**Practical PostgreSQL Backup & Restore**

Table of Contents

1

[Introduction](#gjdgxs)

2

[How PostgreSQL Maintains Consistency](#30j0zll)

2.1

[Write Ahead Log (WAL)](#1fob9te)

2.2

[Checkpoints](#3znysh7)

2.3

[Crash Recovery](#2et92p0)

3

[Setup](#tyjcwt)

3.1

[Introduction](#3dy6vkm)

3.2

[Setup PostgreSQL](#1t3h5sf)

3.3

[Sample Data](#4d34og8)

4

[pg\_dump](#2s8eyo1)

4.1

[Introduction](#17dp8vu)

4.2

[Usage](#3rdcrjn)

4.2.1

[Dumping data](#26in1rg)

4.2.2

[Restoring data](#lnxbz9)

5

[pg\_basebackup](#35nkun2)

5.1

[Introduction](#1ksv4uv)

5.2

[Usage](#44sinio)

5.2.1

[Create a Base Backup](#2jxsxqh)

5.2.2

[Restore a Base Backup](#z337ya)

6

[pgBackRest](#3j2qqm3)

6.1

[Introduction](#1y810tw)

6.2

[Installation](#4i7ojhp)

6.3

[Setup](#2xcytpi)

6.4

[Backup](#1ci93xb)

6.5

[Retention](#3whwml4)

6.6

[Restore](#2bn6wsx)

7

[Point-in-Time Recovery](#qsh70q)

1

Introduction

The purpose of this document is to familiarize the user with the core backup tools that are distributed with PostgreSQL as well as one third-party option, pgBackRest.

Docker is used to simulate a realistic environment so that actual backup strategies can be demonstrated. To logon to any Docker container run:

docker exec -it doc-<hostname> bash

Similarly, a command can be run directly without logging on:

docker exec -it doc-<hostname> <cmd>

A command can be run as a specific user:

docker exec -u postgres -it doc-<hostname> <cmd>

Pipes and redirection should be wrapped in a bash command:

docker exec -it doc-<hostname> bash -c 'echo "test" > testfile'

All commands are intended to be run as an unprivileged user that has sudo privileges for both the root and postgres users. It's also possible to run the commands directly as their respective users without modification and in that case the sudo commands can be stripped off.

Each main section includes a command to bring the environment to a state where that section's commands can be run. Thus, if the environment gets into a bad state its possible to reset it without starting from the beginning.

2

How PostgreSQL Maintains Consistency

2.1

Write Ahead Log (WAL)

WAL is the mechanism that PostgreSQL uses to ensure that no committed changes are lost. Transactions are written sequentially to the WAL and a transaction is considered to be committed when those writes are flushed to disk. Afterwards, a background process writes the changes into the main database cluster files (also known as the heap). In the event of a crash, the WAL is replayed to make the database consistent.

WAL is conceptually infinite but in practice is broken up into individual 16MB files called segments. WAL segments follow the naming convention 0000000100000A1E000000FE where the first 8 hexadecimal digits represent the timeline and the next 16 digits are the logical sequence number (LSN).

An exhaustive description of how WAL works can be found at [The Internals of PostgreSQL (Chapter 9)](http://www.interdb.jp/pg/pgsql09.html).

2.2

Checkpoints

PostgreSQL will periodically ensure that all data written to the WAL is flushed to disk. This is called a checkpoint. Checkpoints minimize the amount of WAL replay required after a crash and allows the database to reuse WAL segments applied before a checkpoint.

More information about checkpoints can be found in the [PostgreSQL Documentation](https://www.postgresql.org/docs/9.6/static/wal-configuration.html).

2.3

Crash Recovery

If PostgreSQL crashes, is terminated, or the server goes down, crash recovery must be performed when PostgreSQL restarts to ensure the database is consistent. PostgreSQL simply replays all the WAL since the last checkpoint. Any transactions that were in progress when PostgreSQL terminated will not be committed.

Backup recovery uses the same process except that WAL may need to be replayed from a checkpoint that is older than the last checkpoint. The backup\_label file lets PostgreSQL know which checkpoint it should start recovery from.

Replication also uses this same process to replay transactions from the primary onto the standby.

