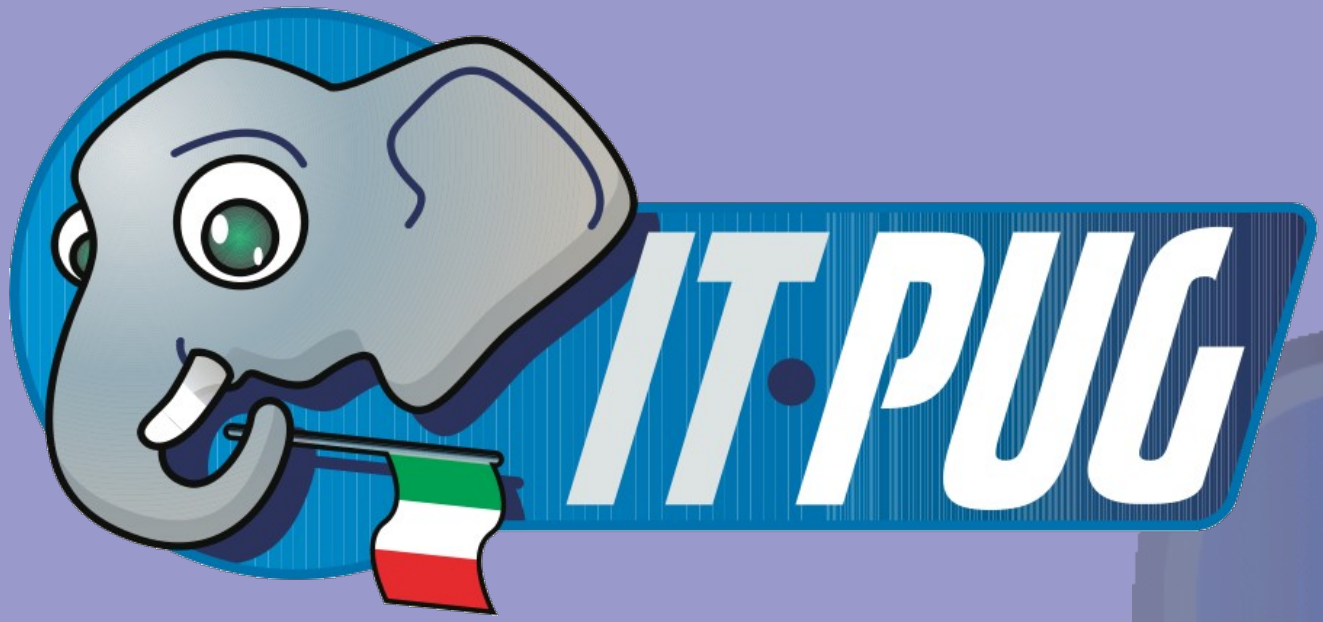


PostgreSQL 9

il piu' avanzato database OpenSource al mondo

Italian PgDay 2012 – 23 Novembre 2012 - Prato

Ing. Luca Ferrari, PhD

Adjunt Professor – Nipissing University

fluca1978@gmail.com

Synopsis (revisione 21/11/12)

Questa presentazione vuole fornire una introduzione al progetto PostgreSQL, un database *OpenSource* di fascia *Enterprise* mantenuto da un gruppo di esperti mondiali. Saranno trattati diversi temi, in particolare:

- Server Side programming (cenni)
- Point in Time Recovery
- Replica

Server Side Programming

PostgreSQL supporta egregiamente il Server Side Programming, consentendo l'inclusione di logica applicativa direttamente nel backend:

- Triggers → gestori di eventi
- Stored Procedures → funzioni user-defined
- Rules → riscrittura di query al volo
- Listeners → gestori di eventi (non standard)

Una caratteristica importante è che le funzioni (e quindi anche i trigger e le rules) possono essere scritti in *qualunque* linguaggio.

Linguaggi trusted vs Linguaggi Untrusted

Triggers

When	DML query	Applies to	NEW tuple means	OLD tuple means	Return value
BEFORE	INSERT	EACH ROW	The tuple that will be committed.	N/A	NULL to abort changes of the current tuple NEW to commit changes on the tuple
		STATEMENT	N/A	N/A	NULL
BEFORE	UPDATE	EACH ROW	The tuple that will be committed.	The tuple at the time the transaction began.	NULL to abort changes of the current tuple NEW to commit changes on the tuple OLD to rollback changes
		STATEMENT	N/A	N/A	NULL
AFTER	INSERT	EACH ROW	The tuple that will be committed.	N/A	Either OLD, NEW or NULL but it will not change the tuple values (i.e., the return value is ignored).
		STATEMENT	N/A	N/A	NULL
AFTER	UPDATE	EACH ROW	The tuple that will be committed.	The tuple at the time the transaction began.	Either OLD, NEW or NULL but it will not change the tuple values (i.e., the return value is ignored).
		STATEMENT	N/A	N/A	NULL
BEFORE	DELETE	EACH ROW	N/A	The tuple that will be deleted.	OLD to commit changes (i.e., delete the tuple). NULL to abport the deletion
		STATEMENT	N/A	N/A	NULL
AFTER	DELETE	EACH ROW	N/A	The tuple that will be deleted.	Either OLD, NEW or NULL but it will not change the tuple values (i.e., the return value is ignored).
		STATEMENT	N/A	N/A	NULL
BEFORE	TRUNCATE	STATEMENT	N/A	N/A	NULL
AFTER	TRUNCATE	STATEMENT	N/A	N/A	NULL

