



新手机 新应用 新娱乐

PostgreSQL 9.3 培训

Day 4

digoal.zhou

2013/12/11

课程内容

- Day - 4
- 数据库安全
- 目标:
- PostgreSQL安全简介, 认证体系, 基于角色的权限管理, 事件触发器;
- 如何防范SQL注入

- 高可用,负载均衡
- 目标:
- 了解高可用架构, PostgreSQL高可用的实现方法, 挑选几种演示讲解
- 了解负载均衡的应用场景, PG的实现方法, 挑选几种演示讲解

- 数据库规划
- 目标:
- 根据业务形态, 合理规划PostgreSQL数据库硬件和操作系统.
- 如何建模以及压力测试.

数据库安全

- PostgreSQL安全简介, 认证, 基于角色的权限管理, 事件触发器
- 数据传输加密
- 审计日志
- 安全标签, se-postgres
- 约束
- 对象触发器: 权限限制, 数据追踪审计
- 视图
- 物化视图
- 如何防范SQL注入
- 给数据库打补丁

数据库认证

■ 密码存储的安全注意事项

- CREATE ROLE name [[WITH] option [...]]
 - | [ENCRYPTED | UNENCRYPTED] PASSWORD 'password'
 - ENCRYPTED 存储为MD5
 - UNENCRYPTED 存储为明文
-
- postgres=# alter role rep encrypted password '123';
 - ALTER ROLE
 - postgres=# select usename,passwd,md5('123rep') from pg_shadow where usename='rep';
 - usename | passwd | md5
 - -----+-----+-----
 - rep | md5a6f8eo1c46849ed88d55e3c347ec318a | a6f8eo1c46849ed88d55e3c347ec318a
 - (1 row)
 - 明文
 - postgres=# alter role rep unencrypted password '123';
 - ALTER ROLE
 - postgres=# select usename,passwd,md5('123rep') from pg_shadow where usename='rep';
 - usename | passwd | md5
 - -----+-----+-----
 - rep | 123 | a6f8eo1c46849ed88d55e3c347ec318a
 - (1 row)

数据库认证

■ 密码复杂度注意事项

- 确保已编译passwordcheck, 未编译的话到源码的contrib目录中编译一下
- pg93@db-172-16-3-150-> ll \$PGHOME/lib/passwordcheck*
- -rwxr-xr-x 1 root root 14K Oct 13 09:00 /home/pg93/pgsql/lib/passwordcheck.so
- 修改数据库配置文件
- vi \$PGDATA/postgresql.conf
- shared_preload_libraries = 'passwordcheck,pg_stat_statements,auto_explain'
- 重启数据库
- pg93@db-172-16-3-150-> pg_ctl restart -m fast
- 再次修改密码时, 如果密码太简单会报错
- digoal=# alter role rep unencrypted password '123';
- ERROR: password is too short
- digoal=# alter role rep unencrypted password 'rep1234567RER';
- ERROR: password must not contain user name
- digoal=# alter role rep unencrypted password 're1234567RER';
- ALTER ROLE

数据库认证

■ 密码更换周期注意事项

- postgres=# alter role rep VALID UNTIL '2013-12-18 00:09:07.549152';

- digoal=> \du+ rep

- ```
List of roles
```

| Role name | Attributes | Member of | Description |
|-----------|------------|-----------|-------------|
|-----------|------------|-----------|-------------|

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|--|--|--|--|

|     |             |       |  |
|-----|-------------|-------|--|
| rep | Replication | +  {} |  |
|-----|-------------|-------|--|

|  |                |   |  |
|--|----------------|---|--|
|  | 32 connections | + |  |
|--|----------------|---|--|

|  |                                                    |  |  |
|--|----------------------------------------------------|--|--|
|  | Password valid until 2013-12-18 00:09:07.549152+08 |  |  |
|--|----------------------------------------------------|--|--|

- 密码到期后, 将无法认证通过

- pg93@db-172-16-3-150-> psql -h 172.16.3.150 -U rep digoal

- Password for user rep:

- psql: FATAL: password authentication failed for user "rep"

# 数据库认证

- <http://www.postgresql.org/docs/9.3/static/auth-pg-hba-conf.html>
- TYPE DATABASE USER ADDRESS METHOD
- 认证类型(pg\_hba.conf - TYPE)
  - local 本地unix socket, host, hostssl, hostnoss ; (host支持hostssl和hostnoss两种模式), ssl表示网络传输的数据使用加密方式传输, 包括认证过程也加密.
- 认证方法(pg\_hba.conf - METHOD) 逐条匹配, 如果匹配到了则不需要往下匹配了.
  - trust -- 无需密码
  - reject -- 拒绝认证
  - md5 -- 校验过程密码加密传输, 但是其他数据是否加密传输要看配置的认证类型是否为SSL.
  - password -- 校验过度密码明文传输, 如果认证类型为SSL, 则同样会被加密.
  - gss -- 使用GSSAPI认证方法, 仅支持TCP/IP连接
  - sspi -- 使用SSPI认证方法, 仅支持windows
  - krb5 -- 使用Kerberos V5 认证方法, 仅支持TCP/IP连接
  - ident -- 客户端操作系统用户和数据库用户映射关系认证, 仅支持tcp/ip连接
  - peer -- 客户端操作系统用户和数据库用户映射关系认证, 仅支持unix socket连接
  - ldap -- 使用LDAP服务认证
  - radius -- 使用radius服务认证
  - cert -- 使用客户端的ssl身份认证.
  - pam -- 使用操作系统的PAM认证模块进行认证.