WAL replay is the central mechanism for crash recovery, backup recovery, and replication. Understanding how WAL works is the key to understanding how PostgreSQL achieves consistency.

3

Setup

3.1

Introduction

Setup the test cluster and some sample data that will be used during the training.

**Start working at this point by running**:

/docdynamo/doc/doc.pl --no-cache --doc-path=/training/doc \

    --include=backup-training --require=/setup/intro

3.2

Setup PostgreSQL

Create a cluster and start it.

db-primary **⇒** Create the test cluster

sudo -u postgres /usr/pgsql-9.6/bin/initdb -A peer -E UTF8 -k /var/lib/pgsql/9.6/data

By default PostgreSQL will only accept local connections. The examples in this training will require connections from other servers so listen\_addresses is configured to listen on all interfaces. This may not be appropriate for secure installations.

db-primary:/var/lib/pgsql/9.6/data/postgresql.conf **⇒** Set listen\_addresses

listen\_addresses = '\*'

For training purposes the log\_line\_prefix setting will be minimally configured. This keeps the log output as brief as possible to better illustrate important information.

db-primary:/var/lib/pgsql/9.6/data/postgresql.conf **⇒** Set log\_line\_prefix

listen\_addresses = '\*'

log\_line\_prefix = ''

By default PostgreSQL includes the day of the week in the log filename. This makes automating the user guide a bit more complicated so the log\_filename is set to a constant.

db-primary:/var/lib/pgsql/9.6/data/postgresql.conf **⇒** Set log\_filename

listen\_addresses = '\*'

log\_filename = 'postgresql.log'

log\_line\_prefix = ''

WAL senders are required for pg\_basebackup and replication so enable them.

db-primary:/var/lib/pgsql/9.6/data/postgresql.conf **⇒** Set max\_wal\_senders

listen\_addresses = '\*'

log\_filename = 'postgresql.log'

log\_line\_prefix = ''

max\_wal\_senders = 10

The WAL level must be set to replica.

db-primary:/var/lib/pgsql/9.6/data/postgresql.conf **⇒** Set wal\_level

listen\_addresses = '\*'

log\_filename = 'postgresql.log'

log\_line\_prefix = ''

max\_wal\_senders = 10

wal\_level = replica

db-primary **⇒** Start the test cluster

sudo systemctl start postgresql-9.6

3.3

Sample Data

Create a sample database to provide some data to backup up.

db-primary **⇒** Show sample database script

sudo -u postgres cat /training/script/setup-db.sql

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Create TEST1 database and connect to it  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
create database test1;  
\c test1;  
  
begin;  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
WIDGET Table  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

create table widget

(  
 id int not null,  
 name text not null,  
 constraint widget\_pk primary key (id)  
);  
  
-- Sample data  
insert into widget (id, name) values (1, 'thingamabob');  
insert into widget (id, name) values (2, 'whatsawhosit');  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
WIDGET\_DETAIL Table  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

create table widget\_detail

(  
 id int,  
 widget\_id int not null  
 constraint widgetdetail\_widgetid\_fk references widget (id),  
 key text not null,  
 value text not null,  
 constraint widgetdetail\_pk primary key (id),  
 constraint widgetdetail\_key\_id\_unq unique (key, id)  
);  
  
-- Sample data  
insert into widget\_detail (id, widget\_id, key, value) values (100, 1, 'color', 'yellowish');  
insert into widget\_detail (id, widget\_id, key, value) values (101, 1, 'shape', 'squarish');  
insert into widget\_detail (id, widget\_id, key, value) values (200, 2, 'color', 'redish');  
insert into widget\_detail (id, widget\_id, key, value) values (201, 2, 'shape', 'blobish');  
  
commit;

db-primary **⇒** Create sample database

sudo -u postgres psql -f /training/script/setup-db.sql

CREATE DATABASE  
You are now connected to database "test1" as user "postgres".  
BEGIN  
CREATE TABLE  
INSERT 0 1  
INSERT 0 1  
CREATE TABLE  
INSERT 0 1  
INSERT 0 1  
INSERT 0 1  
INSERT 0 1  
COMMIT

4

pg\_dump

4.1

Introduction

Pros:

* Readable format (when using plain).
* Very granular specific databases or objects can be dumped and restored.
* Generally smaller than a file-level backup because index data is not included.