Esempio di trigger

```
CREATE OR REPLACE FUNCTION compute_download_path()
RETURNS trigger AS
$BODY$
DECLARE

BEGIN
  -- if the magazine has been issued
  -- compose the download path
  IF NEW.issuedon IS NOT NULL THEN
     NEW.download_path := 'http://bsdmag.org/download-demo/BSD_' || NEW.id || '.pdf';
     RAISE LOG 'Computed download for issue $ path is %', NEW.title, NEW.download_path;
  ELSE
     RAISE LOG 'Removing the download path for issue %', NEW.title;
     NEW.download_path := NULL;
  END IF;

  RETURN NEW;

END;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

1 – Definire una procedura che implementa la logica del trigger.

2 – Associare la procedura ad un evento su una tabella

```
CREATE TRIGGER tr_download_path
BEFORE INSERT OR UPDATE ON magazine
FOR EACH ROW
EXECUTE PROCEDURE compute_download_path();
```

Esempio di Trigger parametrico

```
CREATE OR REPLACE FUNCTION compute_download_path()
RETURNS trigger AS
$BODY$
        default_path text;
BEGIN
  -- check if the trigger has a path as argument,
  -- otherwise use a default path
  IF TG_NARGS > 0 THEN
     -- first argument is the path
     default_path := TG_ARGV[ 0 ];
     RAISE LOG 'Using a trigger-level path %', default_path;
  ELSE
    -- a default hard coded path
    default_path :=  'http://bsdmag.org/download-demo/';
  END IF;
  -- print an LOG message
  RAISE LOG 'Trigger % executing for % event', TG_NAME, TG_OP;
  -- if executing for a single column then compute the path
  IF TG_OP = 'UPDATE'  THEN
     IF NEW.issuedon IS NOT NULL THEN
        NEW.download_path := default_path || 'BSD_' || NEW.id || '.pdf';
        RAISE LOG 'Computed download fo
     ELSE
        RAISE LOG 'Removing the downloa
        NEW.download_path := NULL;
     END IF;
     RETURN NEW;
  END IF;
```

Il prototipo della funzione trigger non accetta mai parametri, ma il trigger può accedere ad eventuali parametri mediante TG_ARGV

Un trigger può anche fare introspezione sapendo per quale evento è stato chiamato.

I parametri sono specificati quando il trigger viene definito.

```
CREATE TRIGGER tr_i_download_path
AFTER INSERT
ON magazine
FOR EACH STATEMENT
EXECUTE PROCEDURE compute_download_path( 'http://bsdmag.org/download-demo/' );
```

Rules

```
CREATE OR REPLACE  RULE r_delete_magazine
AS ON DELETE
TO magazine
DO INSTEAD
UPDATE magazine SET available = false, issuedon = NULL
WHERE  pk = OLD.pk;
```

Ogni volta che viene eseguito un DELETE sulla tabella MAGAZINE eseguire invece un UPDATE

Stored Procedures: plpgsql

```
CREATE OR REPLACE FUNCTION max_age()
  RETURNS integer AS
$BODY$
DECLARE
        /* dichiarazione variabili */
        max_age_found   integer;
BEGIN
        /* codice funzione */

        SELECT max(age)
        INTO   max_age_found
        FROM   person;

        RETURN max_age_found;



END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Esempio di Trigger in Java

```
hrpmdb=# create table java_table( id serial, nome varchar, description text);
NOTICE:  CREATE TABLE will create implicit sequence "java_table_id_seq" for serial column "java_table.id"
CREATE TABLE
```

```
hrpmdb=# CREATE FUNCTION pljtrigger() RETURNS trigger
hrpmdb-# AS 'itpug.pljava.Trigger.triggerJavaMethod'
hrpmdb-# LANGUAGE 'java';
```

```
hrpmdb=# CREATE TRIGGER pljtrigger_impl BEFORE INSERT ON java_table
hrpmdb-# FOR EACH ROW EXECUTE PROCEDURE pljtrigger();
```

Esempio di Trigger in Java (2)

```
public static void triggerJavaMethod( TriggerData triggerData ){
    StringBuffer info = new StringBuffer(100);

    try {

        // nome del trigger
        info.append( triggerData.getName() );

        // istante di invocazione
        if( triggerData.isFiredAfter() )
            info.append("after");
        else
            info.append("before");

        info.append("-");

        // per cosa e' stato invocato?
        if( triggerData.isFiredByDelete() )
            info.append("by_delete");
        else if( triggerData.isFiredByInsert() )
            info.append("by_insert");
        else if( triggerData.isFiredByUpdate() )
            info.append("by_update");

        // update della colonna nel nuovo result set
        ResultSet newRS = triggerData.getNew();
        newRS.updateString("description", info.toString());


    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

```
hrpmdb=# insert into java_table(nome) values('Luca');
INSERT 0 1
hrpmdb=# select * from java_table;
 id | nome |           description
----+------+-------------------------------
  1 | Luca | pljtrigger_implbefore-by_insert
(1 row)
```

Esempio di Stored Procedure in Java

```java
package itpug.pljava;

public class StringUtils {

    public static int length(String s){
        if( s == null )
            return 0;
        else
            return s.length();
    }
}

```

```
hrpmdb=# CREATE FUNCTION example2( CHARACTER VARYING )
  RETURNS INTEGER
  AS 'itpug.pljava.StringUtils.length'
  IMMUTABLE LANGUAGE java;
```

