ASCAP Quick Run Book

EDB Postgres Run Book
by EnterpriseDB® Corporation

Table Of Contents

Installing EDB Postgres Advanced Server (EPAS) 9.6

Please make sure to get your EDB YUM repo credentials before using following steps:

1. Add the EnterpriseDB (EDB) YUM Repository to your server

```
rpm -ivh http://yum.enterprisedb.com/edbrepos/edb-repo-9.6-4.noarch.rpm
```

2. Set the YUM username and password in the edb.repo created on step 1

```
# sed -i -- "s#<username>#$YUMUSERNAME#g" /etc/yum.repos.d/edb.repo
# sed -i -- "s#<password>#$YUMPASSWORD#g" /etc/yum.repos.d/edb.repo
# sed -i "\/edbas96/,/gpgcheck/ s/enabled=0/enabled=1/" \
         /etc/yum.repos.d/edb.repo
# sed -i "\/enterprisedb-dependencies/,/gpgcheck/ s/enabled=0/enabled=1/"
/etc/yum.repos.d/edb.repo
```

3. Install EDB Postgres Advanced Server 9.6 core database server components

```
yum install edb-as96-server -y
```

4. Initialize the EDB Postgres database cluster

To use a non-default PGDATA location, simply create the desired PGDATA folder structure and create a symbolic link from the default location to the custom location

```
mkdir -p $PGDATA
chown enterprisedb:enterprisedb -R $PGDATA
chmod 0700 $PGDATA
rm -rf /var/lib/edb/as9.6/data/
sudo -u enterprisedb ln -s $PGDATA /var/lib/edb/as9.6/data
```

Run initdb for Oracle compatibility

```
/usr/edb/as9.6/bin/edb-as-96-setup initdb
```

Run initdb for Postgres compatibility

```
/usr/edb/as9.6/bin/initdb --no-redwood-compat --encoding=UTF8 -U postgres
-D /var/lib/edb/as9.6/data -X /xlog

/pgdata- database stuff
/xlog - transaction logs
/wal_archive - replay of transaction logs for replication
```

5. Start EDB Postgres as a system service.

- For postgres compatibility mode only, change owner of data directory from enterprisedb to postgres
  Because /var/lib/edb/as9.6/data is linked to /pgdb/as9.6/data, two data directory must be changed from enterprisedb to postgres

```
# chown postgres:postgres -R /var/lib/edb
# chmod 700 /var/lib/edb
# chown postgres:postgres -R /pgdb/as9.6/data
# chmod 700 /pgdb/as9.6/data
```

- For postgres compatibility mode only, change user and group name from enterprisedb to postgres

```
# vi /usr/lib/systemd/system/edb-as-9.6.service

#User=enterprisedb
#Group=enterprisedb

User=postgres
Group=postgres
```

- For postgres compatibility mode only,reload systemctl daemon-reload

```
# systemctl daemon-reload
```

- For starting service on RHEL/CentOS version 6, following command can be used

```
# service edb-as-9.6 start
```

- For starting service on RHEL/CentOS version 7, following command can be used

```
# systemctl start edb-as-9.6
```

6. Configure EDB Postgres to start on boot

```
chkconfig edb-as-9.6 on
```

7. Verify services are running

```
# ps -ef | grep postgres
root       806 28543  0 10:20 pts/0    00:00:00 grep --color=auto
postgres
postgres 29342     1  0 09:49 ?        00:00:00
/usr/edb/as9.6/bin/edb-postmaster -D /var/lib/edb/as9.6/data
postgres 29346 29342  0 09:49 ?        00:00:00 postgres: logger process
postgres 29348 29342  0 09:49 ?        00:00:00 postgres: checkpointer
process
postgres 29349 29342  0 09:49 ?        00:00:00 postgres: writer process
postgres 29350 29342  0 09:49 ?        00:00:00 postgres: wal writer
process
postgres 29351 29342  0 09:49 ?        00:00:00 postgres: autovacuum
launcher process
postgres 29352 29342  0 09:49 ?        00:00:00 postgres: archiver
process
```

```
postgres 29353 29342  0 09:49 ?        00:00:00 postgres: stats
collector process
postgres 29354 29342  0 09:49 ?        00:00:00 postgres: bgworker:
dbms_aq launcher
postgres 32285 29342  0 10:12 ?        00:00:00 postgres: postgres
postgres 127.0.0.1[52958] idle
```

EDB Backup and Recovery Tool (BART)

Database administrators today typically have to create complex scripts, manage backup file naming and storage, move files for recovery, create special recovery configuration files, manage file permissions, and clean up after every recovery event. At its best, this complex set of tasks has many points of potential problems. At its worst, it may discourage DBAs from performing backups effectively — jeopardizing organizational data when a recovery is needed.

Based on best practices and built by the world leader in Postgres tools, EDB Postgres Backup and Recovery makes it simple and easy for database administrators to safeguard organizational data.

All of the examples show the most common recovery target of 'latest'. It is possible to restore to a single point in time or to a txid. This is possible by adding the recovery_target_time to restore to a timestamp:

```
recovery_target_time = '2017-09-13 12:19:00'
```

Introduction

EDB Backup and Recovery Tool (BART) is an easy to use backup and recovery utility allowing DBA's to manage backup and recoveries (e.g., PITR), compression/validation, and set up retention policies for large scale PostgreSQL and EDB Postgres deployments. `pg_basebackup` is the utility used by BART for creating backups.

This section provides a top level overview of how to quickly get started installing and configuring BART, for more detailed information see link below:

Prerequisites

- Supported Platforms
  - CentOS 6.x or 7.x
  - Red Hat Enterprise Linux (RHEL) 6.x or 7.x

- Required Software
  - Postgres libpq library
  - Postgres `pg_basebackup`

- Database Servers
  - WAL archiving enabled for PITR
  - Replication setup required by `pg_basebackup`

The BART host components are installed using EnterpriseDB RPM packages.

- The Secure Shell (SSH) server daemon must be enabled and active on the BART host as well as on any remote database server hosts on which BART will be managing backup and recovery

- The SSH and SCP client programs must be available on the BART host as well as on the remote database server hosts

1. Install EDB Backup and Recovery with root

```
# yum install edb-bart20 -y
```

2. Create Users for Bart Setup

- For centralized bart setup ,use OS user “postgres” as bart user .
- Make OS user “postgres” with same UID and GID across all of remote DB servers and bart server.
- The owner of bart backup directory and data directory on all of DB servers is OS user “postgres”

- Login postgres as the bart user.

```
# su - postgres
$ vi .bash_profile
        export LD_LIBRARY_PATH=/usr/edb/bart2.0/lib:$PATH
        export PATH=/usr/edb/bart2.0/bin:$PATH

<ESQ>:wq <ENTER>
```

3. Create the backup directory

```
# mkdir -p /pgbackup/bartbkup
# chown postgres:postgres -R /pgbackup/bartbkup
# chmod 700 /pgbackup/bartbkup
```

4. Edit BART Config file: /usr/edb/bart2.0/etc/bart.cfg

```
[BART]
bart_host= postgres@127.0.0.1
backup_path = /pgbackup/bartbkup
pg_basebackup_path = /usr/edb/as9.6/bin/pg_basebackup
logfile = /tmp/bart_log
scanner_logfile = /tmp/bart_scanner.log

[DXPGEDB01]
host = 127.0.0.1
port = 5444
user = postgres
```

```
cluster_owner = postgres
backup_name = dxpgedb01_%year-%month-%dayT%hour:%minute:%second
retention_policy = 3 BACKUPS
archive_command = 'cp %p %a/%f'
allow_incremental_backups = enabled
description = "dxpgedb01"


[DXPGDB01]
host = 10.6.3.230
port = 5432
user = repuser
cluster_owner = postgres
backup_name = dxpgdb01_%year-%month-%dayT%hour:%minute:%second
retention_policy = 3 BACKUPS
archive_command = 'cp %p %a/%f'
allow_incremental_backups = enabled
description = "dxpgdb01"
```

5. Create replication user in each DB server

```
edb=# CREATE ROLE repuser WITH LOGIN SUPERUSER PASSWORD 'repuser';
```

6. Add a pg_hba.conf entry for the repuser from pg_hba.conf

```
host    template1       repuser        $BARTSERVERIP/32         md5
host    replication     repuser        $BARTSERVERIP/32         md5
```

7. Passwordless SSH required

For local backup

```
$ ssh-keygen -t rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 600  ~/.ssh/authorized_keys
```

For remote backup

```
$ ssh-keygen -t rsa    # on both bart server and remote server
$ scp ~/.ssh/id_rsa.pub postgres@remote_server:temp.pub # from bart server
$ scp ~/.ssh/id_rsa.pub postgres@bart_server:tmp.pub # from remote server
$ cat tmp.pub >> ~/.ssh/authorized_keys # on both bart server and remote
$ chmod 600 ~/.ssh/id_rsa.pub # both bart server and remote server
```

8. Create .pgpass file in home directory of postgres on BART server

```
vi .pgpass

*:5432:*:repuser:repuser
*:5444:*:repuser:repuser
```

9. Reload the configuration

```
$ pg_ctl reload
```

10. Enable archive log

"archive_command" will be automatically generated by bart config file in postgresql.auto.conf

```
$ vi $PGDATA/postgres.conf
arhive_mode = on
wal_levle = archive # or replica
show max_wal_senders = 5
```

11. Initial backup server

option -o is automatically adding archive_command to postgresql.auto.conf

```
$ bart INIT -s server_name -o
```