Cons:

* No point-in-time recovery.
* Slow if many indexes and constraints need to be recreated.
* No backup management (i.e., expiring of old backups).

More information about pg\_dump can be found in the [PostgreSQL Documentation](https://www.postgresql.org/docs/9.6/static/app-pgdump.html).

**Start working at this point by running**:

/docdynamo/doc/doc.pl --no-cache --doc-path=/training/doc \

    --include=backup-training --require=/pg-dump/intro

4.2

Usage

4.2.1

Dumping data

pg\_dump can be used to dump an entire cluster. This is the simplest way to backup a database using the core utilities.

db-primary **⇒** Dump entire cluster

sudo -u postgres pg\_dumpall

Output suppressed for testing

pg\_dump can be used to dump a single database. The ability to do logical data dumps is the primary strength of pg\_dump.

db-primary **⇒** Dump test1 database

sudo -u postgres pg\_dump test1

Output suppressed for testing

It is also possible to dump a single object with pg\_dump.

db-primary **⇒** Dump detail table

sudo -u postgres pg\_dump -t widget\_detail test1

Output suppressed for testing

Or even just the data for a single object.

db-primary **⇒** Dump detail table

sudo -u postgres pg\_dump -a -t widget\_detail test1

Output suppressed for testing

There are literally a ton of options.

db-primary **⇒** pg\_dump options (some interesting options highlighted)

sudo -u postgres pg\_dump --help

Output suppressed for testing

4.2.2

Restoring data

The power of pg\_dump is the ability to dump and restore specific parts of a cluster.

db-primary **⇒** Dump test1 database

sudo -u postgres bash -c 'pg\_dump test1 > /var/lib/pgsql/9.6/backups/db-test1.dump'

Output suppressed for testing

db-primary **⇒** Restore the dump to the test2 database

sudo -u postgres psql -c 'create database test2'

Output suppressed for testing

sudo -u postgres psql -f /var/lib/pgsql/9.6/backups/db-test1.dump test2

Output suppressed for testing

db-primary **⇒** Query test2 database to demonstrate success

sudo -u postgres psql -c 'select \* from widget' test2

Output suppressed for testing

Unfortunately, psql is not very concerned with errors by default.

db-primary **⇒** Run again to show errors

sudo -u postgres psql -f /var/lib/pgsql/9.6/backups/db-test1.dump test2

Output suppressed for testing

But it is possible to fix this with some sed magic.

db-primary **⇒** Force psql to exit on first error

sudo -u postgres bash -c ' \  
 cat /var/lib/pgsql/9.6/backups/db-test1.dump | \  
 sed -e "1s/^/\\\\set QUIET on\n\\\\set ON\_ERROR\_STOP on\nbegin transaction;\n/" -e "\$acommit;\n" | \  
 psql test2'

Output suppressed for testing

5

pg\_basebackup

5.1

Introduction

pg\_basebackup is a core tool that performs file-level backups of a PostgreSQL cluster.

Pros:

* Allows point-in-time recovery.
* Indexes do not need to be recreated.

Cons:

* Single-threaded.
* Not granular — the entire cluster must be backed up.
* No archive command for WAL generated in between backups (may use pg\_receive\_xlog).
* No backup or WAL management (i.e., expiring of old backups and WAL).

More information about pg\_basebackup can be found in the [PostgreSQL Documentation](https://www.postgresql.org/docs/9.6/static/app-pgbasebackup.html).

**Start working at this point by running**:

/docdynamo/doc/doc.pl --no-cache --doc-path=/training/doc \

    --include=backup-training --require=/pg-basebackup/intro

5.2

Usage

5.2.1

Create a Base Backup

pg\_basebackup always makes a backup of the entire cluster.

db-primary **⇒** Enable replication for postgres user

sudo -u postgres bash -c 'echo "local replication postgres peer" >> /var/lib/pgsql/9.6/data/pg\_hba.conf'

Output suppressed for testing

sudo systemctl reload postgresql-9.6

db-primary **⇒** Backup entire cluster

sudo -u postgres pg\_basebackup --checkpoint=fast -D /var/lib/pgsql/9.6/backups/backup1