```
hrpmdb=# select example2('Italian PostgreSQL Users Group');
 example2
----------
       30
(1 row)
```

```
hrpmdb=# select pg_proc.prosrc from pg_proc where pg_proc.proname = 'example2';
             prosrc
-------------------------------
 itpug.pljava.StringUtils.length
(1 row)
```

Stored Procedures: Perl (1)

```
CREATE OR REPLACE FUNCTION notify_readers( integer ) RETURNS integer
    LANGUAGE plperlu
    AS $_$
    my ($issuepk)    = @_;
    my $sent_emails = 0;
    my $query        = "SELECT title, download_path FROM magazine WHERE pk = $issuepk";
    my $boundary     = "===BSDMAG===";
    my $sent         = 0;
    my ($result_set, $email_text, $name, $current_row, $email, %mail);
    my ($download_path, $title);


    elog( LOG, "Extracting the issue data via the query $query\n");
    $result_set    = spi_exec_query( $query );
    $current_row   = $result_set->{rows}[ 0 ];
    $title         = $current_row->{"title"};
    $download_path = $current_row->{"download_path"};
    elog( LOG, "Magazine Issue: $title at $download_path \n" );

    $query       = "SELECT email, name FROM readers";
    elog( LOG, "Extracting all readers via the query $query\n");
    $result_set = spi_exec_query( $query );
    $num_rows    = $result_set->{processed};
    elog( LOG, "Found $num_rows readers\n" );
```

```
bsdmagdb=# CREATE LANGUAGE plperl;
```

Stored Procedures: Perl (2)

```
    # iterate on each reader
    for( $i = 0; $i < $num_rows; $i++ ){
       $current_row = $result_set->{rows}[ $i ];
       $name        = $current_row->{"name"};
       $email       = $current_row->{"email"};
       # build the e-mail
       %mail = ( From     => 'postgres@bsdmag.org',
                 To       => $email,
                 Subject  => 'New BSD Magazine Issue!'
               );
       $mail{smtp} = 'yoursmtp.bsdmag.org';
       $mail{body} = << "END_OF_BODY";
$boundary--
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Dear $name,
there is a new issue of BSD Magazine available for
download at the URL $download_path
so please check it out!
$boundary----
END_OF_BODY

       elog( LOG, "Notifying reader $name at email $email\n" );
       sendmail( %mail ) or warn( $Mail::Sendmail::error) ;
       $sent++;
    }
    return $sent;

    $_$;
```

Stored Procedures: Perl (3)

```
bsdmagdb=# select notify_readers( 2 );
LOG:  Extracting the issue data via the query SELECT
                    title, download_path FROM magazine
                    WHERE pk = 2
CONTEXT:  PL/Perl function "notify_readers"
LOG:  Magazine Issue: Rolling Your Own Kernel at http:
                    //bsdmag.org/download-demo/BSD_2011-
                    12.pdf
CONTEXT:  PL/Perl function "notify_readers"
LOG:  Extracting all readers via the query SELECT email,
                    name FROM readers
CONTEXT:  PL/Perl function "notify_readers"
LOG:  Found 2 readers

LOG:  Notifying reader Luca Ferrari at email
                    lf@fakemail.com
CONTEXT:  PL/Perl function "notify_readers"
LOG:  Notifying reader Ritchie Root at email
                    rr@fakemail.com
CONTEXT:  PL/Perl function "notify_readers"
 notify_readers
----------------
              2
```

Event Listeners: logger di cancellazioni (1)

```
#!/usr/bin/env perl
use DBI;
#use PgPP;
use Data::Dumper;
use IO::Select;

my $database = "dbi:PgPP:dbname=bsdmagdb";
my $username = 'bsdmag';
my $connection =  DBI->connect( $database, $username, '' ) || undef();

my $channel = "delete_channel";
print "\nListening on channel $channel\n";
$connection->do( "LISTEN $channel" );
open( $LOG_FILE, ">>", "/tmp/deletion.log" ) || croack("Cannot create log
file\n$!\n");
```

Event Listeners: logger di cancellazioni (2)

```
while( 1 ){
    sleep 10;
    # get the event back
    while( my $event = $connection->func("pg_notifies") ){
    if( defined($event) ) {
        my ($eventName, $pid, $payload) = @$event;
        if( defined( $payload ) && $payload =~ /(.*)\#(.*)\#/ ){
            $sql  = "SELECT username, client_addr, client_hostname";
            $sql .= " FROM pg_stat_activity ";
            $sql .= " WHERE procpid = $pid; ";
            $resultset_arrayref = $connection->selectall_arrayref( $sql );
            my $username = $resultset_arrayref->[0][0];
            my $ip       = $resultset_arrayref->[0][1];
            my $hostname = $resultset_arrayref->[0][2];

            print $LOG_FILE "#### DELETION EVENT ####\n";
            print $LOG_FILE "Backend process $pid deleted the magazine issue titled
$2\n";
            print $LOG_FILE "\tUsername $username from client $ip ($hostname)\n";

        }
    }
    }
}
close( $LOG_FILE );
```

Event Listeners: logger di cancellazioni (3)

```
CREATE OR REPLACE RULE r_delete_magazine
AS ON DELETE TO magazine
DO ALSO
NOTIFY delete_channel, 'Deletion of a tuple';
```

pg_notify consente di passare informazioni piu' dettagliate nell'evento