12. Check backup servers

```

[BART]
bart_host= postgres@127.0.0.1
backup_path = /pgbackup/bartbkup
pg_basebackup_path = /usr/edb/as9.6/bin/pg_basebackup
logfile = /tmp/bart_log
scanner_logfile = /tmp/bart_scanner.log

[DXPGEDB01]
host = 127.0.0.1
port = 5444
user = postgres
cluster_owner = postgres
backup_name = dxpgedb01_%year-%month-%dayT%hour:%minute:%second
retention_policy = 3 BACKUPS
archive_command = 'cp %p %a/%f'
allow_incremental_backups = enabled
description = "dxpgedb01"


[DXPGDB01]
host = 10.6.3.230
port = 5432
user = repuser
cluster_owner = postgres
```

```
backup_name = dxpgdb01_%year-%month-%dayT%hour:%minute:%second
retention_policy = 3 BACKUPS
archive_command = 'cp %p %a/%f'
allow_incremental_backups = enabled
description = "dxpgdb01"




[postgres@dxpgedb01 pgdb]$ bart show-servers
SERVER NAME          : dxpgdb01
BACKUP FRIENDLY NAME: dxpgdb01_%year-%month-%dayT%hour:%minute:%second
HOST NAME            : 10.6.3.230
USER NAME            : repuser
PORT                 : 5432
REMOTE HOST          :
RETENTION POLICY     : 3 Backups
DISK UTILIZATION     : 612.76 MB
NUMBER OF ARCHIVES   : 46
ARCHIVE PATH         : /pgbackup/bartbkup/dxpgdb01/archived_wals
ARCHIVE COMMAND      : cp %p /pgbackup/bartbkup/dxpgdb01/archived_wals/%f
XLOG METHOD          : fetch
WAL COMPRESSION      : disabled
TABLESPACE PATH(s)   :
INCREMENTAL BACKUP   : ENABLED
DESCRIPTION          : "dxpgdb01"

SERVER NAME          : dxpgedb01
BACKUP FRIENDLY NAME: dxpgedb01_%year-%month-%dayT%hour:%minute:%second
HOST NAME            : 127.0.0.1
USER NAME            : postgres
PORT                 : 5444
REMOTE HOST          :
RETENTION POLICY     : 3 Backups
DISK UTILIZATION     : 461.41 MB
NUMBER OF ARCHIVES   : 32
ARCHIVE PATH         : /pgbackup/bartbkup/dxpgedb01/archived_wals
ARCHIVE COMMAND      : cp %p /pgbackup/bartbkup/dxpgedb01/archived_wals/%f
XLOG METHOD          : fetch
WAL COMPRESSION      : disabled
TABLESPACE PATH(s)   :
INCREMENTAL BACKUP   : ENABLED
DESCRIPTION          : "dxpgedb01"
```

13. Restart DB server

```
$ pg_ctl restart
```

14. Take a backup with BART

Take full backup

```
[postgres@dxpgedb01 ~]$bart backup -s dxpgedb01 --backup-name
dxpgedb01_full_parent


INFO:  creating backup for server 'dxpgedb01'
INFO:  backup identifier: '1505937927619'
44586/44586 kB (100%), 1/1 tablespace
```

```
INFO:  backup completed successfully
INFO:  backup checksum: de7691412627a6c3ee1f67c1bf4aa32b of base.tar
INFO:
BACKUP DETAILS:
BACKUP STATUS: active
BACKUP IDENTIFIER: 1505937927619
BACKUP NAME: dxpgedb01_full_parent
BACKUP PARENT: none
BACKUP LOCATION: /pgbackup/bartbkup/dxpgedb01/1505937927619
BACKUP SIZE: 43.54 MB
BACKUP FORMAT: tar
BACKUP TIMEZONE: US/Eastern
XLOG METHOD: fetch
BACKUP CHECKSUM(s): 1
 ChkSum                             File
 de7691412627a6c3ee1f67c1bf4aa32b   base.tar

TABLESPACE(s): 0
START WAL LOCATION: 000000010000000000000023
STOP WAL LOCATION: 000000010000000000000023
BACKUP METHOD: streamed
BACKUP FROM: master
START TIME: 2017-09-20 16:05:27 EDT
STOP TIME: 2017-09-20 16:05:27 EDT
TOTAL DURATION: 0 sec(s)
```

Enable bart-scanner on bart server

```
$ bart-scanner --daemon
```

Incremental backup

```
$bart BACKUP -s dxpgedb01 -F p --parent dxpgedb01_full_parent
--backup-name dxpgedb01_incr_1-a


INFO:  creating incremental backup for server 'dxpgedb01'
INFO:  checking mbm files /pgbackup/bartbkup/dxpgedb01/archived_wals
INFO:  new backup identifier generated 1505938658292
INFO:  reading directory /pgbackup/bartbkup/dxpgedb01/archived_wals
INFO:  all files processed
NOTICE:  pg_stop_backup complete, all required WAL segments have been
archived
INFO:  incremental backup completed successfully
INFO:
BACKUP DETAILS:
BACKUP STATUS: active
BACKUP IDENTIFIER: 1505938658292
BACKUP NAME: dxpgedb01_incr_1-a
BACKUP PARENT: 1505937927619
BACKUP LOCATION: /pgbackup/bartbkup/dxpgedb01/1505938658292
BACKUP SIZE: 32.93 MB
BACKUP FORMAT: plain
BACKUP TIMEZONE: US/Eastern
XLOG METHOD: fetch
BACKUP CHECKSUM(s): 0
TABLESPACE(s): 0
START WAL LOCATION: 00000001000000000000002F
STOP WAL LOCATION: 000000010000000000000030
BACKUP METHOD: pg_start_backup
BACKUP FROM: master
START TIME: 2017-09-20 16:17:38 EDT
STOP TIME: 2017-09-20 16:17:39 EDT
TOTAL DURATION: 1 sec(s)
```

```
```

Incremental backup based previous incremental backup

```
[postgres@dxpgedb01 ~]$ bart BACKUP -s dxpgedb01 -F p --parent
dxpgedb01_incr_1-a --backup-name dxpgedb01_incr_1-b

[postgres@dxpgedb01 archived_wals]$ bart BACKUP -s dxpgedb01 -F p --parent
dxpgedb01_incr_1-a --backup-name dxpgedb01_incr_1-b
INFO:  creating incremental backup for server 'dxpgedb01'
INFO:  checking mbm files /pgbackup/bartbkup/dxpgedb01/archived_wals
INFO:  new backup identifier generated 1505938817312
INFO:  reading directory /pgbackup/bartbkup/dxpgedb01/archived_wals
INFO:  all files processed
NOTICE:  pg_stop_backup complete, all required WAL segments have been
archived
INFO:  incremental backup completed successfully
INFO:
BACKUP DETAILS:
BACKUP STATUS: active
BACKUP IDENTIFIER: 1505938817312
BACKUP NAME: dxpgedb01_incr_1-b
BACKUP PARENT: 1505938658292
BACKUP LOCATION: /pgbackup/bartbkup/dxpgedb01/1505938817312
BACKUP SIZE: 32.93 MB
BACKUP FORMAT: plain
BACKUP TIMEZONE: US/Eastern
XLOG METHOD: fetch
BACKUP CHECKSUM(s): 0
TABLESPACE(s): 0
START WAL LOCATION: 000000010000000000000033
STOP WAL LOCATION: 000000010000000000000034
BACKUP METHOD: pg_start_backup
BACKUP FROM: master
START TIME: 2017-09-20 16:20:17 EDT
STOP TIME: 2017-09-20 16:20:18 EDT
TOTAL DURATION: 1 sec(s)
```

15. List available backups with BART

```
[postgres@dxpgedb01 pgdb]$ bart show-backups
 SERVER NAME   BACKUP ID       BACKUP NAME              BACKUP PARENT
BACKUP TIME                BACKUP SIZE   WAL(s) SIZE   WAL FILES   STATUS

 dxpgdb01      1505936643243   dxpgdb01_incr_1-a
dxpgdb01_full_parent    2017-09-20 15:44:04 EDT   32.81 MB
active
 dxpgdb01      1505935762632   dxpgdb01_full_parent     none
2017-09-20 15:29:27 EDT   57.97 MB      352.00 MB     22          active
 dxpgdb01      1505934943581   full_parent              none
2017-09-20 15:15:47 EDT   57.97 MB      80.00 MB      5           active
 dxpgedb01     1505938817312   dxpgedb01_incr_1-b       dxpgedb01_incr_1-a
2017-09-20 16:20:17 EDT   32.93 MB                                active
 dxpgedb01     1505938658292   dxpgedb01_incr_1-a
dxpgedb01_full_parent   2017-09-20 16:17:38 EDT   32.93 MB
active
 dxpgedb01     1505937927619   dxpgedb01_full_parent    none
2017-09-20 16:05:27 EDT   43.54 MB      304.00 MB     19          active
```

16. Perform a restore with BART

If restoring incremental backup . BART rpm should be installed on remote DB server

```
[remote_server]# yum install edb-bart20 -y
```

Create a data directory to hold the restored backup, making sure the correct privileges and ownership are set.

Following is no incremental backup restore steps

```
# mkdir $RESTOREDIR
# chown postgres:postgres -R $RESTOREDIR
# chmod 0700 $RESTOREDIR
```

Perform a restore of the most recent backup as follows:

```
$ bart RESTORE -s dxpgdb01 -p $RESTOREDIR
```

Use the -i option restore to a specific backup, using the backup id or backup name

Start the database server

```
pg_ctl start -D $RESTOREDIR
```

PEM Server Installation

Introduction

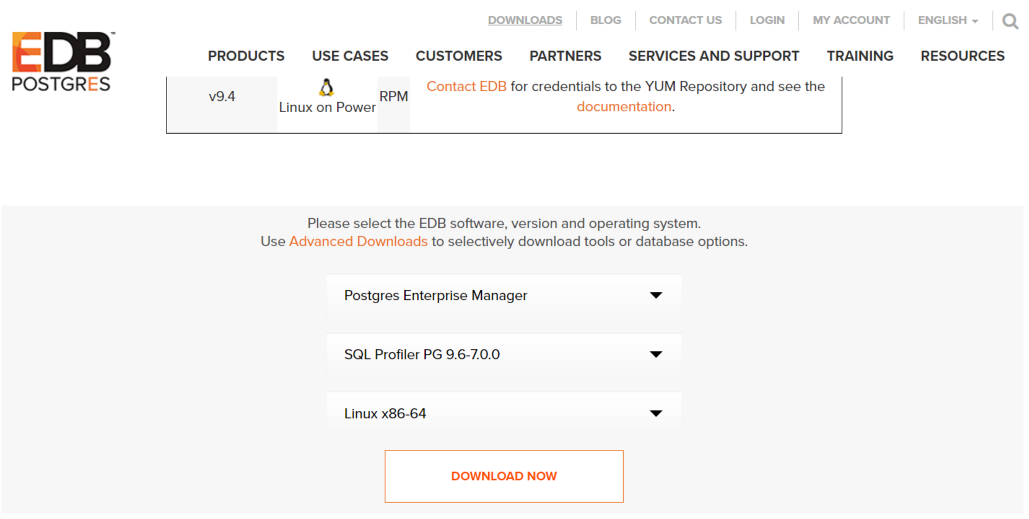
Postgres Enterprise Manager™ (PEM) is designed to assist database administrators, system architects, and performance analysts when administering, monitoring, and tuning PostgreSQL and Advanced Server database servers. PEM has been designed to manage and monitor a single server or multiple servers from a single console, allowing complete control over monitored databases.

