

목차

PostgreSQL 백업 및 복구 - 명령어 사용

3

SQL 덤프

3

파일 시스템 레벨

5

연속 아카이빙

6

서버가 실행중인 경우

6

서버가 중지된 경우

7

참조링크

8

PostgreSQL 백업 및 복구 - 명령어 사용

PostgreSQL 데이터를 백업하는 3가지 방법이 있다.

- SQL 덤프
- 파일 시스템 레벨
- 연속 아카이빙

방법	설명	서버 실행 여부	장점	단점
SQL 덤프	<p>pg_dump, pg_dumpall과 같은 명령어를 통해 백업</p> <p>pg_restore, psql과 같은 명령어를 통해 복구</p>	AgensGraph 서버가 실행되고 있는 상태에서 가능	<p>데이터 압축, 분할, 커스텀 백업 설정 가능</p> <p>부분 및 전체 데이터 백업 가능</p> <p>머신 아키텍처(32비트에서 64비트 전송 가능) 상관 없음</p>	<p>서버가 실행되지 않는 상태에서 명령어 사용 불가능</p> <p>충분 데이터 반영 불가능</p>
연속 아카이빙(PITR)	<p>pg_basebackup 명령어와 wal 폴더 내용을 백업 및 복구</p> <p>tar 명령어와 wal 폴더 내용을 백업 및 복구</p>	<p>AgensGraph 서버가 실행되고 있는 상태에서 가능</p> <p>AgensGraph 서버가 실행되지 않는 상태에서 가능</p>	<p>충분 데이터 반영 가능</p>	<p>부분 데이터가 아닌 전체 데이터 복구 가능</p> <p>머신 아키텍처 상관 있음</p>
파일 시스템 레벨	tar 명령어를 통해 백업 및 복구	AgensGraph 서버가 실행되지 않는 상태에서 가능	손쉬운 백업	<p>충분 데이터 반영 불가능</p> <p>머신 아키텍처 상관 있음</p>

SQL 덤프

명령어를 통해 PostgreSQL 서버가 실행되고 있는 상태에서 PostgreSQL 데이터를 백업 및 복구가 가능하다. 여기서 사용되는 명령어는 pg_dump, pg_dumpall, pg_restore, psql이 사용 가능하며, 이 4가지 명령어는 다음 표와 같은 특징을 가진다.

pg_dump	백업	하나의 데이터베이스만 백업 가능 데이터 압축, 분할, 커스텀 백업 설정 가능	role 또는 tablespace 백업 안됨
pg_dumpall	백업	데이터베이스 전체 내용 백업 가능 role 또는 tablespace 백업 가능	
pg_restore	복구	비 텍스트 파일 덤프에서 가능 pg_dump로 백업된 파일 복구 가능	pg_dumpall로 백업된 파일은 사용 안됨
psql	복구	텍스트 파일 덤프에서 가능 pg_dump, pg_dumpall로 백업된 파일 복구 가능	

위 4가지 명령어에 대한 간단한 시나리오는 다음과 같다.

1. pg_dump의 기본 백업 → psql 복구
2. pg_dump의 커스텀 백업 → pg_restore 복구
3. pg_dumpall 백업 → psql 복구

1.에 대한 상세 실행 결과

실행 환경은 하나의 서버에서 2개 port를 사용해 테스트한다. 이 예제에서는 graph_dump라는 데이터베이스를 백업하고 복구하는 예제이다. pg_dump와 agens 명령어는 터미널 창을 통해 실행할 수 있다. 터미널 창을 열어 PostgreSQL 서버를 시작한 후 pg_dump 명령어를 통해 텍스트 형태의 덤프 파일을 만든다. 이러한 덤프 파일은 agens 명령어로 복구할 수 있으며, 복구할 데이터베이스는 새로운 서버에 미리 생성이 되어 있어야 한다.

```
1.PostgreSQL 서버1 시작
$ pg_ctl -D /path/to/server1/data start

2.백업 실행
$ pg_dump dump > dump.backup

3.PostgreSQL 서버2 시작
$ pg_ctl -D /path/to/server2/data start

4.데이터베이스 생성
$ createdb -p 5433 dump

5.복구 실행
$ psql -p 5433 dump < dump.backup

6.복구 확인
$ psql -p 5433 dump
```