```
CREATE OR REPLACE RULE r_delete_magazine
AS ON DELETE TO magazine
DO ALSO
SELECT pg_notify( 'delete_channel', 'Deletion of the tuple
                  titled: #' || OLD.title || '#');
```

```
#### DELETION EVENT ####
Backend process 3022 deleted the magazine issue titled
                FreeBSD: Get Up To Date
        Username bsdmag from client 192.168.200.1 (flucabsd)
```

Script Perl in esecuzione!

PITR

Point In Time Recovery: il restore al momento che vuoi!

PITR

Il Point In Time Recovery è una tecnica che consente il ripristino di un database ad un dato istante temporale.

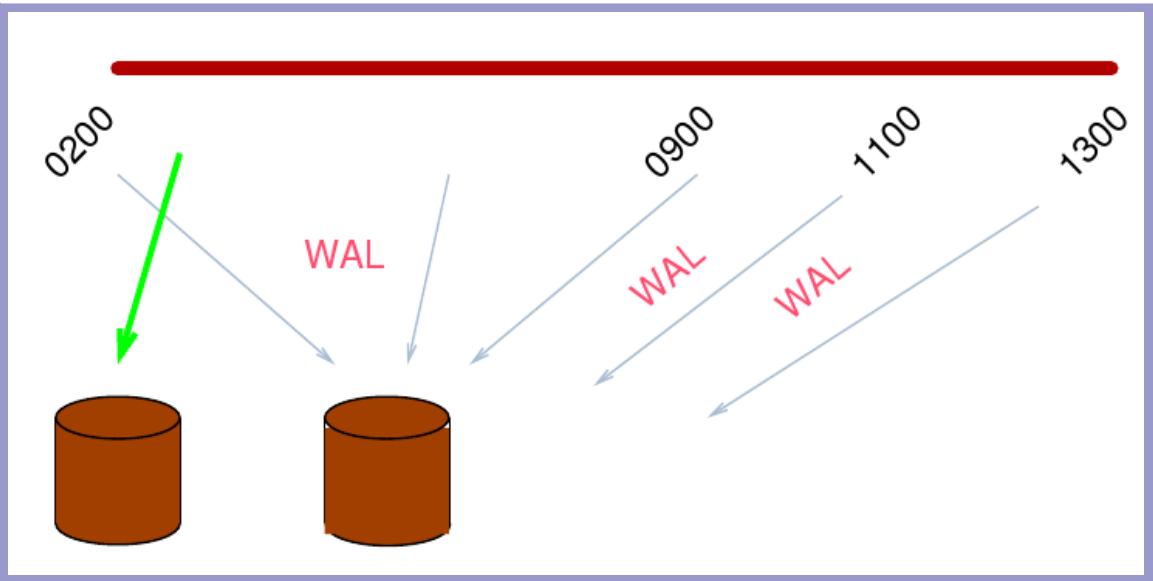
I normali backup (es. `pg_dump`) vengono effettuati ad un dato istante e consentono di ripristinare l'istanza a quel particolare istante. Il PITR invece permette di effettuare un backup continuo e consente all'amministratore di scegliere esattamente a quale istante temporale effettuare il backup.

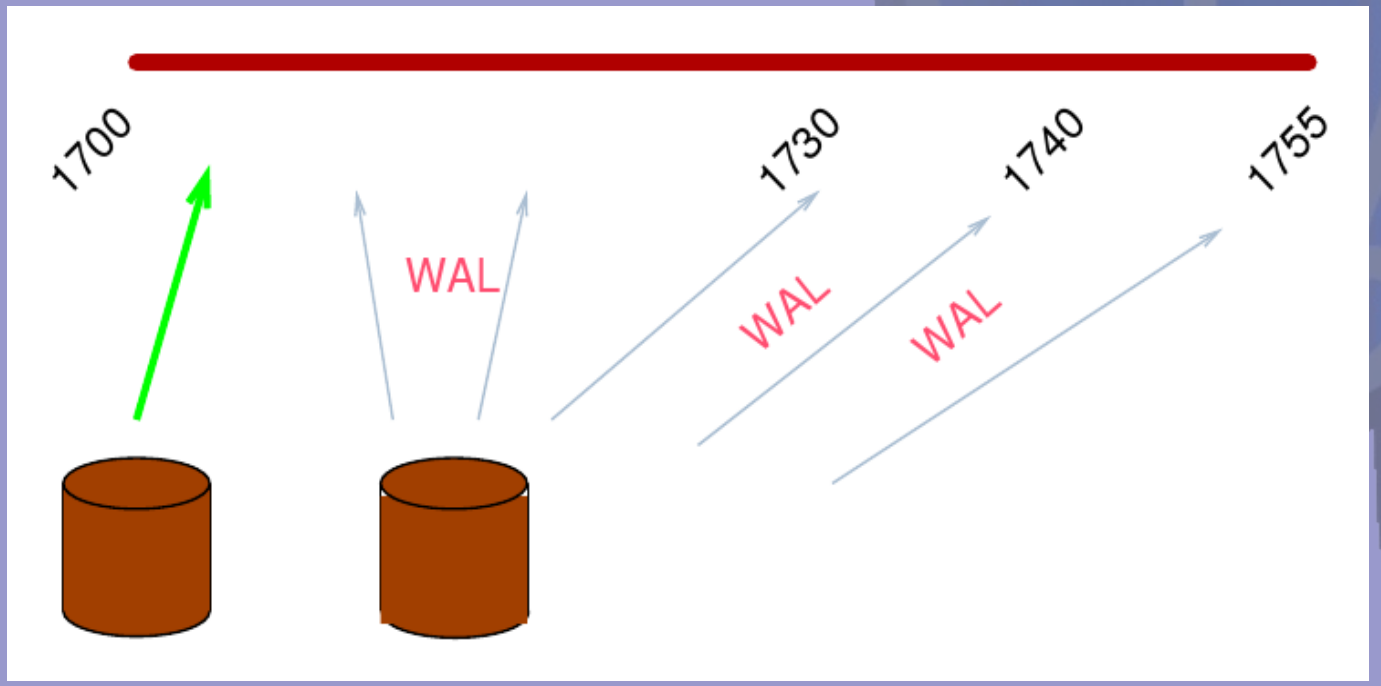
PITR & WAL

Il PITR sfrutta i log WAL (Write Ahead Log) per effettuare la “magia” del ripristino in modo molto elegante:

- Si archiviano i WAL in maniera continua
- Al riavvio dell'istanza si simula di avere una istanza “sporca”
- Il sistema inizia a ripercorrere i WAL per ricostruire la consistenza dei dati
- Si indica al sistema fino a quando percorrere i WAL, ottenendo così un ripristino ad un dato istante temporale.

PITR by pictures





PITR: esempio

- Per meglio comprendere come PITR funzioni, si supponga di avere una tabella che contiene una colonna timestamp (per vedere l'istante di ripristino) e di voler clonare il *cluster1* nel *cluster2*.