Output suppressed for testing

pg\_basebackup does not copy WAL to the backup directory by default. Checking the pg\_xlog directory demonstrates that no WAL was copied.

db-primary **⇒** Check for WAL in backup pg\_xlog directory

sudo -u postgres ls -lah /var/lib/pgsql/9.6/backups/backup1/pg\_xlog

Output suppressed for testing

Try again using the option to automatically copy WAL.

db-primary **⇒** Backup entire cluster with WAL

sudo -u postgres rm -rf /var/lib/pgsql/9.6/backups/backup1

Output suppressed for testing

sudo -u postgres pg\_basebackup --checkpoint=fast -X stream -D /var/lib/pgsql/9.6/backups/backup1

Output suppressed for testing

Now the WAL required to make the backup consistent is present.

db-primary **⇒** Check for WAL in backup pg\_xlog directory

sudo -u postgres ls -lah /var/lib/pgsql/9.6/backups/backup1/pg\_xlog

Output suppressed for testing

5.2.2

Restore a Base Backup

pg\_basebackup does not have a restore command so the simplest way to restore a base backup is to replace the existing data directory.

db-primary **⇒** Stop PostgreSQL

sudo systemctl stop postgresql-9.6

db-primary **⇒** Move current PDATA to a new directory

sudo mv /var/lib/pgsql/9.6/data /var/lib/pgsql/9.6/data.save

db-primary **⇒** Copy base backup to PDATA and fix permissions

sudo cp -r /var/lib/pgsql/9.6/backups/backup1 /var/lib/pgsql/9.6/data

sudo chown -R postgres:postgres /var/lib/pgsql/9.6/data

sudo chmod -R u+r,u+w,g=,o= /var/lib/pgsql/9.6/data

db-primary **⇒** Start PostgreSQL

sudo systemctl start postgresql-9.6

db-primary **⇒** Check the PostgreSQL log to demonstrate that recovery was successful

sudo -u postgres cat /var/lib/pgsql/9.6/data/pg\_log/postgresql.log

Output suppressed for testing

6

pgBackRest

6.1

Introduction

pgBackRest is an alternative to the core utilities that uses the low-level backup method to produce a backup similar to pg\_basebackup. Other tools of this type include [Barman](http://www.pgbarman.org/), [WAL-E](https://github.com/wal-e/wal-e)/[WAL-G](https://github.com/wal-g/wal-g), [OmniPITR](https://github.com/omniti-labs/omnipitr), [PGHoard](https://github.com/ohmu/pghoard), and [pg\_probackup](https://github.com/postgrespro/pg_probackup).

pgBackRest includes most of the important features of the other tools, though each has its specialties.

Pros:

* Allows point-in-time recovery.
* Indexes do not need to be recreated.
* Built-in archive command.
* Built-in backup and WAL management.
* Multi-threaded.

Cons:

* Not distributed with PostgreSQL.

More information about low-level backups can be found in the [PostgreSQL Documentation](https://www.postgresql.org/docs/9.6/static/continuous-archiving.html).

More information about pgBackRest can be found at [the website](http://www.pgbackrest.org).

**Start working at this point by running**:

/docdynamo/doc/doc.pl --no-cache --doc-path=/training/doc \

    --include=backup-training --require=/pgbackrest/intro

6.2

Installation

pgBackRest can be installed from [yum.postgresql.org](https://yum.postgresql.org/) using the same repository as PostgreSQL.

db-primary **⇒** Install pgBackRest

sudo -u postgres pgbackrest

Output suppressed for testing

6.3

Setup

db-primary:/etc/pgbackrest.conf **⇒** Configure the PostgreSQL cluster data directory and repository path

Config suppressed for testing

db-primary:/var/lib/pgsql/9.6/data/postgresql.conf **⇒** Configure PostgreSQL

Config suppressed for testing

db-primary **⇒** Restart PostgreSQL to enable WAL archiving

sudo -u postgres bash -c 'echo "local replication postgres peer" >> /var/lib/pgsql/9.6/data/pg\_hba.conf'