2.에 대한 상세 실행 결과 실행 환경은 하나의 서버에서 2개 port를 사용해 테스트한다. 이 예제에서는 pg_c_dump라는 데이터베이스를 백업하고 복구하는 예제이다. pg_dump와 pg_restore 명령어는 터미널 창을 통해 실행할 수 있다. 터미널 창을 열어 PostgreSQL 서버를 시작한 후 pg_dump 명령어를 통해 비 텍스트 형태의 덤프 파일을 만든다. 비 텍스트 형태의 덤프 파일을 만들기 위해 -Fc라는 옵션을 추가한다. 이러한 덤프 파일은 비 텍스트 형태라서 psql 명령어로 복구할 수 없고, pg_restore 명령어로 복구할 수 있다. 복구할 데이터베이스는 새로운 서버에 미리 생성이 되어 있어야 한다.

```
1.PostgreSQL 서버1 시작
$ pg_ctl -D /path/to/server1/data start

2.백업 실행
$ pg_dump -Fc pg_c_dump > pg_c_dump.backup

3.PostgreSQL 서버2 시작
$ pg_ctl -D /path/to/server2/data start

4.데이터베이스 생성
$ createdb -p 5433 pg_c_dump
```

```
5.복구 실행
$ pg_restore -p 5433 -d pg_c_dump pg_c_dump.backup

6.복구 확인
$ psql -p5433 pg_c_dump
```

3.에 대한 상세 실행 결과 실행 환경은 하나의 서버에서 2개 port를 사용해 테스트한다. 이 예제에서는 모든 데이터베이스를 백업하고 복구하는 예제이다. 확인을 위해 `pg_dumpall`라는 데이터베이스를 사용하였다. `pg_dumpall`과 `psql` 명령어는 터미널 창을 통해 실행할 수 있다. 터미널 창을 열어 PostgreSQL 서버를 시작한 후 `pg_dumpall` 명령어를 통해 텍스트 형태의 덤프 파일을 만든다. 이러한 덤프 파일은 `psql` 명령어로 복구할 수 있으며, 복구할 데이터베이스는 새로운 서버에 미리 생성할 필요는 없다. 하지만 시작할 기존 데이터베이스 이름을 지정해야 하며, 일반적으로 `postgres` 지정한다. `pg_dumpall` 명령어에 대한 여러 옵션의 설명은

<https://www.postgresql.org/docs/current/static/app-pg-dumpall.html>에서 확인할 수 있다.

```
1.PostgreSQL 서버1 시작
$ pg_ctl -D /path/to/server1/data start

2.백업 실행
$ pg_dumpall > pg_dumpall.backup

3.PostgreSQL 서버2 시작
$ pg_ctl -D /path/to/server2/data start

5.복구 실행
$ psql -p 5433 -f pg_dumpall.backup postgres

6.복구 확인
$ psql -p 5433 pg_dumpall
```

파일 시스템 레벨

이 예제에서는 모든 데이터베이스를 백업하고 복구하는 예제이다. 확인을 위해 `pg_backup`이라는 데이터베이스를 사용하였다. 백업 및 복구는 `tar` 명령어로 터미널 창을 통해 실행할 수 있다. 백업은 서버를 정지시킨 다음에 `data` 폴더를 `tar` 명령어로 압축한다. `tar` 명령어로 압축 시 `c` 옵션은 새로운 압축 파일을 만든다는 의미이고, `f` 옵션은 압축 파일의 이름을 사용자가 지정한다는 의미이다. 복구는 `tar` 명령어로 압축해제를 한 후 압축해제된 폴더를 `pg_ctl` 명령어로 실행한다. 압축해제에서 사용한 `-C` 옵션은 압축해제된 파일이 위치할 경로이다. 주의할 점은 압축해제 폴더는 서버 시작시 `700` 권한이 필요하기 때문에 폴더를 생성할 때 `-m` 옵션을 통해 권한을 설정하고, 부모 폴더가 생성되어 있지 않을 경우를 대비해 `-p` 옵션을 추가한다.

```
1.PostgreSQL 서버 정지
$ pg_ctl -D /path/to/server/data stop

2.백업 실행
$ cd /path/to/server/data
$ tar cf backup.tar *
```

3. 복구 실행

```
$ mkdir -m 700 -p /path/to/server/backup/data
$ tar xf backup.tar -C /path/to/server/backup/data
```

4. PostgreSQL 서버 시작

```
$ pg_ctl -D /path/to/server/backup/data start
```

5. 확인

```
$ psql pg_backup
```

연속 아카이빙

이 예제에서는 모든 데이터베이스를 백업하고 복구하는 예제이다. 서버가 실행중인 경우와 서버가 중지된 경우로 각각 나눠서 진행한다.