```
~> psql -U bsdmag -c "SELECT pg_start_backup('MY FIRST PITR');" bsdmagdb
 pg_start_backup
-----------------
 0/3C000020
(1 row)

~> cp -Rf /postgresql/cluster1/* /postgresql/cluster2/
~> rm /postgresql/cluster2/pg_xlog/*
~> rm /postgresql/cluster2/postmaster.pid
~> psql -U bsdmag -c "SELECT pg_stop_backup();" bsdmagdb
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
 pg_stop_backup
----------------
 0/3C000094
(1 row)
```

1 – si informa il cluster che si sta iniziando il backup fisico

2 – copia fisica dei dati (mentre il database sta girando!)

3 – si informa il cluster che la copia fisica è terminata

```
wal_level = archive
archive_mode = on
archive_command = 'cp -i %p /postgresql/pitr/%f'
```

0 – postgresql.conf deve contenere la configurazione su dove archiviare i WAL

PITR: esempio (2)

```
bsdmagdb=# INSERT INTO magazine(id, title)
VALUES( generate_series(1, 200000), 'BATCH-1');
INSERT 0 200000
bsdmagdb=# INSERT INTO magazine(id, title)
VALUES( generate_series(200001, 400000), 'BATCH-2');
INSERT 0 200000
bsdmagdb=# INSERT INTO magazine(id, title)
VALUES( generate_series(400001, 600000), 'BATCH-3');
INSERT 0 200000
bsdmagdb=# INSERT INTO magazine(id, title)
VALUES( generate_series(600001, 1000000), 'BATCH-4');
INSERT 0 400000
bsdmagdb=# SELECT ts::time, title FROM magazine GROUP BY ts,title ORDER BY title;

-----------------+---------
 08:22:42.982515 | BATCH-1
 08:22:51.401579 | BATCH-2
 08:23:03.243077 | BATCH-3
 08:23:17.011491 | BATCH-4
(4 rows)
bsdmagdb=# TRUNCATE TABLE magazine;
TRUNCATE TABLE
```

Inserimento di 4 gruppi di tuple da 2 milioni)

Si cancella tutto!

PITR: recovery

```
restore_command = 'cp /postgresql/pitr/%f "%p"'
recovery_target_time = '2012-01-20 08:23:18'
```

0 – recovery.conf deve contenere la locazione dei WAL e l'istante di ripristino (si esclude il BATCH 4)

```
~> service postgresql start
~> psql -U bsdmag -c "SELECT ts::time, title, count(title) FROM magazine GROUP BY ts,title ORDER BY title;" bsdmagdb
       ts        |  title  | count
-----------------+---------+--------
 08:22:42.982515 | BATCH-1 | 200000
 08:22:51.401579 | BATCH-2 | 200000
 08:23:03.243077 | BATCH-3 | 200000
~> cat /postgresql/cluster2/recovery.done
restore_command = 'cp /postgresql/pitr/%f "%p"'
recovery_target_time = '2012-01-20 08:23:18'
```

1 – all'avvio il cluster si accorge del recovery.conf, inizia la fase di recovery con i WAL archiviati

2 – il file viene rinominato in recovery.done per tenere traccia dell'avvio in PITR del cluster

PITR: avvio e ripristino

All'avvio del cluster i log contengono informazioni circa il replay delle transazioni come specificato nel file di recovery.