1. Download PEM binary

Go to EDB Downloads

https://www.enterprisedb.com/software-downloads-postgres

Scroll down to Download Pick list
Select your product: Postgres Enterprise Manager
Select your version: PEM Server $latestversion
Select your operating system: Linux x86-64



v9.4	Linux on Power	RPM	Contact EDB for credentials to the YUM Repository and see the documentation.

Please select the EDB software, version and operating system.
Use Advanced Downloads to selectively download tools or database options.

Postgres Enterprise Manager

SQL Profiler PG 9.6-7.0.0

Linux x86-64

2. Copy the PEM binary to the server

```
# scp ~/Downloads/pem-server-7.0.0-1-linux-x64.run@pem_server:/tmp
```

3. Extract the binaries

Extract the binaries to any valid directory

```
# ./pem-server-7.0.0-1-linux-x64.run  --extract-dependents /tmp/
```

4. Install the Language Pack

```
# ./edb-languagepack-10-2-linux-x64.run
```

5. Install Apache

You will be prompted for the port number and installation directory.

```
# ./pem-httpd-2.4.25-1-linux-x64.run
```

6. Install PostgreSQL

If a PostgreSQL or EDB Postgres installation is not already running, you can install PostgreSQL from the extracted binary.
Install PostgreSQL to host the PEM database. The installer will prompt you for the installation directory, Data directory, port, and password for the database superuser. A postgres user account will be created if not present.

```
[root@dxpgpem01 ~]# ./postgresql-9.6.3-2-linux-x64.run

Welcome to the PostgreSQL Setup Wizard.


Please specify the directory where PostgreSQL will be installed.

Installation Directory [/opt/PostgreSQL/9.6]:


Please select a directory under which to store your data.

Data Directory [/opt/PostgreSQL/9.6/data]:


Please provide a password for the database superuser (postgres). A locked
Unix
user account (postgres) will be created if not present.

Password :postgres
Retype password :postgres

Please select the port number the server should listen on.

Port [5432]:


Advanced Options

Select the locale to be used by the new database cluster.

Locale

[1] [Default locale]
[2] aa_DJ
[3] aa_DJ.iso88591
[4] aa_DJ.utf8
.........
[771] zu_ZA
[772] zu_ZA.iso88591
[773] zu_ZA.utf8
Please choose an option [1] :


Setup is now ready to begin installing PostgreSQL on your computer.

Do you want to continue? [Y/n]: y
```

```
---------------------------------------------------------------------------
--
Please wait while Setup installs PostgreSQL on your computer.

 Installing
 0% ______________ 50% ______________ 100%
 #########################################

---------------------------------------------------------------------------
--
Setup has finished installing PostgreSQL on your computer.
```

7. Install PEM Server 7.0

During the PEM Server installation you will be prompted for your enterprisedb.com user account. The install will prompt you to select an installation directory, ask for which components you would like to install, and check for running database servers. You will need the superuser credentials (postgres or enterprisedb) to complete the installation.
Be sure to Select Advanced Options when prompted to install Web Services and Database.

```
# ./pem-server-7.0.0-1-linux-x64.run

Do you accept this license? [y/n]: y

EnterpriseDB User Account Information

Please enter the email address and password for your enterprisedb.com user
account.

Email address []:  guorong.zhang@enterprisedb.com

Password :

Installation Directory

Please select a directory for the PEM server installation.

Installation Directory [/opt/PEM]:

Show &advanced options [y/N]: y

Advanced options
```

```
Select installation type

[1] Web Services and Database [2] Web Services [3] Database Please choose
an option [1] :

Database Server Selection

Select the database server.

The database servers below may be used by Postgres Enterprise Manager for
its data store.

[1] PostgreSQL 9.5 on Port 5432

[2] Other Database Server

Please choose an option [1] : 1

Database Server Installation Details

Please specify user and password for the local server - PostgreSQL 9.5
installation running on port 5432.

User [postgres]:

Password :postgres

Network Details

Please enter the CIDR formatted network address range that agents will
connect to the server from, to be added to the server's pg_hba.conf file.
For example,192.168.1.0/24.

Network address [127.0.0.1/32]:

Agent Details.

Please specify a description for the agent.

Description. [Postgres Enterprise Manager Host]:

Agent certificate path [/root/.pem]:

Setup is now ready to begin installing the PEM server on your computer.

Do you want to continue? [Y/n]: Y

Please wait while Setup installs the PEM server on your computer.

Installing
```

```
0% ______________ 50% ______________ 100%

#########################################

Info: Configured the webservice for Postgres Enterprise Manager (PEM)
Server on port 8443. Created and configured the pem database. Press
[Enter] to continue:

EnterpriseDB is the leading provider of value-added products and services
for

the Postgres community.

Please visit our website at www.enterprisedb.com
```

8. Add PEM agent to servers to be monitored

- install pem agent on managed DB server

```
# yum install pem-agent -y
```

pem-agent installation: /usr/pem/agent

- Add a pg_hba.conf entry (if necessary)

If the IP address of the server to be monitored was not entered during the PEM server install then it will have to be added in the pg_hba.conf.

--before add:

```
# Allow local PEM agents and admins to connect to PEM server

hostssl pem      +pem_user   10.6.3.64/32       md5

hostssl postgres +pem_user   10.6.3.64/32       md5

hostssl pem      +pem_user   127.0.0.1/32       md5

hostssl pem      +pem_agent  127.0.0.1/32       cert
```

-- after add:

hostssl pem   +pem_user 10.6.3.63/32 md5

hostssl pem   +pem_agent 10.6.3.63/32 cert

hostssl postgres +pem_user   10.6.3.63/32 md5

# Allow local PEM agents and admins to connect to PEM server

hostssl pem   +pem_user   10.6.3.64/32   md5

hostssl postgres +pem_user   10.6.3.64/32   md5

hostssl pem   +pem_user   127.0.0.1/32   md5

hostssl pem   +pem_agent 127.0.0.1/32   cert

# PostgreSQL Client Authentication Configuration File

-- reload pg_hba.conf on PEM server

```
$ pg_ctl reload
```

9. Configure the Agent after pem agent Installation

```
#  cd /usr/pem/agent

# PGPASSWORD=postgres /usr/pem/agent/bin/pemworker --register-agent --pem-server
dxpgpem01.ascap.com  --pem-port 5432 --pem-user postgres
```

10.Edit pem-agent file

Adding following to file:t/usr/pem/agent/etc/ agent.cfg
heartbeat_connection = true
ca_file=/usr/libexec/libcurl-pem7/share/certs/ca-bundle.cr

```
[root@dxpgedb01 etc]# cat/usr/pem/agent/etc/ agent.cfg
[PEM/agent]
pem_host=dxpgpem01.ascap.com
pem_port=5432
```

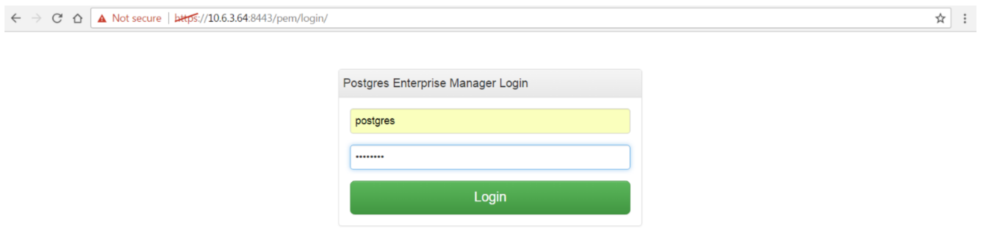
```
agent_id=3
agent_ssl_key=/root/.pem/agent3.key
agent_ssl_crt=/root/.pem/agent3.crt
log_level=warning
log_location=/var/log/pem/worker.log
agent_log_location=/var/log/pem/agent.log
long_wait=30
short_wait=10
alert_threads=0
enable_smtp=true
enable_snmp=ture
allow_server_restart=true
allow_package_management=false
allow_streaming_replication=false
max_connections=0
connect_timeout=-1
connection_lifetime=0
allow_batch_probes=false
# add following tow line
heartbeat_connection = true
ca_file=/usr/libexec/libcurl-pem7/share/certs/ca-bundle.cr
```

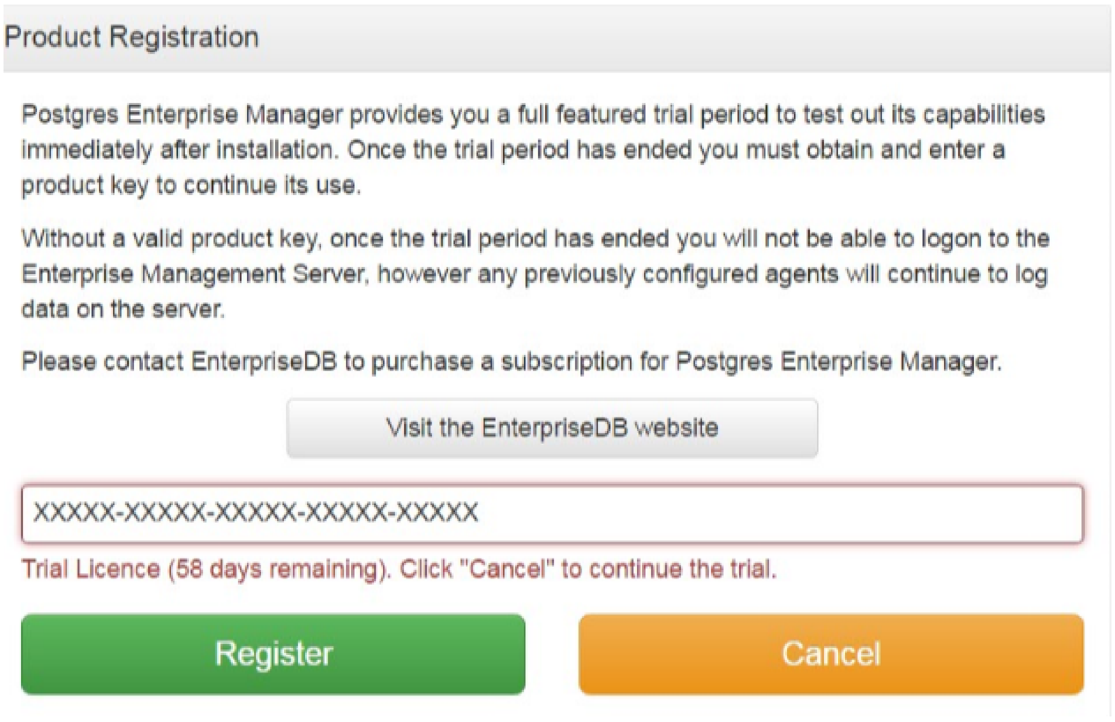
11. Start pemagent on managed db server

```
[root@dxpgedb01 bin]# systemctl start pemagent
```

12. Verify the pem agent in PEM and bind the Agent

Log in to PEM by going to: https://10.6.3.64:8443/pem/browser/


Not secure
https://10.6.3.64:8443/pem/login/
Postgres Enterprise Manager Login
postgres
Login



Product Registration

Postgres Enterprise Manager provides you a full featured trial period to test out its capabilities immediately after installation. Once the trial period has ended you must obtain and enter a product key to continue its use.