# 数据库认证

## ■ 密码方式认证过程的安全注意事项

- 在没有使用ssl连接类型的情况下，不要使用password认证方法，因为这种方法会传输明文密码，可能被截获.
- 密码认证的方式请使用md5

## ■ 密码方式认证如何防范暴力破解和类DDOS攻击

- 对于输错密码的情况，延迟反馈给用户密码错误的消息. 这样可以防止暴力破解和类DDoS攻击.
- <http://www.postgresql.org/docs/9.3/static/auth-delay.html>
- # postgresql.conf
- shared\_preload\_libraries = 'auth\_delay'
  
- auth\_delay.milliseconds = '500'

## ■ pg\_hba.conf的鉴权.

- 使用最小权限范围鉴权，尽量避免使用all, 0.0.0.0/0 这种大范围授权

# 基于角色的权限管理

- 表级别权限控制
- GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
  - [, ...] | ALL [ PRIVILEGES ] }
  - ON { [ TABLE ] table\_name [, ...]
    - | ALL TABLES IN SCHEMA schema\_name [, ...] }
  - TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 列级别权限控制
- GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column\_name [, ...] )
  - [, ...] | ALL [ PRIVILEGES ] ( column\_name [, ...] ) }
  - ON [ TABLE ] table\_name [, ...]
  - TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 序列权限控制
- GRANT { { USAGE | SELECT | UPDATE }
  - [, ...] | ALL [ PRIVILEGES ] }
  - ON { SEQUENCE sequence\_name [, ...]
    - | ALL SEQUENCES IN SCHEMA schema\_name [, ...] }
  - TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

# 基于角色的权限管理

- 数据库权限控制
- GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ] }
- ON DATABASE database\_name [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 类型域的权限控制(域简单来说就是自定义的带约束的数据类型)
- GRANT { USAGE | ALL [ PRIVILEGES ] }
- ON DOMAIN domain\_name [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- FDW权限控制
- GRANT { USAGE | ALL [ PRIVILEGES ] }
- ON FOREIGN DATA WRAPPER fdw\_name [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- FS权限控制
- GRANT { USAGE | ALL [ PRIVILEGES ] }
- ON FOREIGN SERVER server\_name [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

# 基于角色的权限管理

- 函数权限控制
- GRANT { EXECUTE | ALL [ PRIVILEGES ] }
- ON { FUNCTION function\_name ( [ [ argmode ] [ arg\_name ] arg\_type [, ...] ] ) [, ...]
- | ALL FUNCTIONS IN SCHEMA schema\_name [, ...] }
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 语言的权限控制, 如C语言默认只有超级用户可以使用.
- GRANT { USAGE | ALL [ PRIVILEGES ] }
- ON LANGUAGE lang\_name [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 大对象的权限控制
- GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
- ON LARGE OBJECT loid [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- SCHEMA的权限控制
- GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
- ON SCHEMA schema\_name [, ...]
- TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]

# 基于角色的权限管理

- 表空间的权限控制
  - GRANT { CREATE | ALL [ PRIVILEGES ] }
  - ON TABLESPACE tablespace\_name [, ...]
  - TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 数据类型的权限控制
  - GRANT { USAGE | ALL [ PRIVILEGES ] }
  - ON TYPE type\_name [, ...]
  - TO { [ GROUP ] role\_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
- 角色的权限控制
  - GRANT role\_name [, ...] TO role\_name [, ...] [ WITH ADMIN OPTION ]

# 基于角色的权限管理

- 例子：
- pg93@db-172-16-3-150-> pwd
- /ssd2/pg93/pg\_root
- pg93@db-172-16-3-150-> mkdir tbs\_test
- digoal=# create tablespace tbs\_test location '/ssd2/pg93/pg\_root/tbs\_test';
- CREATE TABLESPACE
- digoal=# create database test with template template0 owner postgres encoding 'UTF8' tablespace tbs\_test;
- CREATE DATABASE
- digoal=# \c digoal digoal
- You are now connected to database "digoal" as user "digoal".
- digoal=> drop tablespace tbs\_test;
- ERROR: must be owner of tablespace tbs\_test
- digoal=> drop database test;
- ERROR: must be owner of database test
- digoal=> \c test digoal
- You are now connected to database "test" as user "digoal".

# 基于角色的权限管理

- test=> create schema digoal;
- ERROR: permission denied for database test
- test=> create table public.test (id int);
- CREATE TABLE
- -- 默认只有connect和public SCHEMA权限. 回收后就无法连接了
- test=> \c test postgres
- You are now connected to database "test" as user "postgres".
- test=# revoke all on database test from public;
- REVOKE
- test=# revoke all on schema public from public;
- REVOKE
- test=# \c test digoal
- FATAL: permission denied for database "test"
- DETAILED: User does not have CONNECT privilege.
- Previous connection kept

# 基于角色的权限管理

- -- 赋权
- test=# grant all on database test to digoal;
- GRANT
- test=# \c test digoal
- You are now connected to database "test" as user "digoal".
- test=> create schema digoal;
- CREATE SCHEMA

# 事件触发器

- Command: CREATE EVENT TRIGGER
- Description: define a new event trigger
- Syntax:
- CREATE EVENT TRIGGER name
- ON event
- [ WHEN filter\_variable IN (filter\_value [ , ... ]) [ AND ... ] ]
- EXECUTE PROCEDURE function\_name()
  
- <http://blog.163.com/digoal@126/blog/static/16387704020132131361949/>
- 语法解释：
  - -- name : 触发器名称
  - -- event : 事件名称, 现在支持的事件为ddl\_command\_start 和 ddl\_command\_end.
  - 支持触发事件触发器的DDL如下(包括select into) :
- <http://www.postgresql.org/docs/devel/static/event-trigger-matrix.html>
- 但是触发事件中不包括对系统共享对象的CREATE, ALTER, DROP操作, 如 :
- databases, roles, and tablespaces
- 同样对事件触发器本身的DDL操作也不会触发事件触发器.

# 事件触发器

- -- filter\_variable 目前只支持TAG
- -- filter\_value 是 <http://www.postgresql.org/docs/devel/static/event-trigger-matrix.html> 这里定义的DDL
- -- function\_name 就是我们创建好的事件触发器函数。
  
- 以plpgsql函数语言为例讲解事件触发器函数的创建方法：
- 事件触发器函数的返回类型为event\_trigger, 同时事件触发器的顶级块带入了两个特殊变量, TG\_EVENT和TG\_TAG.
- TG\_EVENT表示EVENT信息, 如现在支持的为ddl\_command\_start 和 ddl\_command\_end.
- TG\_TAG表示的是DDL信息, 信息在 <http://www.postgresql.org/docs/devel/static/event-trigger-matrix.html> 查询.
  