```
2010-07-16 09:07:49 GMT LOG:  database system was shut down at 2010-07-16 09:02:09 GMT
2010-07-16 09:07:49 GMT LOG:  starting point-in-time recovery to 2010-07-16 08:52:00+01
cp: cannot stat `/sviluppo/backup/pg_wal/000000010000000000000007E': No such file or directory
2010-07-16 09:07:50 GMT LOG:  consistent recovery state reached at 0/7E000078
2010-07-16 09:07:50 GMT LOG:  record with zero length at 0/7E000078
2010-07-16 09:07:50 GMT LOG:  redo is not required
cp: cannot stat `/sviluppo/backup/pg_wal/00000002.history': No such file or directory
2010-07-16 09:07:50 GMT LOG:  selected new timeline ID: 2
cp: cannot stat `/sviluppo/backup/pg_wal/00000001.history': No such file or directory
2010-07-16 09:07:50 GMT LOG:  archive recovery complete
2010-07-16 09:07:50 GMT FATAL:  the database system is starting up
2010-07-16 09:07:50 GMT LOG:  database system is ready to accept connections
2010-07-16 09:07:50 GMT LOG:  autovacuum launcher started
```

Considerazioni

- Il PITR è una tecnica molto potente ma richiede precisione nell'uso degli istanti temporali. E' possibile specificare una transazione come limite di recovery (quindi passare ad un tempo discreto) ma le transazioni potrebbero essere riordinate, quindi non si ha un effettivo vantaggio.
- Durante il ripristino i client possono collegarsi all'istanza, è opportuno disabilitare l'accesso tramite `pg_hba.conf` per tutta la durata del ripristino.
- Gli indici non vengono ripristinati, è necessario un `REINDEX`.

Replica

Sopravvivere ad un disastro!

Replication

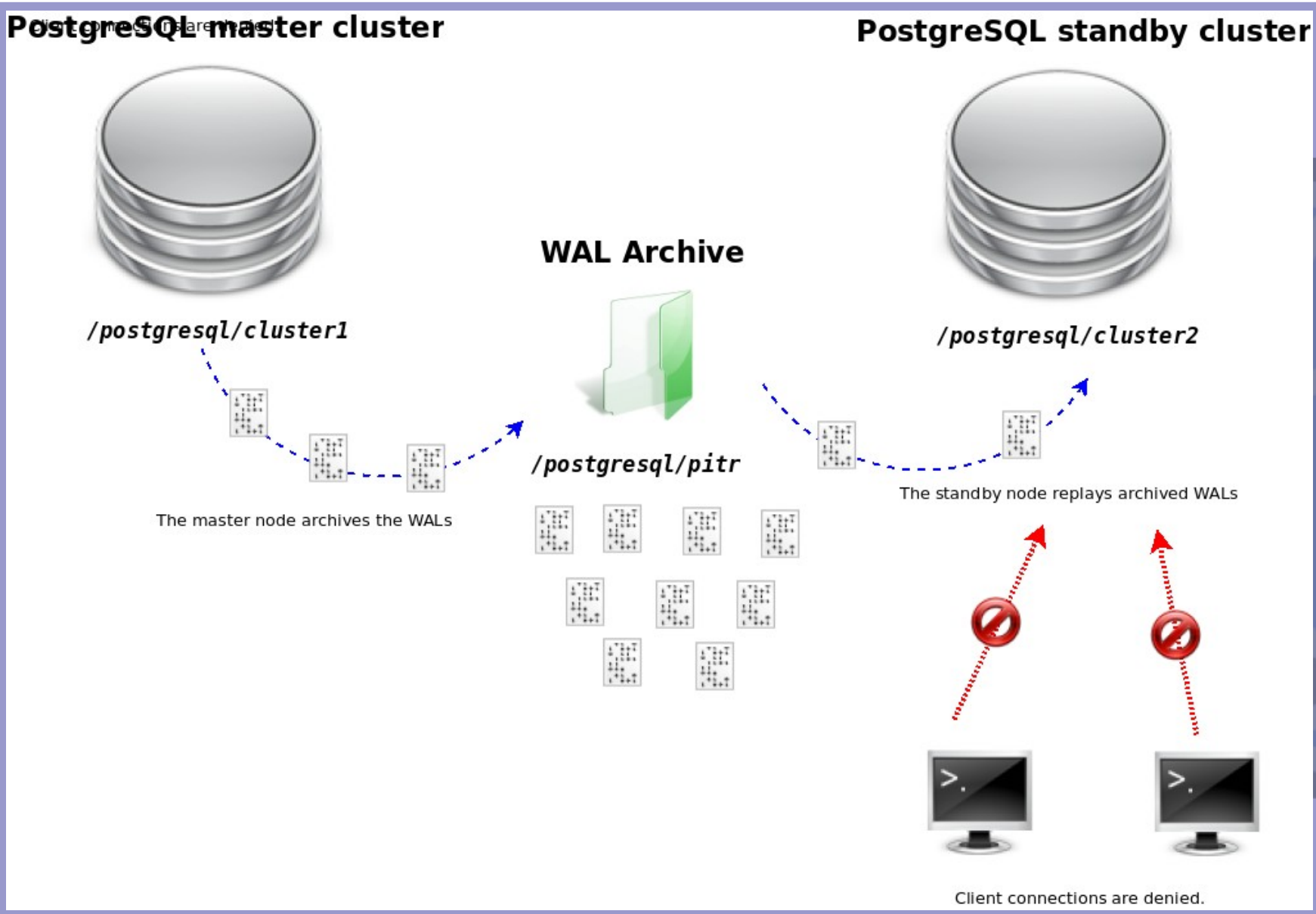
La replica di un cluster consente l'utilizzo in diversi scenari, in particolare:

- High Availability (HA): fornire accesso garantito 24x7
- Disaster Recovery: si sopperisce alla caduta disastrosa di un cluster

PostgreSQL dispone di soluzioni di replication fin dalla versione 7 (es. Mammoth Replicator, Slony) ma solo dalla serie 9 la replication viene inglobata nella installazione di default.

Disaster

Log Shipping Replication