Without a valid product key, once the trial period has ended you will not be able to logon to the Enterprise Management Server, however any previously configured agents will continue to log data on the server.

Please contact EnterpriseDB to purchase a subscription for Postgres Enterprise Manager.

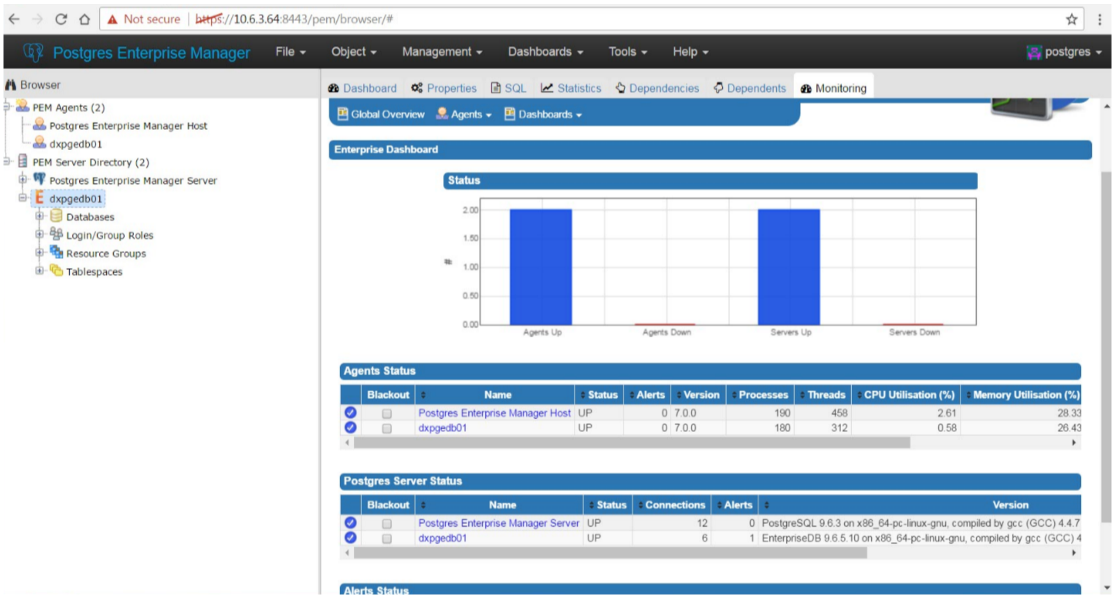
Visit the EnterpriseDB website

XXXXX-XXXXX-XXXXX-XXXXX-XXXXX

Trial Licence (58 days remaining). Click "Cancel" to continue the trial.

Register Cancel

Please enter production key , if not click Cancel .



Agents Status

	Blackout	Name	Status	Alerts	Version	Processes	Threads	CPU Utilisation (%)	Memory Utilisation (%)
✓	☐	Postgres Enterprise Manager Host	UP	0	7.0.0	190	458	2.61	28.33
✓	☐	dxpgedb01	UP	0	7.0.0	180	312	0.58	26.43

Postgres Server Status

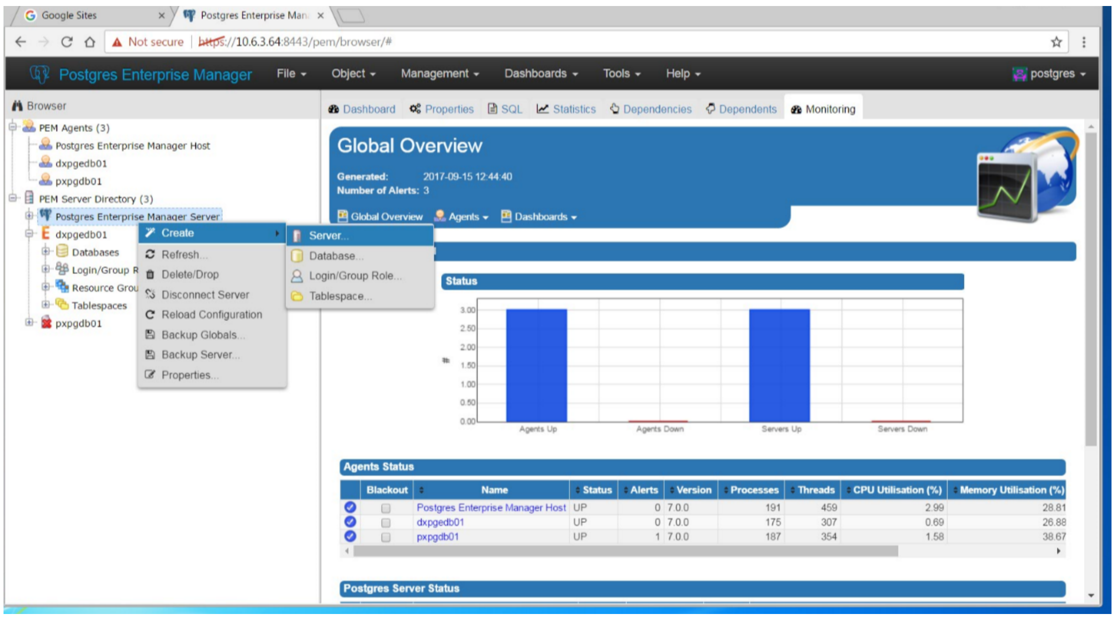
	Blackout	Name	Status	Connections	Alerts	Version
✓	☐	Postgres Enterprise Manager Server	UP	12	0	PostgreSQL 9.6.3 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.4.7
✓	☐	dxpgedb01	UP	6	1	EnterpriseDB 9.6.5.10 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4

13.Add managed server to PEM

Add PEM server entry to pg_hba.conf in managed server

```
host    all         postgres        10.6.3.64/32          md5  # this is pem server
```

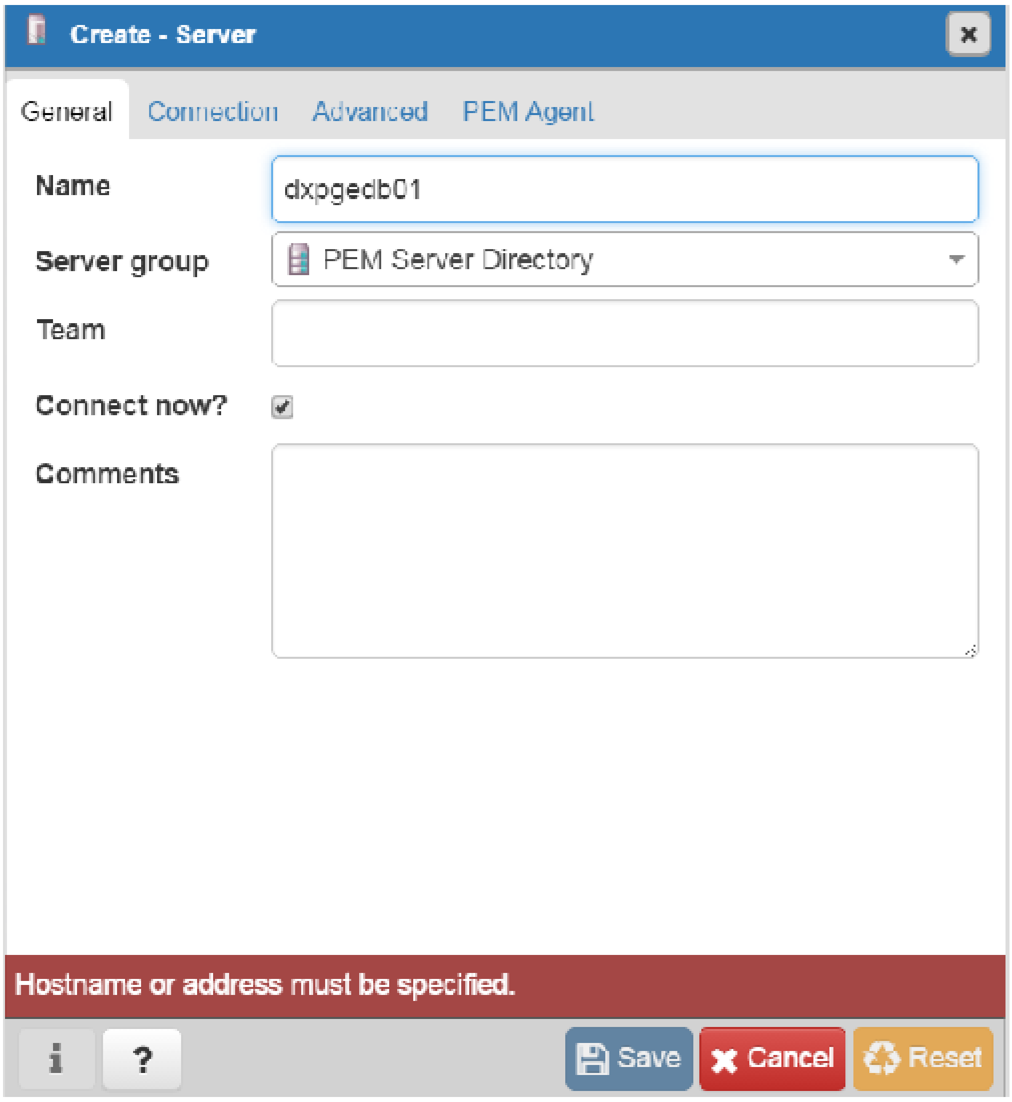
In PEM web client , righ click “ pem server directory” , then clidk “Create” then click “Server”



Agents Status

	Blackout	Name	Status	Alerts	Version	Processes	Threads	CPU Utilisation (%)	Memory Utilisation (%)
✓	☐	Postgres Enterprise Manager Host	UP	0	7.0.0	191	459	2.99	28.81
✓	☐	dxpgedb01	UP	0	7.0.0	175	307	0.69	26.88
✓	☐	pxpgdb01	UP	1	7.0.0	187	354	1.58	38.67

under General: enter managed server name.



