# 安装Linux系统

1.下载CentOS： https://developer.aliyun.com/mirror/

2.安装CentOS-7

分区方案：标准分区

LVM(逻辑卷管理器)是一个软件实现磁盘动态调整的工具，即在硬盘上面增加了一层管理软件，问题在于高并发性能下降5-10%。

自定义分区：

/boot：256MB

swap分区：16GB，又称交换分区或虚拟分区（也就是相当于我们平时所说的运行内存）。一般都是给到内存的2倍，建议最大不超过20GB，否则就要浪费很大的空间。

/根分区：80GB

/home：所有都给

# 安装PostgreSQL

1.下载源码软件包：https://www.postgresql.org/ftp/source/v14.5/

yum -y install wget bzip2

wget https://ftp.postgresql.org/pub/source/v14.5/postgresql-14.5.tar.bz2

tar xvf postgresql-14.5.tar.bz2

2.安装依赖软件

yum -y install zlib-devel readline-devel libxml2-devel libxslt-devel openssl-devel perl-devel perl-ExtUtils-Embed python-devel gcc-c++

* zlib-devel ：备份时使用的压缩功能
* readline-devel ：在psql中使用上下方向键把历史命令找出来
* libxml2-devel ：使用xml数据类型
* libxslt-devel：使用libxslt
* openssl-devel：支持使用SSL连接加密
* perl-devel：使用Perl语言来开发
* python-devel ：使用Python语言来开发

3.编译源码

cd postgresql-14.5

./configure --prefix=/usr/local/pgsql14.5 --with-perl --with-python --with-libxml --with-libxslt --with-openssl [--with-blocksize=32] [--with-wal-blocksize=32] [--with-wal-segsize=64]

* --prefix ：指定安装目录
* --with-perl： 使用Perl语言来编写自定义函数，使用该项要先安装perl开发包(perl-devel)
* --with-python：使用Python语言来编写自定义函数，使用该选项要先安装python-dev开发包(python-devel)
* --with-libxml ：使用xml数据类型，使用该选项要先安装python-dev开发包(libxml2-devel)
* --with-libxslt ：使用libxslt构建，启用xml2模块从而可以从xml到xsl的转换。

PostgreSQL在数据仓库使用场景中， 较大的数据块以提高I/O性能。

* --with-blocksize：指定数据块为32KB，默认是8KB
* --with-wal-blocksize：指定WAL日志块为32KB，默认为8KB
* --with-wal-segsize： 指定WAL日志文件为64MB，默认是16MB

make

make install

cd /usr/local

ln -s /usr/local/pgsql14.5 /usr/local/pgsql

# 初始化数据库

1.添加用户和组：

groupadd -g 1001 postgres添加一个组

useradd -g 1001 -u 1001 -m postgres添加一个用户

passwd oracle设置密码

2.配置环境变量

pgdata：数据目录

pghost：数据库主机或socket目录(默认："本地接口")

pgport：数据库服务器的端口(默认："5432")

pgdatabase： 指定要连接的数据库 (默认：“postgres")

pguser：指定数据库用户名(默认：“postgres")

vi .bashrc对当前用户生效

export PATH=/usr/local/pgsql/bin:$PATH

export LD\_LIBRARY\_PATH=/usr/local/pgsql/lib

export PGDATA=/home/postgres/pgdata/

source .bashrc

vi /etc/profile对所有用户生效

export PATH=/usr/local/pgsql/bin:$PATH

export LD\_LIBRARY\_PATH=/usr/local/pgsql/lib

3.PostgreSQL数据库初始化

initdb -D datadir -E encoding --locale LOCALE -W [-k]