```
$ psql -U bsdmag -p 5435 bsdmagdb
psql: FATAL: the database system is starting up
```

Log Shipping Replication: configurazione

```
wal_level='archive'
archive_mode=on
archive_command='cp -i %p /postgresql/pitr/%f'
```

La configurazione del master è simile a quella in PITR.

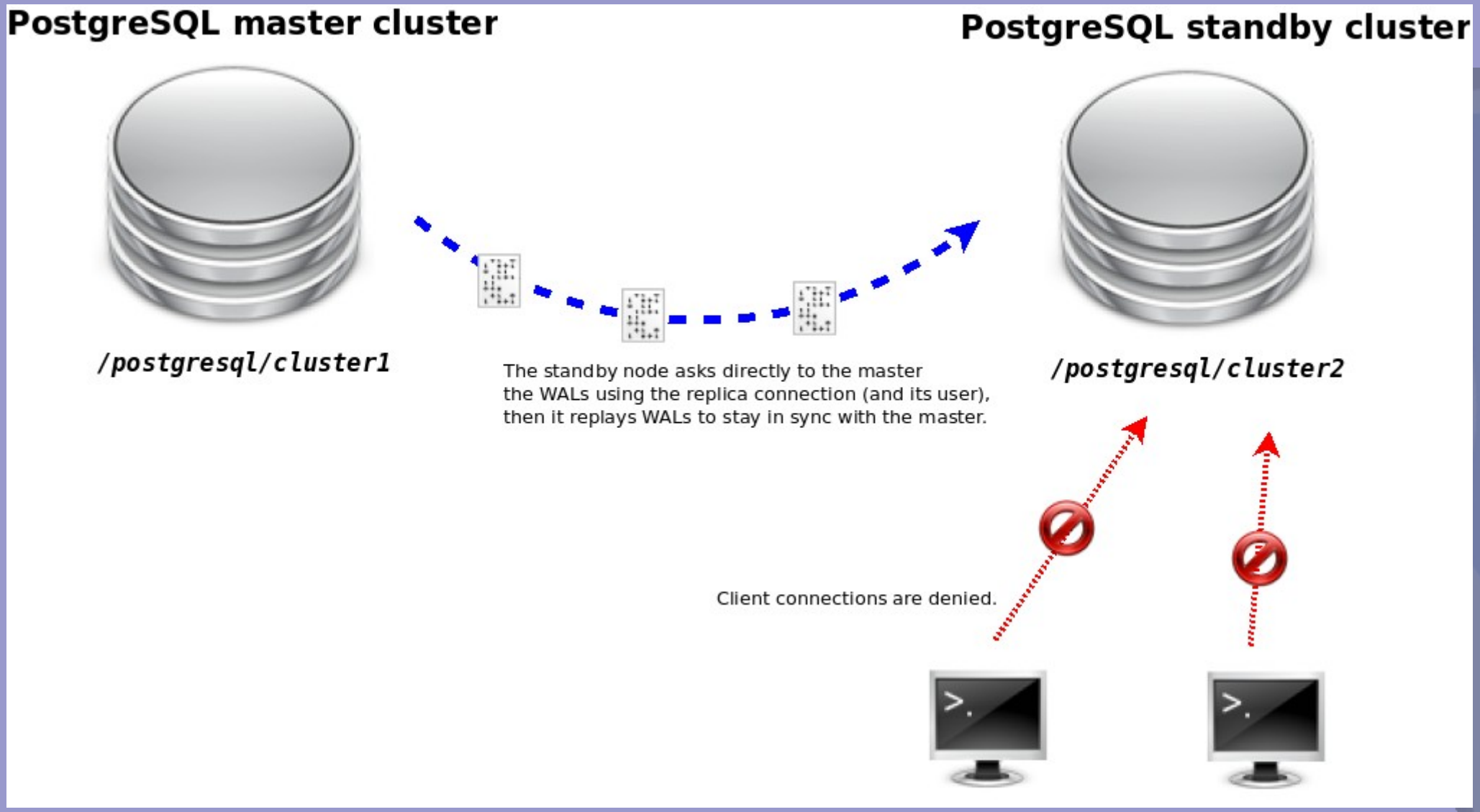
Lo slave non archivia i WAL, ed effettua un continuo ripristino (replay) dei WAL presi dall'archivio del master. Il cleanup serve a risparmiare spazio.

Il trigger file viene usato per far partire il clone come istanza singola (ferma la sincronizzazione).

```
wal_level='minimal'
archive_mode='off'

standby_mode='on'
restore_command='cp /postgresql/pitr/%f %p'
archive_cleanup_command='pg_archivecleanup /postgresql/
                         pitr %r'
trigger_file='/postgresql/standby.3.trigger'
```

Log Streaming Replication



Log Streaming Replication: configurazione del master

```
wal_level='archive'
archive_mode=on
archive_command='test 1 = 1'
archive_timeout=30
max_wal_senders=1
wal_keep_segments=50
```

Di fatto non c'è archiviazione, i log sono mandati direttamente allo slave, quindi ogni comando che ritorna uno status di "true" va bene. Il parametro archive_timeout indica ogni quanti secondi un WAL segment è forzatamente scritto (e quindi disponibile allo slave).

Il wal_keep_segments indica quanti segmenti di WAL (16 MB l'uno) devono essere tenuti per eventuali latenze di rete o disconnessioni temporanee dello slave.

I log sono inviati con connessione diretta mediante un utente che abbia il GRANT di REPLICATION.