서버가 실행중인 경우

기본 데이터는 `pg_basebackup` 명령어를 통해 가져오고, 증분 데이터는 `pg_wal` 폴더(PG 9.x 기반에서는 `pg_xlog` 폴더명을 사용했는데 PG 10 기반에서는 `pg_wal`로 폴더명이 변경됨)에 있는 WAL(Write Ahead Log) 파일에 기록이 된다.

`pg_wal` 폴더의 파일을 관리하기 위해 `wals`이라는 폴더를 만들어 이 폴더로 WAL 파일을 주기적으로 복사하고, 복구 과정에서는 복사된 WAL 파일을 사용한다. 진행 절차는 다음과 같다.

- WAL 파일을 `backup_wal`이라는 폴더에 주기적으로(1분마다) 복사한다. 이를 위해 `postgresql.conf` 파일에 있는 몇가지 옵션을 수정한다.

```
wal_level = archive
archive_mode = on
archive_command = 'cp %p /home/bylee/test/backup/wals/%f'
archive_timeout = 1min
```

- `pg_basebackup` 명령어를 통해 새로운 `data` 폴더를 생성한다.

```
pg_basebackup -D /home/bylee/test/data
```

- `recovery.conf` 파일을 `data` 폴더 내에 생성하고, `restore_command` 옵션을 사용해 WAL 파일을 복구한다.

전체 증분 데이터 복구와 특정 시점까지 증분 데이터 복구 중 특정 시점까지 증분 데이터 복구에 대해서만 테스트를 한다.

전체 증분 데이터 복구(됨)

```
restore_command = 'cp /home/bylee/test/backup/wals/%f %p'
```

특정 시점까지 증분 데이터 복구(안됨)

```
restore_command = 'cp /home/bylee/test/backup/wals/%f %p'
recovery_target_time = '2018-05-09 13:50:00'
```

- 서버를 시작한다.
- 복구가 완료되면 `recovery.conf` 파일은 `recovery.done` 파일로 변경된다.

특정 시점까지 증분 데이터 복구와 관련된 문서:

<https://www.postgresql.org/docs/current/static/recovery-target-settings.html>

서버가 중지된 경우

압축된 기본 데이터는 `tar` 명령어를 통해 압축해제한다. 증분 데이터는 압축해제된 폴더의 `pg_wal` 폴더에 있는 WAL 파일을 통해 복구를 하거나, 실행중인 서버의 `data` 폴더의 `pg_wal` 폴더에 있는 WAL 파일을 통해 복구가 가능하다.

증분 데이터 복구는 위 2가지 방법을 사용하지 않고, 3.1에서 사용한 방법을 사용한다. 진행 절차는 다음과 같다.

- WAL 파일을 `backup_wal`이라는 폴더에 주기적으로(1분마다) 복사한다.
- 이를 위해 `postgresql.conf` 파일에 있는 몇가지 옵션을 수정한다.

```
wal_level = archive
archive_mode = on
archive_command = 'cp %p /home/bylee/test/backup/wals/%f'
archive_timeout = 1min
```

- `tar` 명령어로 압축한 `data` 폴더를 압축해제한다. `data` 폴더 내 `pg_wal` 폴더를 삭제한다.

```
cd /path/to/data
tar cf backup.tar *
tar xf backup.tar /path/to/new/data
```

- `recovery.conf` 파일을 `/path/to/new/data` 폴더 내에 생성하고, `restore_command` 옵션을 사용해 WAL 파일을 복구한다. 전체 증분 데이터 복구와 특정 시점까지 증분 데이터 복구 중 특정 시점까지 증분 데이터 복구에 대해서만 테스트를 한다.

전체 증분 데이터 복구

```
restore_command = 'cp /home/bylee/test/backup/wals/%f %p'
```

특정 시점까지 증분 데이터 복구

```
restore_command = 'cp /home/bylee/test/backup/wals/%f %p'
recovery_target_time = '2018-05-09 13:50:00'
```

- 서버를 시작한다.

복구가 완료되면 `recovery.conf` 파일은 `recovery.done` 파일로 변경된다. 특정 시점까지 증분 데이터 복구와 관련된 문서: <https://www.postgresql.org/docs/current/static/recovery-target-settings.html>

참조링크

- <https://bylee5.tistory.com/73>

From: <https://atl.kr/dokuwiki/> - AllThatLinux!

Permanent link: https://atl.kr/dokuwiki/doku.php/postgresql_%EB%B0%B1%EC%97%85_%EB%B0%8F_%EB%B3%B5%EA%B5%AC_-_%EB%AA%85%EB%A0%B9%EC%96%B4_%EC%82%AC%EC%9A%A9

Last update: 2023/01/30 11:32