Create - Server

Name: dxpgedb01

Server group: PEM Server Directory

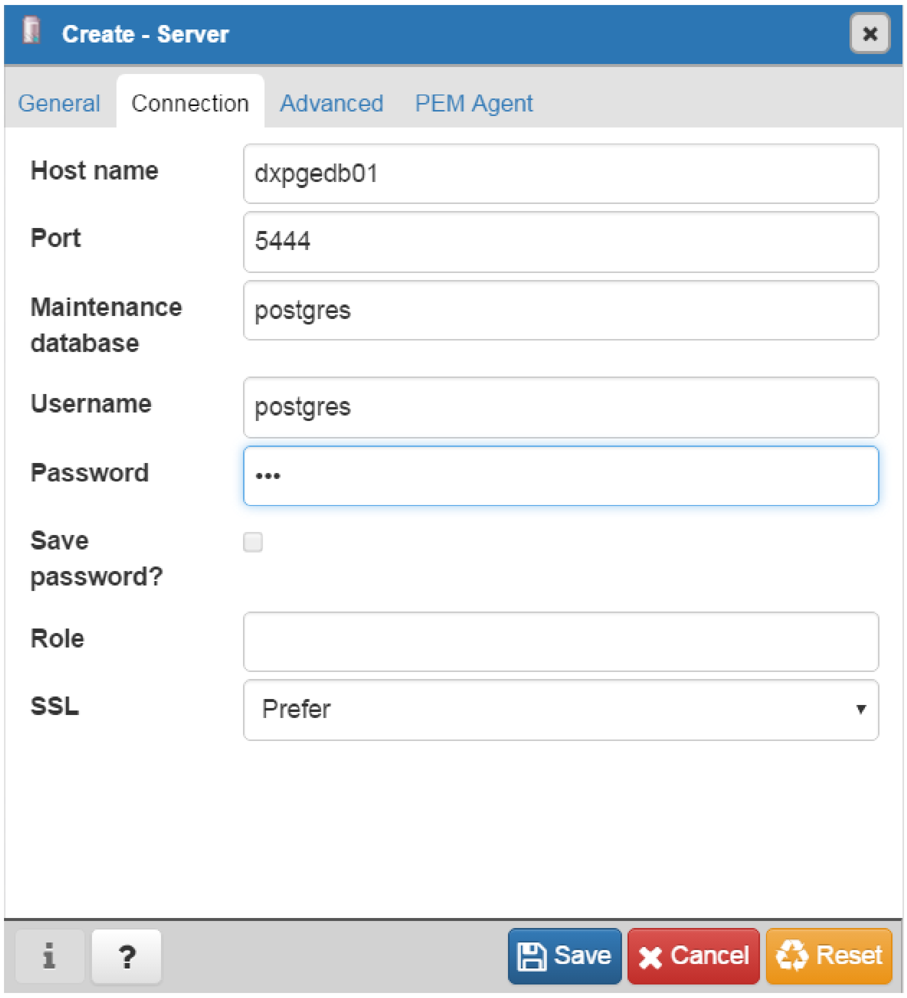
Team:

Connect now? ✓

Comments:

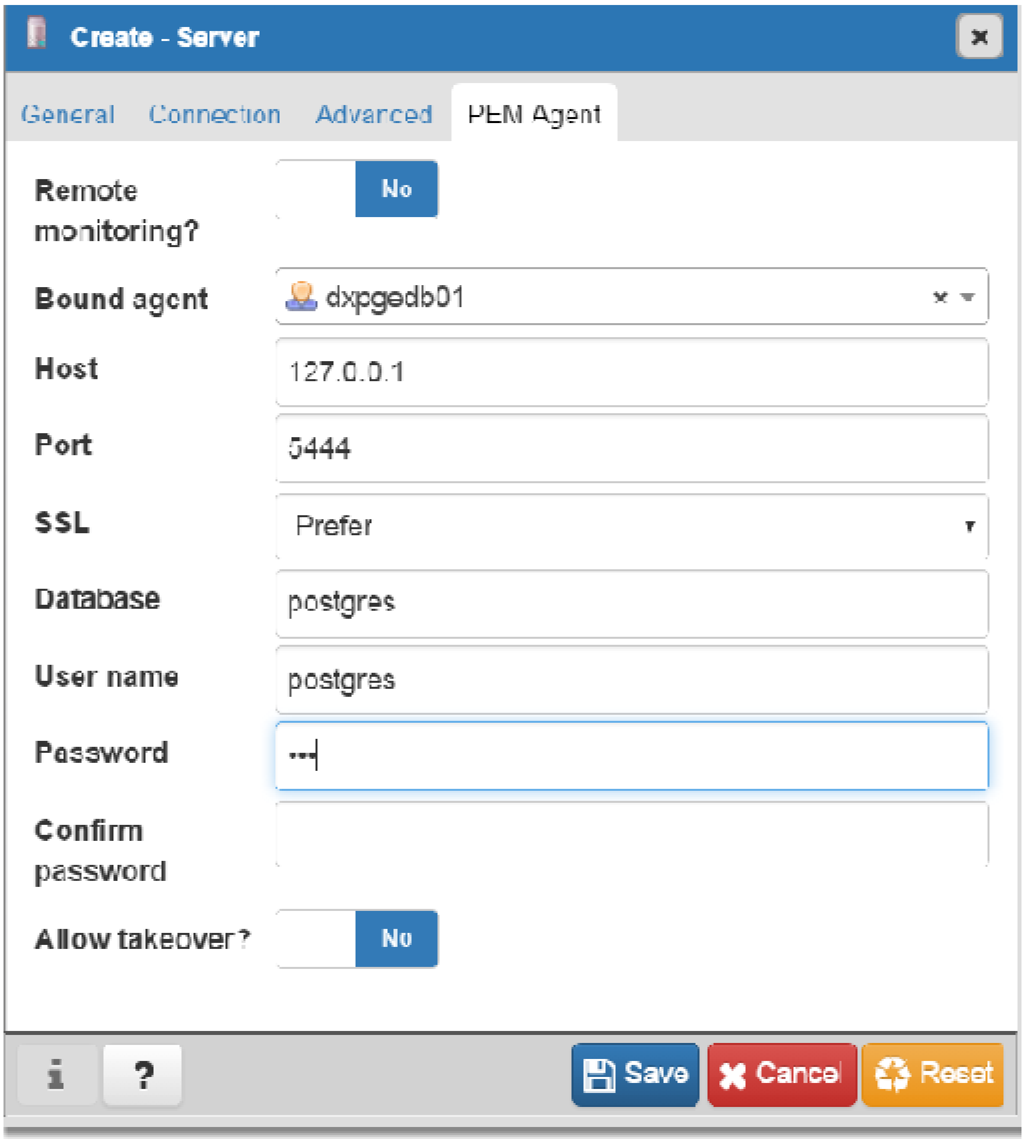
Hostname or address must be specified.

Enter managed server connection information.



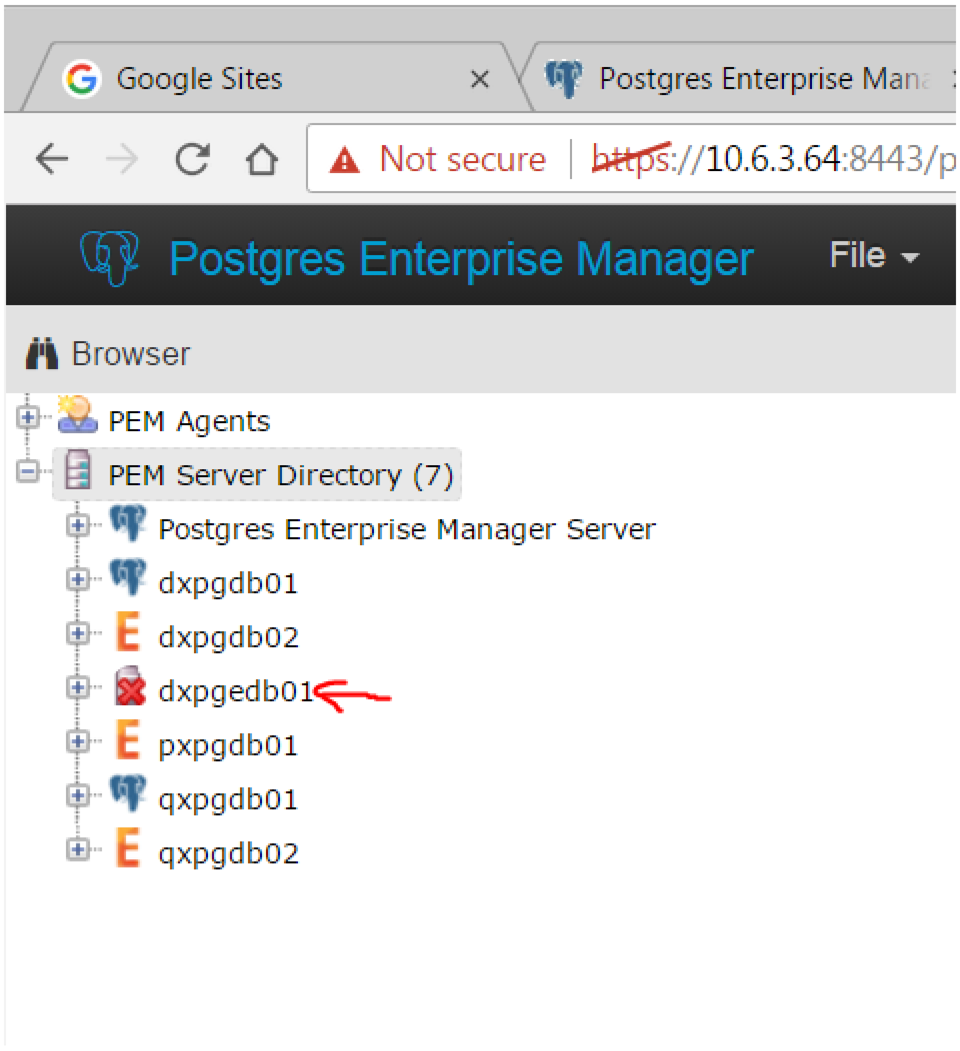
Host name	dxpgedb01
Port	5444
Maintenance database	postgres
Username	postgres
Password	•••
Save password?	☐
Role	
SSL	Prefer