```
CREATE USER replicator WITH REPLICATION LOGIN CONNECTION
LIMIT 1 PASSWORD 'replicatorPassword';
```

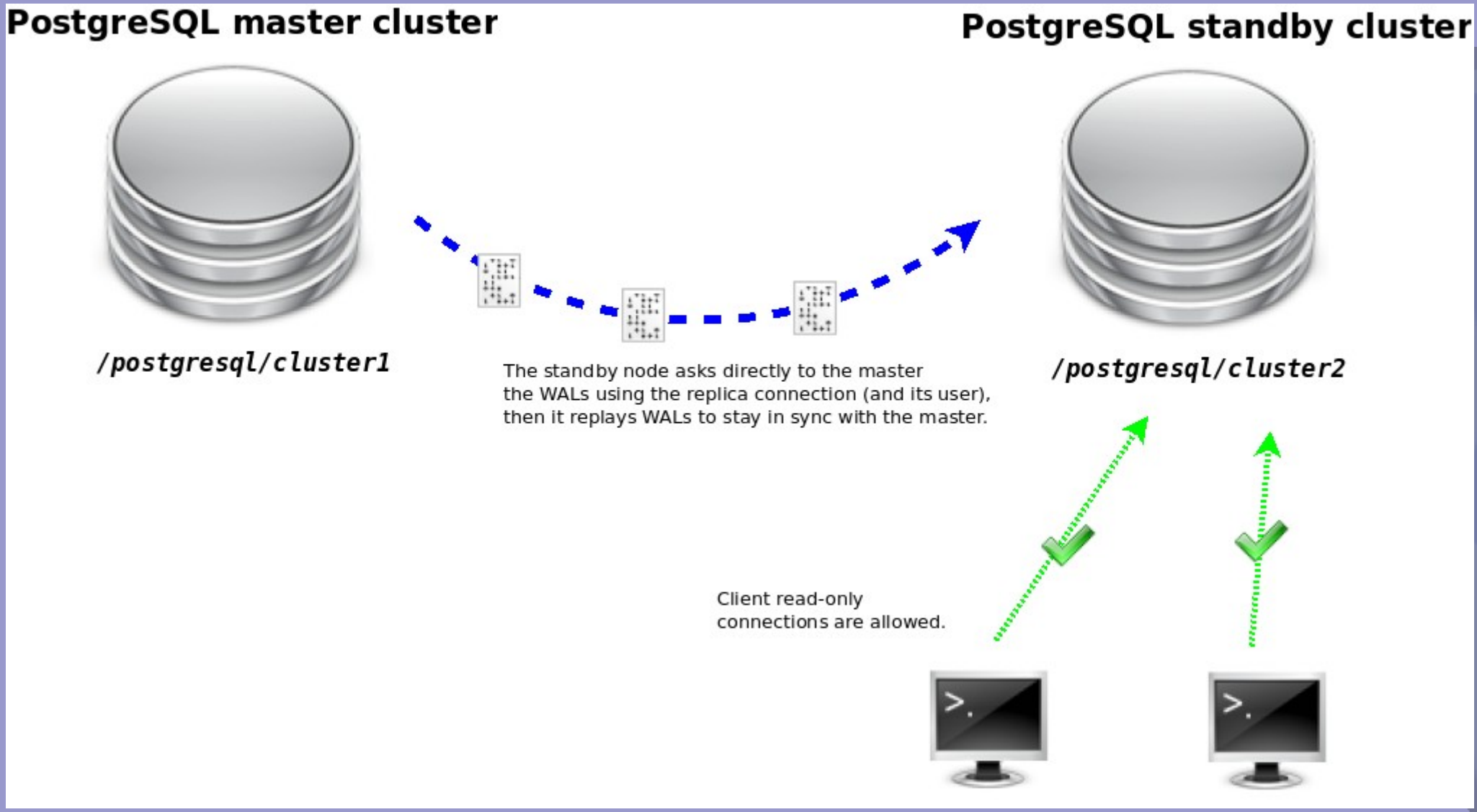
```
host replication replicator 192.168.200.0/24 trust
```

Log Streaming Replication: configurazione dello slave

```
standby_mode='on'
primary_conninfo=' host=192.168.200.2 user=replicator'
trigger_file='/postgresql/standby.4.trigger'
```

Lo slave ha un file recovery.conf che contiene le informazioni per la replica.

Hot Standby


PostgreSQL master cluster
PostgreSQL standby cluster
/postgresql/cluster1
The standby node asks directly to the master
the WALs using the replica connection (and its user),
then it replays WALs to stay in sync with the master.
/postgresql/cluster2
Client read-only
connections are allowed.

Hot Standby Replication: configurazione

```
wal_level='hot_standby'
archive_mode=on
archive_command='test 1 = 1'
archive_timeout=30
max_wal_senders=1
replication_timeout=60
wal_keep_segments=16
```

Simile alla configurazione per Log Streaming, ma senza archiviazione. Occorre avere un utente di connessione che gli slave useranno per ottenere le informazioni (segmenti WAL).

Lo slave ha un file recovery.conf che contiene le informazioni per la replica.

```
standby_mode='on'
primary_conninfo=' host=192.168.200.2 user=replicator'
trigger_file='/postgresql/standby.4.trigger'
```

```
hot_standby=on
```

Lo slave sa che deve accettare query in lettura.