- 如果同一个事件上建立了多个事件触发器, 执行顺序按触发器名字(而非触发器函数的名字)的字母先后顺序来执行, 这个和普通触发器的触发规则是一样的.
- CREATE EVENT TRIGGER b ON ddl\_command\_start EXECUTE PROCEDURE etgr1();
- CREATE EVENT TRIGGER a ON ddl\_command\_start EXECUTE PROCEDURE etgr2();
- 先触发a, 后触发b.

# 事件触发器

- -- 查询当前数据库中有哪些事件触发器：  
digoal=# select \* from pg\_event\_trigger ;  
evtname | evtevent | evtowner | evtoid | evtenabled | evttags  
-----+-----+-----+-----+-----+-----  
b | ddl\_command\_start | 10 | 16669 | O |  
a | ddl\_command\_start | 10 | 16671 | O |  
(2 rows)

- -- evtoid指事件触发器函数的oid,  
digoal=# select proname from pg\_proc where oid=16669;  
proname  
-----  
etgr1  
(1 row)  
digoal=# select proname from pg\_proc where oid=16671;  
proname  
-----  
etgr2  
(1 row)

# 事件触发器

- 事件触发器和DDL语句本身是在同一个事务中处理的,所以任何事件触发器抛出异常的话,整个事务都会回滚,并且后续的操作也不会执行下去.
  
- 事件触发器应用举例
- 禁止postgres用户在数据库digoal中执行CREATE TABLE和DROP TABLE命令.
- CREATE OR REPLACE FUNCTION abort()
- RETURNS event\_trigger
- LANGUAGE plpgsql
- AS \$\$
- BEGIN
- if current\_user = 'postgres' then
- RAISE EXCEPTION 'event:%, command:%', tg\_event, tg\_tag;
- end if;
- END;
- \$\$;

# 事件触发器

- digoal=# create event trigger a on ddl\_command\_start when TAG IN ('CREATE TABLE', 'DROP TABLE') execute procedure abort();
- CREATE EVENT TRIGGER
- digoal=# select \* from pg\_event\_trigger ;

| evtname | evtevent          | evtowner | evtfoid | evtenabled | evtags                        |
|---------|-------------------|----------|---------|------------|-------------------------------|
| a       | ddl_command_start | 10       | 16683   | O          | {"CREATE TABLE","DROP TABLE"} |
- (1 row)
- 测试postgres用户是否可以使用create table和drop table .
- digoal=# \c digoal postgres
- You are now connected to database "digoal" as user "postgres".
- -- 无法新建表了
- digoal=# create table new(id int);
- ERROR: event:ddl\_command\_start, command:CREATE TABLE
- digoal=# \d new
- Did not find any relation named "new".

# 事件触发器

- digoal=# \dt
- List of relations
- Schema | Name | Type | Owner
- -----+-----+-----+-----
- public | digoal | table | postgres
- public | digoal1 | table | postgres
- public | test | table | postgres
- (3 rows)
- -- 无法删表了
- digoal=# drop table digoal;
- ERROR: event:ddl\_command\_start, command:DROP TABLE
- digoal=# \d digoal
- Table "public.digoal"
- Column | Type | Modifiers
- -----+-----+-----
- id | integer |

# 事件触发器

- -- 测试其他用户是否会有影响
- digoal=# \c digoal digoal
- You are now connected to database "digoal" as user "digoal".
- digoal=> create table tbl(id int);
- CREATE TABLE
- digoal=> drop table tbl;
- DROP TABLE
- -- 未受到影响.

# 数据传输加密

- <http://blog.163.com/digoal@126/blog/static/163877040201342233131835/>
- 要支持ssl连接, 数据库服务端和客户端都需要openssl包.
  
- 端口加密
- <http://blog.163.com/digoal@126/blog/static/163877040201342383123592/>
- ssl ciphers 效率对比
- <http://blog.163.com/digoal@126/blog/static/16387704020134229431304/>

# 审计日志 - SQL

- 记录某用户的所有SQL(假设某用户非业务用户, 是DBA用户)
- 业务用户可以审计DDL, 减少审计日志的输出
  
- 通过配置postgresql.conf就可以实现.
- `log_statement = 'ddl' # none, ddl, mod, all`
- Valid values are none (off), ddl, mod, and all (all statements).
- ddl logs all data definition statements, such as CREATE, ALTER, and DROP statements.
- mod logs all ddl statements, plus data-modifying statements such as INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM. PREPARE, EXECUTE, and EXPLAIN ANALYZE statements are also logged if their contained command is of an appropriate type.
  
- 用户级或结合数据库级别的配置
- `ALTER ROLE name [ IN DATABASE database_name ] SET configuration_parameter { TO | = } { value | DEFAULT }`
- `ALTER ROLE { name | ALL } [ IN DATABASE database_name ] SET configuration_parameter FROM CURRENT`
- `ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET configuration_parameter`
- `ALTER ROLE { name | ALL } [ IN DATABASE database_name ] RESET ALL`

# 审计日志 - SQL

- 设置举例：
- digoal=# alter role digoal set log\_statement = 'mod';
- ALTER ROLE
- digoal=# select \* from pg\_db\_role\_setting ;
- setdatabase | setrole | setconfig
- -----+-----+-----
- 91246 | 0 | {enable\_seqscan=off}
- 0 | 91250 | {log\_statement=mod}
- (2 rows)

# 审计日志 - 基于对象的审计

- 对象触发器: 权限限制, 数据追踪审计
- <http://blog.163.com/digoal@126/blog/static/163877040201252575529358/>
  