Output suppressed for testing

sudo systemctl restart postgresql-9.6

db-primary **⇒** Create the main stanza

sudo -u postgres pgbackrest --stanza=main stanza-create

Output suppressed for testing

sudo -u postgres pgbackrest --stanza=main info

Output suppressed for testing

db-primary **⇒** Check that WAL archiving configured

sudo -u postgres pgbackrest --stanza=main check

Output suppressed for testing

sudo -u postgres pgbackrest --stanza=main info

Output suppressed for testing

6.4

Backup

db-primary **⇒** Perform first full backup

sudo -u postgres pgbackrest --stanza=main --log-level-console=info backup

Output suppressed for testing

The first backup must be full so pgBackRest will change the backup type if necessary.

db-primary **⇒** Perform differential backup

sudo -u postgres pgbackrest --stanza=main --type=diff backup

Output suppressed for testing

db-primary **⇒** Display backup info

sudo -u postgres pgbackrest --stanza=main info

Output suppressed for testing

6.5

Retention

db-primary:/etc/pgbackrest.conf **⇒** Configure backup retention

Config suppressed for testing

db-primary **⇒** Perform second full backup

sudo -u postgres pgbackrest --stanza=main --type=full backup

db-primary **⇒** Perform third full backup and check that the first full backup has expired

sudo -u postgres pgbackrest --stanza=main --type=full backup

sudo -u postgres pgbackrest --stanza=main info

Output suppressed for testing

6.6

Restore

db-primary **⇒** Attempt a restore

sudo -u postgres pgbackrest --stanza=main --delta restore

Output suppressed for testing

Why the failure? PostgreSQL must be stopped before a restore can be performed.

db-primary **⇒** Stop PostgreSQL

sudo systemctl stop postgresql-9.6

db-primary **⇒** Perform restore

sudo -u postgres pgbackrest --stanza=main --delta --log-level-console=detail restore

Output suppressed for testing

pgBackRest automatically create the recovery.conf required to fetch WAL from the archive

db-primary **⇒** Show recovery.conf file

sudo -u postgres cat /var/lib/pgsql/9.6/data/recovery.conf

Output suppressed for testing

During the backup PostgreSQL create a backup\_label which defines the checkpoint where WAL replay must start.

db-primary **⇒** Show backup\_label file

sudo -u postgres cat /var/lib/pgsql/9.6/data/backup\_label

Output suppressed for testing

db-primary **⇒** Start PostgreSQL

sudo systemctl start postgresql-9.6

db-primary **⇒** Check the PostgreSQL log to demonstrate that recovery was successful

sudo -u postgres cat /var/lib/pgsql/9.6/data/pg\_log/postgresql.log

Output suppressed for testing

Note that a new timeline has been started. This happens whenever a cluster in recovery is promoted. The new timeline prevents duplicate WAL in the repo when a recovery does not play all the way to the end of the WAL stream.

db-primary **⇒** Check repository for the new timeline

sudo -u postgres pgbackrest --stanza=main check

sudo -u postgres pgbackrest --stanza=main info

Output suppressed for testing

7

Point-in-Time Recovery

The [Restore](#2bn6wsx) performed default recovery, which is to play all the way to the end of the WAL stream. In the case of a hardware failure this is usually the best choice but for data corruption scenarios (whether machine or human in origin) Point-in-Time Recovery (PITR) is often more appropriate.

Point-in-Time Recovery (PITR) allows the WAL to be played from the last backup to a specified time, transaction id, or recovery point. For common recovery scenarios time-based recovery is arguably the most useful. A typical recovery scenario is to restore a table that was accidentally dropped or data that was accidentally deleted. Recovering a dropped table is more dramatic so that's the example given here but deleted data would be recovered in exactly the same way.