-D 指定数据目录，必选项

-E 设置数据库的默认编码，实际是设置template1的编码

--locale 设置区域，即设置默认语言环境， locale就是某一个地域内的人们的语言习惯和文化传统和生活习惯。

-W 在initdb过程中，为超级用户设置一个密码

-k 使用数据页产生效验和

checksum 是针对数据可靠性很高的场景(金融)，开启后数据库会产生更多的WAL日志

pg\_checksums 查看是否打开checksum功能

pg\_checksums -e -P 启用checksums功能

show data\_directory;查看数据目录

4.编译插件

cd postgresql-14.5/contrib

make

make install

# 启动、停止数据库

pg\_ctl是一个实用工具， 它具有以下功能：

* 初始化PostgreSQL
* 启动、 停止或重启PostgreSQL
* 查看PostgreSQL状态
* 让数据库重新加载配置文件
* 允许给一个指定的进程发送信号
* 在Windows平台下为数据库注册或取消服务

**1.启动数据库**

pg\_ctl start -D 数据目录

postgres -D 数据目录 or postmaster -D 数据目录

pg\_ctl restart 重启数据库

**2.停止数据库**

pg\_ctl stop -D 数据目录 [-m smart|fast(默认)|immediate]

* smart：等所有连接中止后， 关闭数据库。 如果客户端连接不终止， 则无法关闭数据库。
* fast：快速关闭数据库， 断开客户端的连接， 让已有的事务回滚， 然后正常关闭数据库。
* immediate：立即关闭数据库， 相当于数据库进程立即停止， 直接退出， 下次启动数据库需要进行恢复。

实际上在关闭数据库的时候是直接向数据库的主进程发送signal信号，有以下3种类型：

* SIGTERM： 发送此信号为Smart Shutdown关机模式。
* SIGINT： 发送此信号为Fast Shutdown关机模式。
* SIGQUIT： 发送此信号为Immediate Shutdown关机模式。

pg\_ctl kill TERM|INT|QUIT pid

kill -sigterm pid

3.查看数据库状态

pg\_ctl status -D datadir

psql -h -p -d –U

pg\_isready

4.设置开机启动

su - root

cd /root/postgresql-14.5/contrib/start-scripts

cp linux /etc/init.d/postgres

chmod +x /etc/init.d/postgres

vi /etc/init.d/postgres

PGDATA="/home/postgres/pgdata"

service postgres start|stop|restart

chkconfig --list

chkconfig postgres on/off

# 修改配置文件(pg\_hba.conf)

pg\_hba.conf：基于主机的认证 ，HBA是“Host-Based Authentication”的缩写。

数据库创建后是无法接受远程连接的， 因为在pg\_hba.conf中没有相应的配置项。

配置文件里有多条匹配记录，记录匹配方式为从上往下进行匹配，如果匹配成功，将不再匹配后面的记录；如果匹配失败， 也不再考虑后面的记录。 如果没有匹配的记录， 访问将被拒绝。

记录格式：连接类型、数据库名、用户名、 IP地址范围、 认证方法--针对远程连接

记录格式：连接类型、数据库名、用户名、 认证方法--针对本地连接

**连接类型：**

local： 通过unix socket连接，当psql后面不指定IP地址时， 即使用这种方式连接。

host：通过TCP/IP进行的连接， 包括SSL和非SSL的连接，启用远程连接需要修改 postgresql.conf 中的 listen\_addresses 参数。

hostssl： 使用TCP/IP的SSL方式连接，前提是编译时要打开SSL，启动服务器时必须打开SSL配置选项。

hostnossl： 与hostssl相反， 只匹配在TCP/IP上不使用SSL的连接请求。

hostgssenc ：使用TCP/IP的GSSAPI 方式连接，同hostssl。

hostnogssenc ：与hostgssenc 相反，同hostnossl。

**数据库名：**

all：任何数据库

replication：允许流复制连接， 而不是名为“replication”的数据库。

**用户名：**

all：任何用户。

**IP地址范围：**

192.168.1.100/32：表示192.168.1.10这台主机

192.168.1.0/24：表示192.168.1.0这个网段

0.0.0.0/0：表示所有网络

0/0：表示所有网络

**认证方法：**

trust, reject, scram-sha-256, peer, ident, md5, password, gss, sspi, pam, ldap, radius ,cert

trust：不需要密码就可进行连接。

reject： 拒绝连接，相当于数据库的黑名单。

**基于密码的身份验证**：password、md5、scram-sha-256。

* password:要求客户端以明文形式发送密码，因此很容易受到密码嗅探攻击。一般这种方法使用得很少。
* md5:要求客户端提供一个MD5加密的口令进行认证。目前md5现在被设为是不安全的认证方法。
* scram-sha-256 ：要求客户端提供一个scram-sha-256加密的口令进行认证。 目前被认为是安全的加密机制。