- 在数据库应用中, 某些数据表的记录更改可能会需要跟踪, 例如删除, 新增, 更新。
- 跟踪的信息包括: 老的记录值, 新的记录值, 记录变更时间, 哪个用户操作的等等。
- 在数据库中, 跟踪可以通过触发器来做。
- 使用 hstore 类型来存储变更前后的row信息。并且使用hstore提供的each函数, 可以很方便的取出原来存储的值。

# 审计日志 - 基于对象的审计

- -- 创建需要被跟踪的测试表
- CREATE TABLE test (id int primary key, info text, crt\_time timestamp(0));
- -- 创建hstore extension;
- CREATE EXTENSION hstore;
- -- 创建通用的存储跟踪记录的记录表
- CREATE TABLE table\_change\_rec (
  - id serial8 primary key,
  - relid oid,
  - table\_schema text,
  - table\_name text,
  - when\_tg text,
  - level text,
  - op text,
  - old\_rec hstore,
  - new\_rec hstore,
  - crt\_time timestamp without time zone DEFAULT now(),
  - username text,
  - client\_addr inet,
  - client\_port int
- );

# 审计日志 - 基于对象的审计

```
■ -- 创建通用的触发器函数
■ CREATE OR REPLACE FUNCTION dml_trace()
■ RETURNS trigger
■ LANGUAGE plpgsql
■ AS $BODY$
■ DECLARE
■ v_new_rec hstore;
■ v_old_rec hstore;
■ v_username text := session_user;
■ v_client_addr inet := inet_client_addr();
■ v_client_port int := inet_client_port();
■ BEGIN
■ -- 如果要加上用户的判断, 这里加个条件就好了.
■ case TG_OP
■ when 'DELETE' then
■ v_old_rec := hstore(OLD.*);
■ insert into table_change_rec (relid, table_schema, table_name, when_tg, level, op, old_rec, username, client_addr, client_port)
■ values (tg_relid, tg_table_schema, tg_table_name, tg_when, tg_level, tg_op, v_old_rec, v_username, v_client_addr, v_client_port);
■ when 'INSERT' then
■ v_new_rec := hstore(NEW.*);
```

# 审计日志 - 基于对象的审计

```
■ insert into table_change_rec (relid, table_schema, table_name, when_tg, level, op, new_rec, username, client_addr, client_port)
■ values (tg_relid, tg_table_schema, tg_table_name, tg_when, tg_level, tg_op, v_new_rec, v_username, v_client_addr, v_client_port);
■ when 'UPDATE' then
■ v_old_rec := hstore(OLD.*);
■ v_new_rec := hstore(NEW.*);
■ insert into table_change_rec (relid, table_schema, table_name, when_tg, level, op, old_rec, new_rec, username, client_addr, client_port)
■ values (tg_relid, tg_table_schema, tg_table_name, tg_when, tg_level, tg_op, v_old_rec, v_new_rec, v_username, v_client_addr, v_client_port);
■ else
■ return null;
■ end case;
■ RETURN null;
■ END;
■ $BODY$ strict;
```

# 审计日志 - 基于对象的审计

- -- 在测试表上创建触发器.
- CREATE TRIGGER tg AFTER DELETE or INSERT or UPDATE ON test FOR EACH ROW EXECUTE PROCEDURE dml\_trace();
- -- 测试插入, 删除, 更新操作是否被跟踪.
  
- digoal=# insert into test values (1, 'digoal', now());
- INSERT 0 1
- digoal=# update test set info='DIGOAL' where id=1;
- UPDATE 1
- digoal=# delete from test where id=1;
- DELETE 1

# 审计日志 - 基于对象的审计

```
■ digoal=# select * from table_change_rec;
■ id | relid | table_schema | table_name | when_tg | level | op | old_rec
■ | new_rec | crt_time | username | client_addr | client_port
■
■ +-----+-----+-----+-----+-----+
■ +-----+-----+-----+-----+-----+
■ -
■ 1 | 93063 | postgres | test | AFTER | ROW | INSERT |
■ | "id"=>"1", "info"=>"digoal", "crt_time"=>"2013-12-18 11:09:17" | 2013-12-18 11:09:16.501033 | postgres |
■ 2 | 93063 | postgres | test | AFTER | ROW | UPDATE | "id"=>"1", "info"=>"digoal", "crt_time"=>"2013-12-18 11:09:17"
■ | "id"=>"1", "info"=>"DIGOAL", "crt_time"=>"2013-12-18 11:09:17" | 2013-12-18 11:09:24.514068 | postgres |
■ 3 | 93063 | postgres | test | AFTER | ROW | DELETE | "id"=>"1", "info"=>"DIGOAL", "crt_time"=>"2013-12-18 11:09:17"
■ | | 2013-12-18 11:09:30.626064 | postgres |
■ (3 rows)
```

# 审计日志 - 基于对象的审计

■ digoal=# select id,(each(old\_rec)).\* from table\_change\_rec;

■ id | key | value

■ -----+-----+

■ 2 | id | 1

■ 2 | info | digoal

■ 2 | crt\_time | 2013-12-18 11:09:17

■ 3 | id | 1

■ 3 | info | DIGOAL

■ 3 | crt\_time | 2013-12-18 11:09:17

■ (6 rows)

■ digoal=# select id,(each(new\_rec)).\* from table\_change\_rec;

■ id | key | value

■ -----+-----+

■ 1 | id | 1

■ 1 | info | digoal

■ 1 | crt\_time | 2013-12-18 11:09:17

■ 2 | id | 1

■ 2 | info | DIGOAL

■ 2 | crt\_time | 2013-12-18 11:09:17

■ (6 rows)

# 对象安全控制 - 约束,视图,物化视图

## ■ 约束

- 一般用于控制数据的安全
- 如 check (balance >=0)