**Start working at this point by running**:

/docdynamo/doc/doc.pl --no-cache --doc-path=/training/doc \

    --include=backup-training --require=/pgbackrest/setup

db-primary **⇒** Backup the main cluster and create a table with very important data

sudo -u postgres pgbackrest --stanza=main --type=diff backup

sudo -u postgres psql -c "begin; \  
 create table important\_table (message text); \  
 insert into important\_table values ('Important Data'); \  
 commit; \  
 select \* from important\_table;"

Output suppressed for testing

It is important to represent the time as reckoned by PostgreSQL and to include timezone offsets. This reduces the possibility of unintended timezone conversions and an unexpected recovery result.

db-primary **⇒** Get the time from PostgreSQL

sudo -u postgres psql -Atc "select current\_timestamp"

Output suppressed for testing

Now that the time has been recorded the table is dropped. In practice finding the exact time that the table was dropped is a lot harder than in this example. It may not be possible to find the exact time, but some forensic work should be able to get you close.

db-primary **⇒** Drop the important table

sudo -u postgres psql -c "begin; \  
 drop table important\_table; \  
 commit; \  
 select \* from important\_table;"

Output suppressed for testing

Now the restore can be performed with time-based recovery to bring back the missing table.

db-primary **⇒** Stop PostgreSQL, restore the main cluster to [Test Variable], and display recovery.conf

sudo systemctl stop postgresql-9.6

sudo -u postgres pgbackrest --stanza=main --delta \  
 --type=time "--target=[Test Variable]" restore

sudo -u postgres cat /var/lib/pgsql/9.6/data/recovery.conf

Output suppressed for testing

The recovery.conf file has been automatically generated by pgBackRest so PostgreSQL can be started immediately. Once PostgreSQL has finished recovery the table will exist again and can be queried.

db-primary **⇒** Start PostgreSQL and check that the important table exists

sudo systemctl start postgresql-9.6

sudo -u postgres psql -c "select \* from important\_table"

Output suppressed for testing

The PostgreSQL log also contains valuable information. It will indicate the time and transaction where the recovery stopped and also give the time of the last transaction to be applied.

db-primary **⇒** Examine the PostgreSQL log output

sudo -u postgres cat /var/lib/pgsql/9.6/data/pg\_log/postgresql.log

Output suppressed for testing

This example was rigged to give the correct result. If a backup after the required time is chosen then PostgreSQL will not be able to recover the lost table. PostgreSQL can only play forward, not backward. To demonstrate this the important table must be dropped (again).

db-primary **⇒** Drop the important table (again)

sudo -u postgres psql -c "begin; \  
 drop table important\_table; \  
 commit; \  
 select \* from important\_table;"

Output suppressed for testing

Now take a new backup and attempt recovery from the new backup.

db-primary **⇒** Perform a backup then attempt recovery from that backup

sudo -u postgres pgbackrest --stanza=main --type=incr backup

sudo systemctl stop postgresql-9.6

sudo -u postgres pgbackrest --stanza=main --delta \  
 --type=time "--target=[Test Variable]" restore

sudo systemctl start postgresql-9.6

sudo -u postgres psql -c "select \* from important\_table"

Output suppressed for testing

Looking at the log output it's not obvious that recovery failed to restore the table. The key is to look for the presence of the recovery stopping before... and last completed transaction... log messages. If they are not present then the recovery to the specified point-in-time was not successful.

db-primary **⇒** Examine the PostgreSQL log output to discover the recovery was not successful

sudo -u postgres cat /var/lib/pgsql/9.6/data/pg\_log/postgresql.log

Output suppressed for testing

Using an earlier backup will allow PostgreSQL to play forward to the correct time. The info command can be used to find the next to last backup.

db-primary **⇒** Get backup info for the main cluster

sudo -u postgres pgbackrest info

Output suppressed for testing

The default behavior for restore is to use the last backup but an earlier backup can be specified with the --set option.

db-primary **⇒** Stop PostgreSQL, restore from the selected backup, and start PostgreSQL

sudo systemctl stop postgresql-9.6

sudo -u postgres pgbackrest --stanza=main --delta \  
 --type=time "--target=[Test Variable]" \  
 --set=[Test Variable] restore

sudo systemctl start postgresql-9.6

sudo -u postgres psql -c "select \* from important\_table"

Output suppressed for testing

Now the the log output will contain the expected recovery stopping before... and last completed transaction... messages showing that the recovery was successful.

db-primary **⇒** Examine the PostgreSQL log output for log messages indicating success

sudo -u postgres cat /var/lib/pgsql/9.6/data/pg\_log/postgresql.log

Output suppressed for testing

Copyright © 2015-2017, The PostgreSQL Global Development Group, [MIT License](https://github.com/pgbackrest/training/blob/master/LICENSE). Updated September 21, 2017