Choose binding pemagent from drop list of Bound agent and enter pass word of db user. Then click save

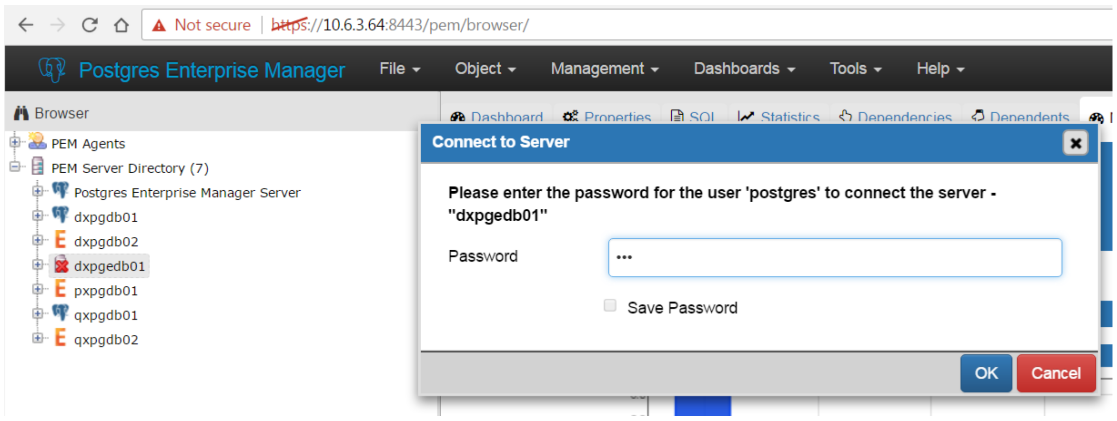


Remote monitoring?	No
Bound agent	dxpgedb01
Host	127.0.0.1
Port	5444
SSL	Prefer
Database	postgres
User name	postgres
Password	•••
Confirm password	
Allow takeover?	No

New server is added to PME as below marked with red cross



Click new added server and enter pass of db user from pop window



Connect to Server

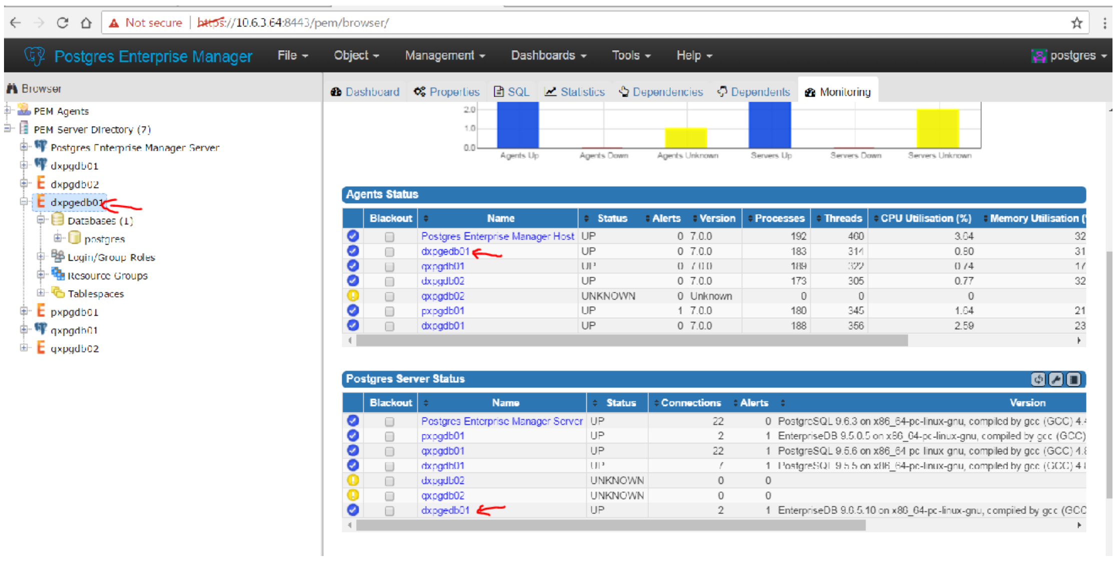
Please enter the password for the user 'postgres' to connect the server - "dxpgedb01"

Password: •••

Save Password

OK Cancel

Now, you new added server is accessed and managed by PEM server .



Agents Status

	Blackout	Name	Status	Alerts	Version	Processes	Threads	CPU Utilisation (%)	Memory Utilisation (
✓	☐	Postgres Enterprise Manager Host	UP	0	7.0.0	192	460	3.64	32
✓	☐	dxpgedb01	UP	0	7.0.0	183	314	0.80	31
✓	☐	qxpgdb01	UP	0	7.0.0	189	322	0.74	17
✓	☐	dxpgdb02	UP	0	7.0.0	173	305	0.77	32
!	☐	qxpgdb02	UNKNOWN	0	Unknown	0	0	0	
✓	☐	pxpgdb01	UP	1	7.0.0	180	345	1.64	21
✓	☐	dxpgdb01	UP	0	7.0.0	188	356	2.59	23

Postgres Server Status

	Blackout	Name	Status	Connections	Alerts	Version
✓	☐	Postgres Enterprise Manager Server	UP	22	0	PostgreSQL 9.6.3 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.4
✓	☐	pxpgdb01	UP	2	1	EnterpriseDB 9.5.0.5 on x86_64-pc-linux-gnu, compiled by gcc (GCC)
✓	☐	qxpgdb01	UP	22	1	PostgreSQL 9.5.6 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8
✓	☐	dxpgdb01	UP	7	1	PostgreSQL 9.5.5 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8
!	☐	dxpgdb02	UNKNOWN	0	0	
!	☐	qxpgdb02	UNKNOWN	0	0	
✓	☐	dxpgedb01	UP	2	1	EnterpriseDB 9.6.5.10 on x86_64-pc-linux-gnu, compiled by gcc (GCC

Performance Best practices

Operating System

To change the values of the following operating system parameters, the `/etc/sysctl.conf` file will need to be edited as root. The following lines should either be edited in the file, or added if not already present. For clarity in administration, it is recommended to create a `postgres.conf` (or similarly named) file in `/etc/sysctl.d` for kernel parameters being tuned specifically for Postgres.

Note: *that editing `/etc/sysctl.conf` (or adding `*.conf` files to `/etc/sysctl.d`) will cause the changes to take effect when the operating system next boots.*

In order to also apply changes immediately, it will be necessary to run the following as root:

```
# sysctl -p /etc/sysctl.d/postgres.con
```

1.

   ```
   kernel.sched_migration_cost = 5000000
   kernel.sched_migration_cost_ns = 5000000
   ```

   Since PostgreSQL is fork-based (not thread-based), the cost of context switching under a large number of processes (say, a high connection count) is significant. As such, EDB recommends telling the kernel that they are more expensive, so it tries harder to avoid them if possible.
   Note *that this tunable is called kernel.sched_migration_cost for kernels before 3.11 and kernel.sched_migration_cost_ns for kernel 3.11 or later.*

2.

   ```
   kernel.sched_autogroup_enabled = 0
   ```

   This setting groups tasks by TTY, to improve perceived responsiveness on an interactive system. On a server with a long running forking daemon, this will tend to keep child processes from migrating away as soon as they should. Since Postgres fits this model, EDB recommends disabling this feature by setting it to 0.

3.

   ```
   vm.dirty_background_bytes = 134217728
   vm.dirty_ratio = 90
   ```

`vm.dirty_background_bytes` contains the number of bytes at which the background kernel flusher threads will start writing out dirty data whereas vm.dirty_ratio contains, as a percentage of total available memory that contains free pages and reclaimable pages, the number of pages at which a process which is generating disk writes will itself start writing out dirty data. In tuning these two parameters, it is ideal to set vm.dirty_background_bytes very low (EDB recommends 134217728 and vm.dirty_ratio very high (EDB recommends 90%) as we want to coerce the kernel into doing asynchronous background flushing of dirty memory for us instead of forcing the Postgres processes to flush their own buffers in the foreground.

4.

```
vm.dirty_background_ratio = 0
vm.dirty_bytes = 0
```

EDB recommends setting these values to 0 and using their counterparts `vm.dirty_background_bytes and vm.dirty_ratio` to more precisely control when the pdflush daemon kicks in

5.

```
vm.dirty_expire_centisecs = 499
```

This tunable is used to define when dirty data is old enough to be eligible for write out by the kernel pdflush threads. It is expressed in 100'ths of a second. Data which has been dirty in-memory for longer than this interval will be written out next time a pdflush thread wakes up.

6.

```
vm.overcommit_memory = 2
```

Linux allows processes to allocate more memory than exists in the system under the assumption that not all of it will actually be used. Although this parameter tends to be set to its default of 0, which prevents excessive memory overcommit (requesting more memory than is available), overcommit is still possible. On a database server, it's better to set the value to 2, which disables this behavior.

7.

```
vm.swappiness = 1
```

This typically defaults to 60. But on a database server this is undesirable, as it will result in a great deal of disk swapping if the server is running low on memory, rather than reducing the buffer cache. More predictable performance is obtained by setting a preference for shrinking the file system cache rather than using swap. It is therefore recommended that this be set to 1.

8.

```
vm.zone_reclaim_mode = 0
```

The default for this varies from system to system. But on a database server this should always be set to 0, as otherwise it won't be using all available RAM for caching, and it can throttle writes.