## ■ 视图和物化视图

- 一般被用于控制列或者值的被查询安全
- 查询列
- 条件筛选行
- 加密列

## ■ 防范视图攻击

- <http://blog.163.com/digoal@126/blog/static/163877040201361031431669/>

# 安全标签

- 结合SELinux使用
- <http://www.postgresql.org/docs/9.3/static/sepgsql.html>
- <http://www.postgresql.org/docs/9.3/static/sql-security-label.html>

# 防范SQL注入

- <http://blog.163.com/digoal@126/blog/static/163877040201342184212205/>
- <http://blog.163.com/digoal@126/blog/static/1638770402012910234402/>
  
- 程序端控制
  - 1. 不要使用simple协议, 因为simple协议允许一次提交多SQL.
  - 例如 :  
statement = "SELECT \* FROM users WHERE name = '" + userName + "';"  
这个SQL, 可以在userName这里注入攻击.  
a';DROP TABLE users; SELECT \* FROM userinfo WHERE 't' = 't  
那么整条sql就变成 :  
SELECT \* FROM users WHERE name = 'a'; DROP TABLE users; SELECT \* FROM userinfo WHERE 't' = 't';
  - 使用绑定变量避免以上问题
  
- 2. 程序端控制, 例如检测变量类型, 以及过滤. 以下攻击只要检查变量类型, 拒绝请求就不会有问题.  
statement := "SELECT \* FROM userinfo WHERE id = " + a\_variable + ";"  
如果a\_variable 不做任何判断, 传入 1;DROP TABLE users  
那么SQL变成 :  
SELECT \* FROM userinfo WHERE id=1;DROP TABLE users;

# 给数据库打补丁

- 方法1, 下载对应版本的小版本更新, 同时根据版本release note升级软件.
  - <http://www.postgresql.org/ftp/source/>
  - 如果还安装了第三方插件, 第三方插件重新安装一下.
  - 升级完后使用新版本的软件重启数据库.
  
- 方法2, 对于未释放小版本的情况, 想更新GIT提交的补丁的话.
  - 去git找到对应的大版本
  - 例如
  - [http://git.postgresql.org/gitweb/?p=postgresql.git;a=shortlog;h=refs/heads/REL9\\_3\\_STABLE](http://git.postgresql.org/gitweb/?p=postgresql.git;a=shortlog;h=refs/heads/REL9_3_STABLE)
  - 下载对应的snapshot重新安装
  - 如果还安装了第三方插件, 第三方插件重新安装一下.
  - 升级完后使用新版本的软件重启数据库.
  
- 方法3, 下载commit fest中已经程提交状态的补丁, 使用diff匹配到当前已经安装的源码中.
  - 重新安装.
  - 注意先备份一下原来安装的源码目录.
  - <http://blog.163.com/digoal@126/blog/static/163877040201252884053930/>

# 练习

- PostgreSQL安全简介, 认证, 基于角色的权限管理, 事件触发器
- 审计日志
- 对象触发器: 权限限制, 数据追踪审计
- 约束
- 视图
- 物化视图
- 安全标签, se-postgres
- 如何防范SQL注入

# 高可用,负载均衡

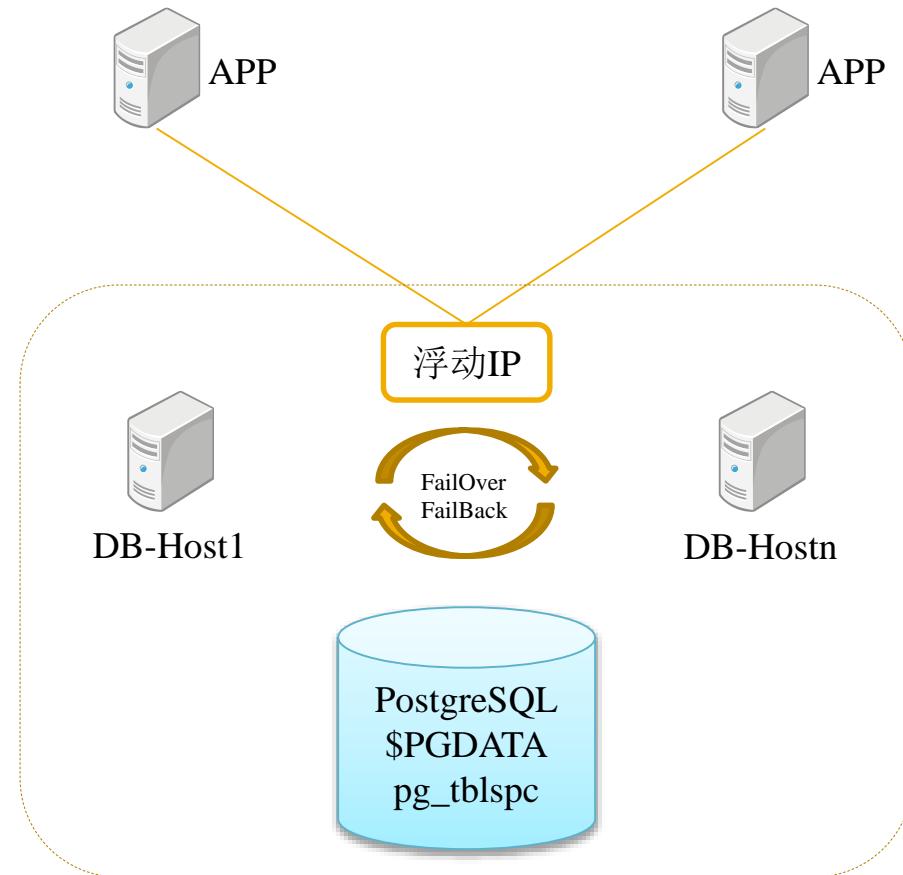
- 了解高可用架构, PostgreSQL高可用的实现方法, 挑选几种演示讲解
- 了解负载均衡的应用场景, PG的实现方法, 挑选几种演示讲解

