSQL 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 Centos user.
17. Shut down the instance:

```
sudo poweroff
```

Create an AMI of your baseline VM

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Running instances.
4. Place a checkmark next to the instance you wish to create an image from.
5. Click the Action dropdown and select Image→Create Image.
6. Enter the Image name and click Create Image.
7. Navigate to Images→AMI's in the menu on the left side of the page to see your new image.

Create your Instance with the baseline image

Create the PostgreSQL VM from your Image

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click on Images→AMIs.
4. Check the box next to the image you created in the previous step, and click Launch.
5. On the Choose Instance Type tab, select your VM size, then click Next: Configure Instance Details.
6. On the Configure Instance tab, set the following:
7. Number of instances: 1
8. Purchasing option: Leave unchecked.
9. Network: Default VPC.
10. Subnet: Choose the region you are working in.
11. Auto-assign Public IP: Enable.
12. Placement Group: Leave unchecked.
13. Capacity Reservation: Open
14. Domain join directory: No Directory
15. IAM role: None
16. Shutdown behavior: Stop
17. Click Next: Add Storage.
18. On the Add Storage tab, set the following:
19. Size: <Size>
20. Volume Type: Set your volume type. We chose gp2.
21. Delete on Termination: Unchecked.
22. Encryption: Not Encrypted.
23. Size: <Size>
24. Volume Type: Set your volume type. We chose gp2.
25. Delete on Termination: Unchecked.
26. Encryption: Not Encrypted.
27. Size: <Size>
28. Volume Type: Set your volume type. We chose gp2.
29. Delete on Termination: Unchecked
30. Encryption: Not Encrypted.
31. Click Next: Add Tags
32. On the Add Tags tab, leave defaults
33. Click Next: Configure Security Group
34. On the Configure Security Group tab, set the following:
35. Add your security group.
36. Click Review and Launch.
37. On the Review Tab, click Launch.
38. Choose the appropriate option for the key pair, then click Launch Instances.

Create the HammerDB VM from your Image

1. Log into AWS and navigate to the AWS Management Console.
2. Click EC2.
3. Click Images→AMIs.
4. Check the box next to the image you created in the previous step, and click Launch.
5. On the Choose Instance Type tab, select your VM size, then click Next: Configure Instance Details.
6. On the Configure Instance tab, set the following:
7. Number of instances: 1
8. Purchasing option: Leave unchecked.
9. Network: Default VPC.
10. Subnet: Choose the region you are working in.
11. Auto-assign Public IP: Enable.
12. Placement Group: Leave unchecked.
13. Capacity Reservation: Open
14. Domain join directory: No Directory
15. IAM role: None
16. Shutdown behavior: Stop
17. Click Next: Add Storage.
18. On the Add Storage tab, set the following:
19. Size: 30GB
20. Volume Type: Set your volume type. We chose gp2.
21. Delete on Termination: Unchecked.
22. Encryption: Not Encrypted.
23. Click Next: Add Tags
24. On the Add Tags tab, leave defaults
25. Click Next: Configure Security Group
26. On the Configure Security Group tab, set the following:
27. Add your security group.
28. Click Review and Launch.
29. On the Review Tab, click Launch.
30. Choose the appropriate option for the key pair, then click Launch Instances.

Configure PostgreSQL on the VMs under test

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

Set the number of Huge Pages

1. Log into the PostgreSQL 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.

Configure and start the database

1. Log into the PostgreSQL instance via ssh.
2. Switch to the PostgreSQL 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 to match the configuration from the files starting on page 13 for your instance size.
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
```

Create the database schema with HammerDB

1. Log into the HammerDB instance via ssh.
2. Switch to the PostgreSQL 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
```

Backup 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
```

Move the write ahead log to the log volume and start 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
```

Run 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, login to 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 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 Amazon employs:

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

M4 Instance

```
/sys/devices/system/cpu/vulnerabilities/itlb_multihit:KVM: Vulnerable
/sys/devices/system/cpu/vulnerabilities/l1tf:Mitigation: PTE Inversion
/sys/devices/system/cpu/vulnerabilities/mds:Vulnerable: Clear CPU buffers attempted, no microcode;
SMT Host state unknown
/sys/devices/system/cpu/vulnerabilities/meltdown:Mitigation: PTI
/sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Vulnerable
/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, STIBP:
disabled, RSB filling
/sys/devices/system/cpu/vulnerabilities/srbds:Not affected
/sys/devices/system/cpu/vulnerabilities/tsx_async_abort:Vulnerable: Clear CPU buffers attempted, no
microcode; SMT Host state unknown
```

m5n instance

```
/sys/devices/system/cpu/vulnerabilities/itlb_multihit:KVM: Vulnerable
/sys/devices/system/cpu/vulnerabilities/l1tf:Mitigation: PTE Inversion
/sys/devices/system/cpu/vulnerabilities/mds:Vulnerable: Clear CPU buffers attempted, no microcode;
SMT Host state unknown
/sys/devices/system/cpu/vulnerabilities/meltdown:Mitigation: PTI
/sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Vulnerable
/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, STIBP:
disabled, RSB filling
/sys/devices/system/cpu/vulnerabilities/srbds:Not affected
/sys/devices/system/cpu/vulnerabilities/tsx_async_abort:Not affected
```

Read the report at http://facts.pt/xHrpYj1

This project was commissioned by Intel.