**基于操作系统用户认证**：peer与ident

**peer认证**：通过操作系统用户映射数据库用户进行认证，客户端与数据库服务器必须在同一台操作系统上。这种方法只在本地连接上支持。

pg\_hba.conf

local all all peer map=testmap

pg\_ident.conf

testmap os\_user postgres

**ident认证**：通过操作系统用户映射数据库用户进行认证，客户端与数据库服务器不需要在同一台操作系统上。需要配置pg\_ident.conf文件。这种方式只在TCP/IP连接上支持。客户端需要安装 oidentd 服务，在epel-release源里。

pg\_hba.conf

host all all 127.0.0.1/32 ident map=testmap

pg\_ident.conf

testmap os\_user postgres

gss： 用GSSAPI认证用户。 只有在进行TCP/IP连接时才能用。

sspi： 用SSPI来认证用户。 仅在Windows系统上使用。

pam： 用操作系统提供的可插入认证模块服务（PAM） 来认证。

ldap： 用LDAP服务器认证。

radius： 用RADIUS服务器认证。

cert： 用SSL客户端证书认证。

# 修改配置文件(postgresql.conf)

**修改配置文件方法：**

* 修改postgresql.conf
* ALTER SYSTEM SET configparameter { TO | = } { value | ‘value’ | DEFAULT } 通过alter system方式修改参数，会自动编辑 postgresql.auto.conf 文件，并用它的配置覆盖postgresql.conf的配置。

* ALTER SYSTEM RESET configparameter
* ALTER SYSTEM RESET ALL

例如：

alter system set work\_mem =8MB; #设置work\_mem为8MB

恢复work\_mem的默认值

* alter system set work\_mem = default;
* alter system set work\_mem to default;
* alter system reset work\_mem ;
* alter system reset all;

**修改局部配置方法：**

设置database级别的配置

ALTER DATABASE dbname SET configparameter { TO | = } { value | DEFAULT }

ALTER DATABASE dbname SET configparameter FROM CURRENT

ALTER DATABASE dbname RESET configparameter

ALTER DATABASE dbname RESET ALL

**设置user级别的配置：**

ALTER USER|ROLE {username | ALL } [ IN DATABASE dbname ] SET configparameter { TO | = } { value | DEFAULT }

ALTER USER|ROLE {username | ALL } [ IN DATABASE dbname ] SET configparameter FROM CURRENT

ALTER USER|ROLE {username | ALL } [ IN DATABASE dbname ] RESET configparameter

ALTER USER|ROLE {username | ALL } [ IN DATABASE dbname ] RESET ALL

**设置session级别的配置**

SET [ SESSION | LOCAL ] configparameter { TO | = } { value | ‘value' | DEFAULT }

SET [ SESSION | LOCAL ] TIME ZONE { 时区 | LOCAL | DEFAULT }

**更新 pg\_settings 视图**

update pg\_settings set setting=new\_value where name=‘configparameter’;

**使用 set\_config 函数修改session配置**

select set\_config(‘configparameter’,new\_value,false); 最后一个参数若为true，则新值将只应用于当前事务。若为false，和SQL语句SET是等效的。

**查看配置：**

* show configparameter|all;
* select name,setting from pg\_settings where name = ‘configparameter’; 精确查询
* select name,setting from pg\_settings where name like ‘%keyword%’; 模糊查询

**使配置生效：**

* pg\_ctl reload 不用登陆数据库
* select pg\_reload\_conf(); 需要登陆数据库