# 高可用解决方案介绍

- 共享存储, 依赖存储本身的高可用
- 块设备复制(开源的drbd, 或者商业存储复制的解决方案)
- 数据库流复制

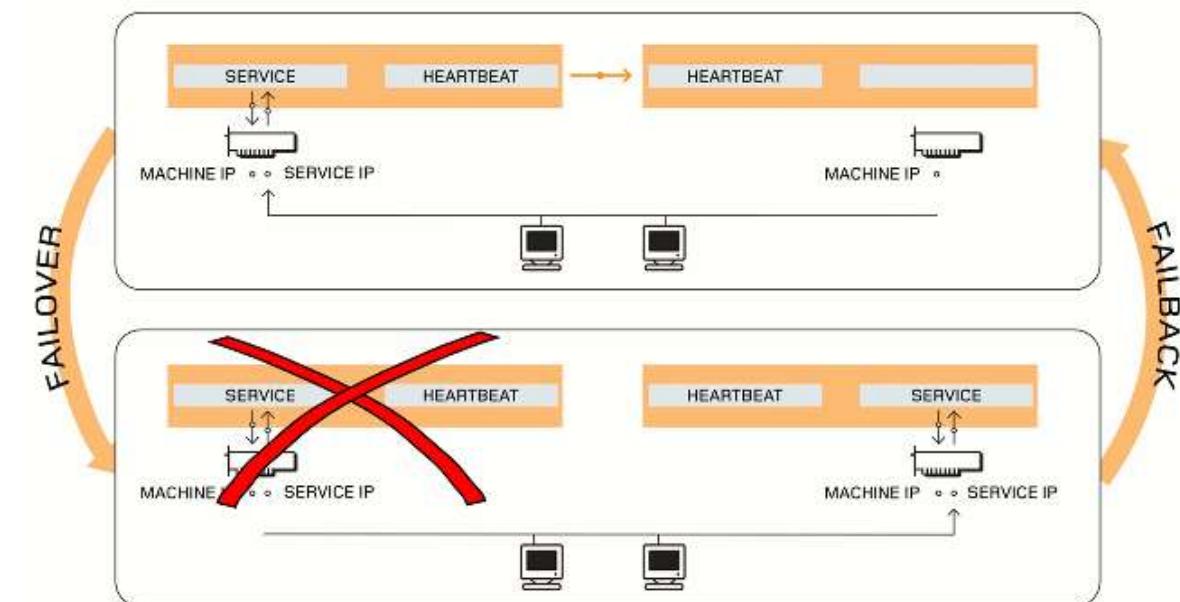
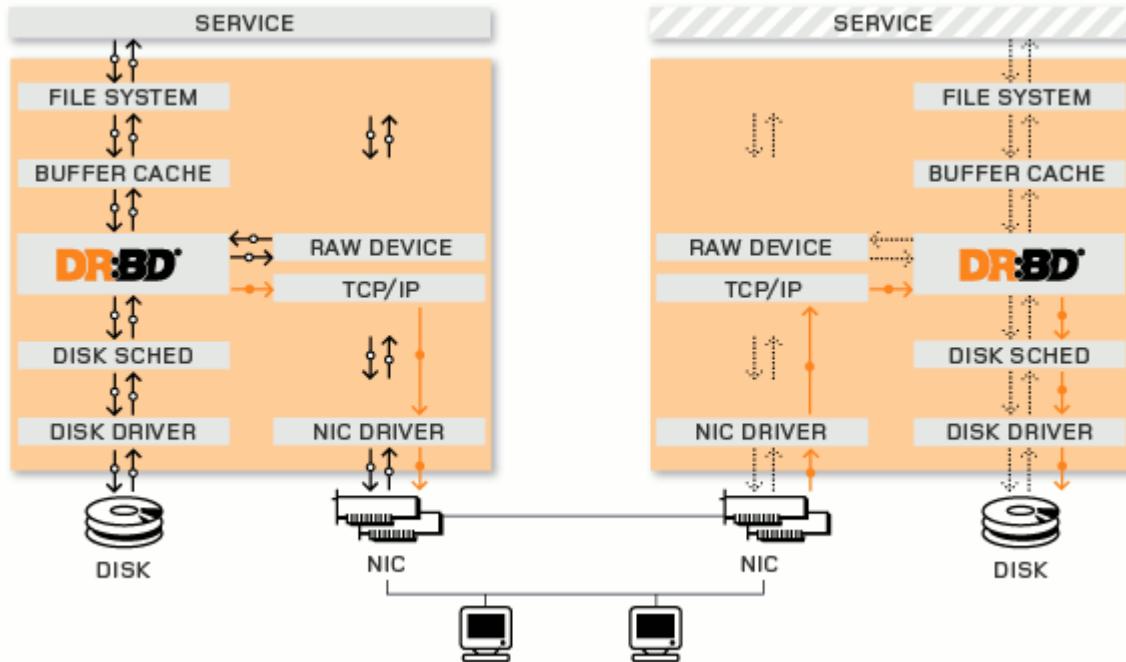
# 共享存储

- 共享存储(数据文件以及表空间的文件必须共享存储).
- 同一时间点只能有1个主机启动数据库, 其他主机可以挂载文件系统(仅限于集群文件系统).
- 浮动IP跟随起数据库的主机一起.
- 通过检测心跳判断异常, fence掉老的活动主机, 在候选机器上启动数据库.
- 主要的弊端:
- 受制于存储可用性