9.

```
transparent_hugepage.defrag
```

Transparent huge pages de-fragmentation can lead to unpredictable database stalls on some Linux kernels. For Postgres 9.4.x and above, we recommend setting this parameter to madvise. This is applied by adding the following to /etc/rc.local:

Resource Usage

When tuning a Postgres cluster for peak performance on a specific server configuration for a given application load, efficient use of memory, disk i/o, and various other system resources is key. The parameters provided below are general best practices for deploying new Postgres databases at ASCAP. Further tuning requires observing the physical characteristics of the server and observing the load generated by the application.

- **`bgwriter_delay = 100ms`**

  On systems with a high volume of writes, the default value of 200 ms is too high and should be reduced. On very high frequency systems, positive results may be seen when taking this value as low as 10 ms. This should be approached in a stepwise fashion, however, with benchmarking as appropriate.

- **`bgwriter_lru_maxpages = 1000`**

  The maximum number of dirty buffers that will be written by the background writer process in a given LRU sweep. Increasing this value to 1000 should allow more buffers to be written at once.

- **`bgwriter_lru_multiplier = 4`**

  The number of dirty buffers actually written in the last sweep is multiplied by this value in order to determine how many will potentially be dirtied before the next sweep. Increasing this value to 4 from 2 will make it more likely that the buffers are cleaned ahead of time.

- **`maintenance_work_mem = 15% of RAM divided by autovacuum_max_workers`**

This determines the maximum amount of memory used for maintenance operations like `VACUUM, CREATE INDEX, ALTER TABLE ADD FOREIGN KEY`, and data-loading operations. These may increase the I/O on the database servers while performing such activities, so allocating more memory to them may lead to these operations finishing more quickly. EDB recommends 15% of RAM divided by `autovacuum_max_workers`.

- **`shared_buffers = 25% of available RAM up to 8GB`**

  This sets the number of shared memory buffers used by the database server. A general guideline is to set this parameter to 25% of available RAM with a maximum of 8GB. The size of the database cluster and amount of active data is also taken into consideration when tuning this parameter.

  ***Note** that your system will need to have shmmax set large enough to accommodate this setting, otherwise the database cluster will fail to start (see also Kernel Parameters).*

- **`vacuum_cost_limit = 800`**

  The accumulated cost that will cause the vacuuming process to sleep. The default value of -1 for `autovacuum_vacuum_cost_limit` *`(see below)`* results in `autovacuum_vacuum_cost_limit = vacuum_cost_limit`. However, this value is distributed proportionally among the running autovacuum workers. This is done in order that the sum of the limits of each worker never exceeds the limit on this variable. Therefore, the default value of 200 for `vacuum_cost_limit` is generally too low for a busy database server with multiple autovacuum workers.

- **`work_mem = 25% of the total system memory by max_connections`**

  The work_mem setting controls the amount of memory that can be used per query operation node, and is generally used for sorting data and hash tables. Any sort or hash operations which require more than this amount of memory will resort to swapping to disk, and will therefore take longer to complete. work_mem is probably the most difficult tuning parameter to right size that has a huge impact on performance.

  This recommendation should be viewed a starting point. The key is to monitor queries at ASCAP to see if they go to disk and require an external merge Disk. If that behavior is observed, work_mem will need to be adjusted or the query will need to be further optimized.

  `work_mem` can also be set dynamically from the application. If it is known that a particular query will be sorting a large data set, it can be advantageous to increase the work_mem of that individual query (being careful not to over-allocate). For example, if the session requires `1GB` for `work_mem`, this would be accomplished with:

```
set work_mem = '1GB';
```

This would set the session's `work_mem` to a new value and allow subsequent queries within that session to utilise more memory for sorting or hashing. This can be set back to the standard value (as defined in `postgresql.conf`) by running the following:

```
reset work_mem;
```

Write-Ahead Log

- **`archive_mode = on`**
  When archive_mode is enabled, completed WAL segments are sent to archive storage by the setting archive_command. It is strongly recommended that any production server have archive_mode turned on in order to enable the archiving of WAL logs for backup & recovery, point in time recovery, etc..

- **`archive_command`**
  Consider using **rsync** rather than scp or cp for archiving of WAL logs. cp is not an atomic operation and can lead to corruption that will not be noticed until attempting to use an archived log for recovery. Also, EDB Postgres comes with tool, `pg_receivexlog,` for archiving WAL files. For more information, please refer following link:
  https://www.postgresql.org/docs/9.6/static/app-pgreceivexlog.html

- **`checkpoint_completion_target = 0.9`**
  This determines the amount of time in which PostgreSQL aims to complete a checkpoint. This means a checkpoint need not result in an I/O spike, and instead aims to spread the writes over a certain period of time. This time is calculated as a fraction of checkpoint_timeout, and the default is 0.5 (half). The higher the value, the smoother the I/O of the checkpoint.
  ***Note** that each checkpoint finishes with a (potentially) large fsync and that the time consumed by this fsync can vary wildly. As such, it is not advisable to set this higher than 0.9.*

- **`checkpoint_timeout = 10min`**
  Maximum time between automatic WAL checkpoints, in seconds. The default is five minutes (5min). Increasing this parameter can increase the amount of time needed for crash recovery. The more checkpoint_segments specified, the more time will be needed to complete a checkpoint. A value in the 5 min to 15 min range is most often adequate.

- **`checkpoint_warning = 1min`**
  Controls when the database should log information about checkpoints occurring too frequently into server logs. Because a checkpoint_timeout of 10min is being recommended, this parameter should be set to less than checkpoint_timeout to capture checkpoints being forced for reasons other than the timeout value being reached.

- **`wal_buffers = 32MB`**
  This controls the amount of memory space available for back ends to place WAL data prior to sync. WAL segments are 16MB each, so buffering a segment is very inexpensive memory-wise. And larger buffer sizes have been observed to have a potentially very positive effect on performance in testing.

- **`wal_log_hints = on`**
  When this parameter is on, the EDB Postgres server writes the entire content of each disk page to WAL during the first modification of that page after a checkpoint, even for non-critical modifications of so-called hint bits.

- **`wal_compression = on`**
  When this parameter is on, the EDB Postgres server compresses a full page image written to WAL when full_page_writes is on or during a base backup. A compressed page image will be decompressed during WAL replay. The default value is off. Only superusers can change this setting.

  Turning this parameter on can reduce the WAL volume without increasing the risk of unrecoverable data corruption, but at the cost of some extra CPU spent on the compression during WAL logging and on the decompression during WAL replay.

- **`wal_level = logical`**
  This controls the level of information that is written to the write-ahead log. The default of minimal is only sufficient for crash recovery, and cannot be used to restore backups or perform. In order to restore from a base backup, this should be set to at least archive. EDB recommends to set the `wal_level` to `logical` due to following reason:
  1. It will help in point in time recovery
  2. will help in adding streaming replicas based on high availability requirement of the servers
  3. will help in creating logical standbys in future upgrade.

- **`max_wal_size = 8GB`**
  Maximum size to let the WAL grow to between automatic WAL checkpoints. This is a soft limit; WAL size can exceed max_wal_size under special circumstances, like under heavy load, a failing archive_command, or a high wal_keep_segments setting.

- **`min_wal_size = 4GB`**
  As long as WAL disk usage stays below this setting, old WAL files are always recycled for future use at a checkpoint, rather than removed. This can be used to ensure that enough WAL space is reserved to handle spikes in WAL usage, for example when running large batch jobs. The default is 80 MB. This parameter can only be set in the postgresql.conf file or on the server command line.

Query Tuning

- **`cpu_tuple_cost = 0.03`**

Specifies the relative cost of processing each row during a query. It is currently set to 0.01, but this is likely to be lower than optimal, and should be increased to 0.03 for a more realistic costing.

- **`effective_cache_size = 75% of RAM`**
  While this parameter does not actually reserve any memory, it is used by PostgreSQL in planning queries. It indicates how much memory the operating system has available for caching data. Adjusting this value to 75% of RAM is recommended.

- **`random_page_cost = 2`**
  This parameter specifies the relative cost of accessing a random page instead of a sequential page. The default is set to 4, but this value is conservative and should be tuned downwards.

- **`max_parallel_workers_per_gather = 4`**
  This parameter controls the number of workers that can assist in a sequential scan (parallel query). Adding 4 workers is a reasonable starting point for the average system, but this value should be increased up to 8 when running Postgres on high CPU count servers.

Autovacuum

Vacuum and analyze operations should be performed periodically to clean up dead tuples (caused by deletions or updates) occupying database pages. This allows the space to be reused for new insertions and collecting statistics that enable the optimizer to choose the best possible query plans. If these run too infrequently, the database cluster can become bloated or provide inefficient query plans. If run too frequently, they can adversely affect system I/O activity.

High-activity tables, where a large portion of the table changes frequently, will need vacuuming and analyzing more frequently than a table that doesn't change much, or consists of mainly new data being appended.

- **`autovacuum = on`**
  It is critically important that autovacuum be active on any database, even more so on an active production database. Turning off the autovacuum process will inevitably lead to poor performance, table and index bloat, and potentially catastrophic data loss due to transaction ID wraparound. While manual vacuum jobs are sometimes used in addition to autovacuum, they should not be used in place of autovacuum.

- **`autovacuum_vacuum_cost_delay = 10ms`**
  The default of 20ms is very conservative and can prevent vacuum from keeping up with changes. Decreasing it's value to 10ms is expected to yield increased performance. With testing, it is not at all uncommon to find performance gains while reducing this to as low as 2ms.

- **`autovacuum_max_workers = 3`**

Often set in the range of 3-6 for busy production systems, start with the default and adjust upward as needed.

- **`autovacuum_naptime = 5`**
  Naptime specifies the minimum delay between autovacuum runs on any given database. In each round the daemon examines the database and issues VACUUM and ANALYZE commands as needed for tables in that database. The delay is measured in seconds, and the default is one minute (1min). On busy databases with many writes, however, it can be beneficial to increase this to prevent autovacuum waking up too often. But it should be noted that tuning this setting too high can result in more work needed to be done in each vacuuming round.