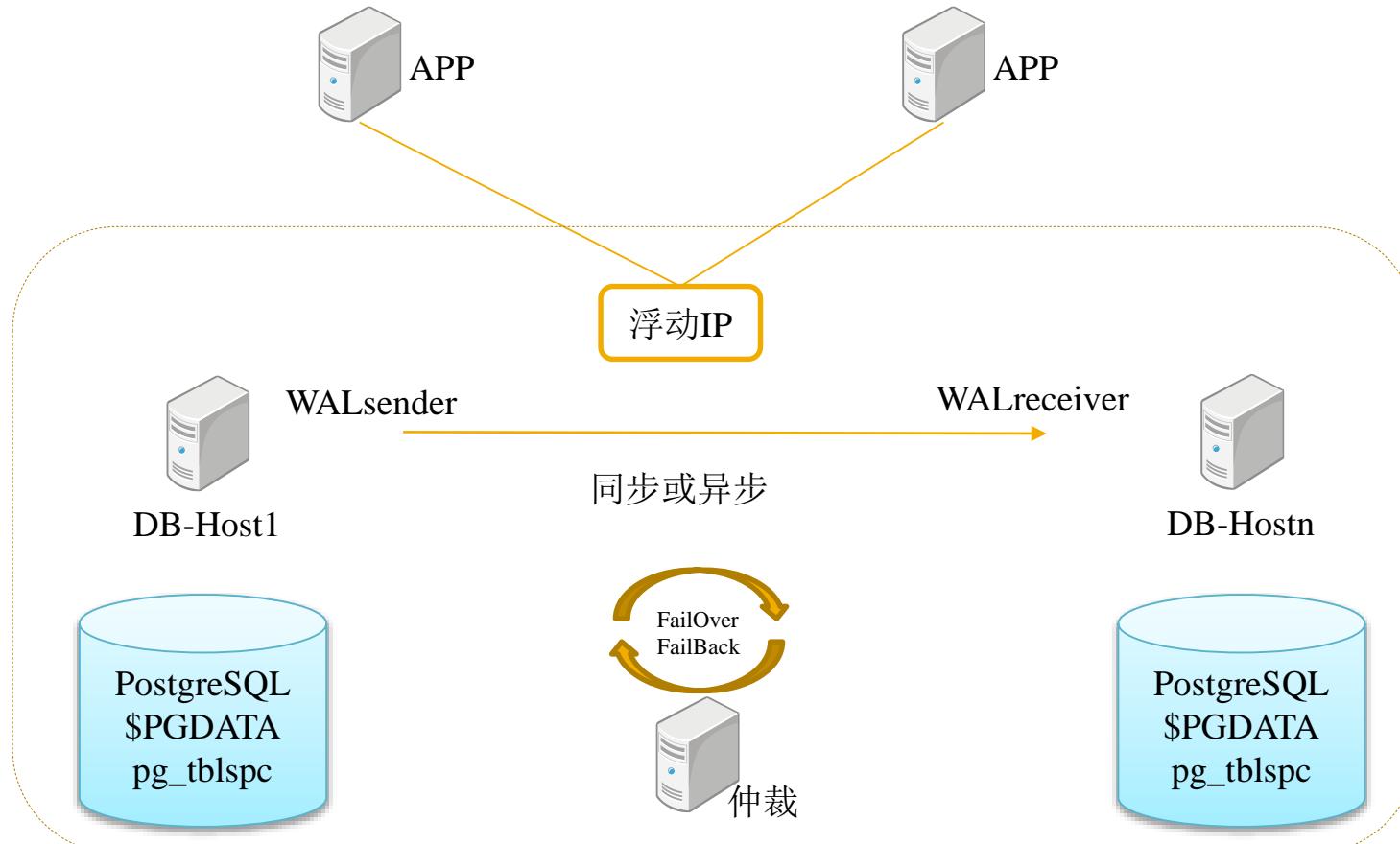
# 块设备复制

- mirroring a whole block device via an assigned network. DRBD can be understood as network based raid-1.



# 数据库流复制+心跳+自动切换

- 利用数据库流复制建立HA, 为了防止脑分裂, 需要仲裁和FENCE设备. FENCE设备即IPMI管理的设备
- 注意
  - 某些不写WAL的数据切换后数据将丢失(例如UNLOGGED TABLE(这个就算不切换也会丢失), HASH index(重建即可))



# 数据库流复制+心跳+自动切换

- 心跳+切换的流程
  - 判断主库是否健康 -> 不健康则退出
  - 判断本机是否为STANDBY角色 -> 不是STANDBY角色则退出
  - 连接到主库执行一次更新, 5秒后到本地STANDBY节点检查这次更新是否已经生效, 用于判断standby是否正常. -> 不正常则退出
  - 循环
    - 执行心跳检测函数
      - (更新检测时间点, 检查各个表空间读写是否正常)
      - 心跳检测函数超时或者异常则判断仲裁节点是否正常, 仲裁节点不正常则跳出循环
      - 仲裁节点正常的情况下从仲裁节点连到主库执行心跳函数, 正常则跳出循环
      - 如果返回结果也不正常的话, 判断STANDBY延迟是否在允许范围内.
      - 开始计数
      - 计数超过10则发生切换(fence主库, 激活备库, 切换VIP)
  - 详见
    - [https://raw.githubusercontent.com/digoal/sky\\_postgresql\\_cluster/master/sky\\_pg\\_clusterd.sh](https://raw.githubusercontent.com/digoal/sky_postgresql_cluster/master/sky_pg_clusterd.sh)

# 数据库流复制+心跳+自动切换

- 心跳检查函数
- CREATE OR REPLACE FUNCTION sky\_pg\_cluster.cluster\_keepalive\_test()
- RETURNS void
- LANGUAGE plpgsql
- STRICT
- AS \$function\$
- declare
- v\_spcname text;
- v\_spcoid oid;
- v\_nsppname name := 'sky\_pg\_cluster';
- begin
- if ( pg\_is\_in\_recovery() ) then
- raise notice 'this is standby node.';
- return;
- end if;
- update cluster\_status set last\_alive=now();

# 数据库流复制+心跳+自动切换

```
■ FOR v_spcname,v_spcoid IN
■ select spcname,oid from pg_tablespace where
■ oid <> (select dattablespace from pg_database where datname=current_database())
■ and spcname <> 'pg_global'
■ LOOP
■ perform 1 from pg_class where
■ reltablespace=v_spcoid
■ and relname='t_||v_spcname
■ and relkind='r'
■ and relnamespace=(select oid from pg_namespace where nspname=v_nspname)
■ limit 1;
■ if not found then
■ execute 'create table'||v_nspname||'.t_||v_spcname||(crt_time timestamp) tablespace'||v_spcname;
■ execute 'insert into'||v_nspname||'.t_||v_spcname||' values ("'||now()||")';
■ else
■ execute 'update'||v_nspname||'.t_||v_spcname||' set crt_time='||""||now()||"';
■ end if;
■ perform pg_stat_file(pg_relation_filepath(v_nspname||'.t_||v_spcname));
```

# 数据库流复制+心跳+自动切换

```
■ END LOOP;
■ select spcname into v_spcname from pg_tablespace where
■ oid = (select dat_tablespace from pg_database where datname=current_database());
■ perform 1 from pg_class where
■ reltablespace=0
■ and relname='t_||v_spcname
■ and relkind='r'
■ and relnamespace=(select oid from pg_namespace where nspname=v_nspname)
■ limit 1;
■ if not found then
■ execute 'create table'||v_nspname||'.t_||v_spcname||(crt_time timestamp) tablespace'||v_spcname;
■ execute 'insert into'||v_nspname||'.t_||v_spcname|| values ("||now()||");
■ else
■ execute 'update'||v_nspname||'.t_||v_spcname|| set crt_time='||""||now()||"';
■ end if;
■ perform pg_stat_file(pg_relation_filepath(v_nspname||'.t_||v_spcname));
■ end;
■ $function$
```

# 数据库流复制集群搭建演示

- fence设备配置,
- DELL 172.16.3.39 fence: 172.16.3.191 cluster/Cluster321
- HP 172.16.3.33 fence: 172.16.3.193 cluster/Cluster321
- 虚拟IP : 172.16.3.38
- 仲裁 : 172.16.3.150:11921
  
- 详细步骤参考
- [https://github.com/digoal/sky\\_postgresql\\_cluster/blob/master/INSTALL.txt](https://github.com/digoal/sky_postgresql_cluster/blob/master/INSTALL.txt)

常规 • IPMI 用户权限 • iDRAC 用户权限

**常规**

|       |                                     |
|-------|-------------------------------------|
| 用户 ID | 3                                   |
| 启用用户  | <input checked="" type="checkbox"/> |
| 用户名   | cluster                             |
| 更改密码  | <input type="checkbox"/>            |
| 新密码   | *****                               |
| 确认新密码 | *****                               |

[返回页首]

## IPMI 用户权限

|                |                                     |
|----------------|-------------------------------------|
| 授予的最大 LAN 用户权限 | 操作员 ▼                               |
| 授予的最大串行端口用户权限  | 操作员 ▼                               |
| 启用 LAN 上串行     | <input checked="" type="checkbox"/> |

# PostgreSQL 负载均衡

## ■ pgpool-II

- pgpool-II作为连接池和中间件应用, 后端接流复制环境可以实现对应用透明的读写分离的负载均衡

## ■ pl/proxy

- 首先pl/proxy是个比较小巧的插件, 利用函数封装代替直接执行SQL语句.
- pl/proxy后端接流复制环境同样可以实现对应用透明的读写分离的负载均衡
- 写函数(RUN ON o;)
- 读函数(RUN ON ANY;) 或者 (RUN ON int4; int4为一个返回1-N随机值的函数).
- 演示

# 练习

- 了解高可用架构, PostgreSQL高可用的实现方法, 挑选几种演示讲解
- 了解负载均衡的应用场景, PG的实现方法, 挑选几种演示讲解

# 数据库规划

- 根据业务形态, 合理规划PostgreSQL数据库硬件和操作系统.
- 如何建模以及压力测试.

# 数据库规划

- 基本原则
  - 数据完全没有交集的业务建议放在不同的数据库中.
  - 有数据交集的业务建议放在同数据库, 使用不同的schema区分.
- 存储划分规则
  - 建议使用独立的块设备的目录
  - \$PGDATA, pg\_xlog, stats\_temp\_directory, pg\_log, 表空间, 归档, 备份
- 定时任务的规则
  - 数据库服务端的定时任如pgagent或crontab, 需要有执行步骤的输出, 有监控.
- 监控的规划
  - 实时的监控用以迅速发现数据库运行中的问题
  - 历史数据的快照存储可以用于输出趋势报告或选取时间段的报告
- BenchMark
  - 对数据量的预估, 初步决定存储容量需求, 以及建模测试数据量需求.
  - 对可能成为数据库瓶颈的SQL的预估(从请求量和运算量两个角度来评估)
  - SQL审核(数据库侧的优化, 对于需要修改SQL或建议优化业务逻辑的情况反馈给开发人员)
  - 建模和压力测试, 输出benchmark.

# 数据库规划

## ■ 规划输出(数据库生命周期管理)：

- 硬件配置(例如CPU, 内存, 存储, 网卡等)
- 操作系统配置(如内核参数, 防火墙, 服务开关等)
- 存储划分, 文件系统初始化
- PostgreSQL编译配置安装, 第三方插件安装.
- 数据库postgresql.conf, pg\_hba.conf的配置
- 表空间以及其他数据库目录对应的OS目录
- 定时任务
- 备份和归档
- 集群的配置
- 监控部署
- 数据库巡检文档
- SQL提交(包括建用户, 建库, 表空间, 表, 索引的DDL; 表, 索引表空间指定; 表级别的存储参数; 初始化数据等等)
- BenchMark

# 数据库规划

## ■ 分布式环境数据库系统的规划的基本原则

- 扩展性和收缩性
- 分布列的选择
- 减少跨库的交互, 例如全局小表多库同步
- 一致性备份的需求

# 建模和压力测试举例

- 建模和优化案例
  - <http://blog.163.com/digoal@126/blog/static/163877040201221382150858/>
  - <http://blog.163.com/digoal@126/blog/static/163877040201221333411196/>
- pgbench工具的使用
  - <http://www.postgresql.org/docs/9.3/static/pgbench.html>

# 练习

- 根据业务形态, 合理规划PostgreSQL数据库硬件和操作系统
- 如何建模以及压力测试.