  Note: *As this setting determines the wake up time per database, an autovacuum worker process will begin as frequently as autovacuum_naptime / number of databases. For example, if autovacuum_naptime = 1min (60 seconds), and there were 60 databases, an autovacuum worker process would be started every second (60 seconds / 60 databases = 1 second).*

- **`autovacuum_vacuum_scale_factor = 0.05`**
  `autovacuum_analyze_scale_factor = 0.05`
  These parameters determine the percentage of a table that must change before the table will be scheduled for an autovacuum, and an autoanalyze respectively. The default for the autovacuum_vacuum_scale_factor is 0.2 (20%) and autovacuum_analyze_scale_factor is 0.1 (10%). While the default values perform acceptably for tables of a modest size (up to around 500MB), for larger tables they are too high.

- **`autovacuum_vacuum_cost_limit = -1`**
  `vacuum_cost_limit = 800`
  The default value of -1 for autovacuum_vacuum_cost_limit results in autovacuum_vacuum_cost_limit = vacuum_cost_limit. However, this value is distributed proportionally among the running autovacuum workers. This is done in order that the sum of the limits of each worker never exceeds the limit on this variable. Therefore, the default value of 200 for vacuum_cost_limit is generally too low for a busy database server with multiple autovacuum workers.

Logging

The parameters below should be applied in order to provide greater insight into the operations and performance of the database. It is recommended that all be in place on any Postgres server so as to aid the database administrator in maintaining and troubleshooting the system. As raw text, these logs are very helpful to any DBA. But when utilizing a log analysis tool such as Postgres Enterprise Manager (PEM), tremendous insights into the operational health of the database, configuration settings potentially in need of adjustment, and queries that are likely in need of tuning.

- **`log_line_prefix = '%t [%p]: [%l-1] user=%u,db=%d '`**

Provides greater detail for each log event, as well as provides for compatibility with various log analysis tools.

- **`log_min_duration_statement = 1000`**
  Log any statements with a duration of greater than 1000 ms.

  Note: *log_min_duration_statement = 0 will log all statements run on the server and can cause tremendous, rapid log file growth. It may be adjusted upward (in millisecond units) should it prove to be too much detail.*

- **`log_autovacuum_min_duration = 0`**
  Gives visibility into what autovacuum is doing (or possibly not doing).

- **`log_checkpoints = on`**
  Provides visibility into how frequently checkpoints occur and how much each checkpoint does.

- **`log_lock_waits = on`**
  Logs lock waits >= deadlock_timeout.

- **`log_temp_files = 0`**
  Logs temporary files equal or larger than the specified size (in kB). This provides insight into whether work_mem is undersized or not. Also exposes malformed queries utilizing temp files that could instead be rewritten for more efficient use of indexes, in-memory sorting, etc..

Other EDB Postgres parameter recommendation for creating template postgresql.conf

Following are the list of parameters we recommend to have in postgresql.conf file of EDB Postgres, which you can consider as template parameters.

- *`hot_standby = on`*

  This parameter specifies whether or not user will be able to connect and run queries during recovery, the default value is off. This parameter gets ignored on the primary since it is always in read/write mode.
- *`wal_level = logical`*

  wal_level parameter determines how much information is written to the WAL. Possible values of this parameter are:
  - `minimal` (default): This value writes only the information needed to recover from a crash or immediate shutdown.
  - `replica`: adds logging required for WAL archiving as well as information required to run read-only queries on a standby server
  - `logical`: This value adds information necessary to support logical decoding.

  Please note, each level includes the information logged at all lower levels. This parameter can only be set at server start.

- *`max_wal_senders = 10`*

  This parameter specifies the maximum number of concurrent connections from standby servers or streaming base backup clients (i.e., the maximum number of simultaneously running WAL sender processes). The default is zero, meaning replication is disabled. WAL sender processes count towards the total number of connections, so the parameter cannot be set higher than max_connections. Abrupt streaming client disconnection might cause an orphaned connection slot until a timeout is reached, so this parameter should be set slightly higher than the maximum number of expected clients so disconnected clients can immediately reconnect. This parameter can only be set at server start.

- *`archive_mode = on`*

  When `archive_mode` is enabled, completed WAL segments are sent to archive storage by setting archive_command. This parameter can take following values:
  - `off`: This value disable the archiving of completed WAL segments.
  - `on`: This value enable the archiving of completed WAL segments using archive_command/
  - `always`: There is no difference between the on & always, but when set to always the WAL archiver is enabled also during archive recovery or standby mode.

  In `always` mode, all files restored from the archive or streamed with streaming replication will be archived.
  `archive_mode` and `archive_command` are separate parameters so that `archive_command` can be changed without leaving archiving mode. This parameter can only be set at server start. `archive_mode` cannot be enabled when `wal_level` is set to minimal.

- *`archive_command = 'rsync -a %p <location of archive directory>/%f'`*
  This parameter is ignored unless archive_mode was enabled at server start. Using this parameter completed WAL segments will be archived.

- *`wal_log_hints = on (default off)`*

  When this parameter is on, the server writes the entire content of each disk page to WAL during the first modification of that page after a checkpoint, even for non-critical modifications of so-called hint bits.

- *`max_replication_slots = 10`*

  This parameter specifies the maximum number of replication slots that the server can support. The default is zero. This parameter can only be set at server start.

- `full_page_writes = on`

  When this parameter is on, the PostgreSQL server writes the entire content of each disk page to WAL during the first modification of that page after a checkpoint. This is needed because a page write that is in process during an operating system crash might be only partially completed, leading to an on-disk page that contains a mix of old and new data. The row-level change data normally stored in WAL will not be enough to completely restore such a page during post-crash recovery. Storing the full page image guarantees that the page can be correctly restored, but at the price of increasing the amount of data that must be written to WAL. (Because WAL replay always starts from a checkpoint, it is sufficient to do this during the first change of each page after a checkpoint. Therefore, one way to reduce the cost of full-page writes is to increase the checkpoint interval parameters.)

Template recovery configuration parameters for EDB Postgres standby.

For streaming replication in EDB Postgres, users are recommended to include following parameters in the recovery.conf file in the Postgres Data directory:

1. `standby_mode = on`

This parameter specifies whether to start the EDB Postgres server as a standby. If this parameter is on, the server will not stop recovery when the end of archived WAL is reached, but will keep trying to continue recovery by fetching new WAL segments using restore_command and/or by connecting to the primary server as specified by the primary_conninfo setting.

2. `primary_conninfo = 'primary connection info with application_name=<name of standby>'`

This parameter specifies a connection string to be used for the standby server to connect with the primary. This string is in the following format mentioned in following link:
http://www.postgresql.org/docs/9.6/static/libpq-connect.html#LIBPQ-CONNSTRING

If any option is unspecified in this string, then the corresponding environment variable will be used. For more information on environment variables, please refer following link:
http://www.postgresql.org/docs/9.6/static/libpq-envars.html

3. `recovery_target_timeline = 'latest'`

This parameter specifies recovering into a particular timeline. The default is to recover along the same timeline that was current when the base backup was taken. Setting this to latest recovers to the latest timeline found in the archive, which is useful in a standby server.

4. `primary_slot_name = '<slot name>'`

This parameter specifies an existing replication slot to be used when connecting to the primary via streaming replication to control resource removal on the upstream node

Note: This setting has no effect if primary_conninfo is not set.

Before EDB postgres 9.4, the master doesn't make any effort to retain extra WAL for a standby that's fallen behind. If the standby gets too far behind, the master will delete WAL segments the standby still needs to replay, and the standby cannot recover. To prevent this, user were expected to configure continuous archiving and provide a restore_command to give the replica access to the archive, or be willing to re-create a standby that falls behind.
A less reliable alternative was to set wal_keep_segments to a "high enough" value to ensure that the replica never falls too far behind. However, there's no real way to guess what is high enough. So generally archiving is preferred. From 9.4 onwards, physical replication slot will make master to keep the WAL segments till standby catch up with all WAL segments. With replication_slot, we will also be using `restore_command` for recovery. `restore_command` in `recovery.conf` file gives a fail safe option for a situation, where primary crashed due to some reason and standby not able to receive WAL segments from master

5. `trigger_file = '<trigger file>'`

This parameter specifies a trigger file whose presence ends recovery in the standby.

6. `restore_command = 'rsync -a <location of archive command>/%f %p'`

This parameter is required for archive recovery, but optional for streaming replication. Any `%f` in the string is replaced by the name of the file to retrieve from the archive, and any `%p` is replaced by the copy destination path name on the server.

It is important for the command to return a zero exit status only if it succeeds. The command will be asked for file names that are not present in the archive; it must return non zero when so asked.

Filesystem Types

XFS/EXT4/EXT3

These general purpose file systems are the most common in use on Linux database servers. However, as "general purpose", they come with some default options that may not be suitable for hosting a database:

- **`noatime`**
  Should be set for the mount point used for the cluster's file-system so as to remove the overhead of maintaining access time updates to a file's meta-data, thereby improving performance.

- **`nobarrier`**
  If a storage system with a battery-backed non-volatile write cache is being used, and the filesystem being used is EXT3/EXT4 or XFS, consider using this option to prevent unnecessary flushing of write cache to disk.
- **`discard`**
  If SSD drives are in use, this attribute should be set to ensure that TRIM support is used, which will prevent performance degradation over time.

***Note:** There have been reports of data loss when using RAID 0 and TRIM enabled on Linux kernels 3.19.7+ or 4.0.2+. At this time EDB suggested that TRIM be disabled when using RAID 0. See bugzilla.kernel.org/show_bug.cgi?id=98501 for more information.*

EXT2

For WAL files (transaction logs), there is no need to use an EXT3/4 filesystem. Use of EXT2 (or another non-journaling file system) is sufficient since data is only written sequentially and never read (unless in recovery mode).

Utilizing a different mount point for pg_xlog with EXT2 file system type is recommended in order to avoid unnecessary journaling.

TMPFS/RAMFS

Using RAMFS/TMPFS, a portion of the physical memory of the server can be allocated to be used as a partition. Two primary examples of architectural changes that increase performance by taking advantage of the speed of a in-memory file system being:

- The statistics collector shares data statistics with other processes through temporary files. These files are stored in the directory named by the stats_temp_directory parameter, $PGDATA/pg_stat_tmp by default. For better performance, stats_temp_directory can be pointed at a RAM-based file system, decreasing physical I/O requirements. When the server shuts down, a permanent copy of the statistics data is stored in the pg_stat subdirectory, so that statistics can be retained across server restarts.

- A data warehouse, reporting server, or simply a database which makes significant use of temporary tables/tablespaces for many sorts and joins, can benefit greatly from the use of a RAM-based file system.