

INSY

5CHIT 2018/19

Backup Strategien

Aufgabe

Mario Fentler

12. Dezember 2018

Bewertung:

Betreuer: Christoph Brein

Version: 1.1

Begonnen: 05. Dezember
2018

Beendet: 12. Dezember 2018

Inhaltsverzeichnis

1	Einführung	3
1.1	Aufgabenstellung	3
1.2	Bewertung	3
2	Aufgabendurchführung	4
2.1	Backup Strategien - Theorie [1]	4
2.1.1	Inkrementelles Backup	5
2.1.2	Differentielles Backup	6
2.1.3	Vollbackup	7
2.2	Backupstrategien in PostgreSQL	8
2.2.1	pg_dump- Vollbackup [2]	8
2.2.2	Vollbackup - Alternative FLSB[3]	11
2.2.3	Continuous Archiving and Point-in-Time Recovery (PITR) - Inkrementel- les Backup [4]	11
2.2.4	Base Backup	12
2.3	Base Backup using Low Level API	12
2.3.1	Recovering using Continous Archiving	13
3	Quellen	14

1 Einführung

In diesem Protokoll werden die verschiedenen Backup-Strategien angesprochen und verglichen. Desweiteren wird geschaut, wie man das ganze in Postgresql umsetzen kann.

1.1 Aufgabenstellung

- Schritt 1):
Hier für arbeitet zunächst die folgende Quelle durch und fasst eure Ergebnisse in einem Protokoll zusammen
<https://www.grundlagen-computer.de/backup/backup-strategien-inkrementell-differentiell-und-vollbackup>

Führt auch an wie ein sinnvoller Restore (Wiederherstellung) der Daten bei den entsprechend durchgeführten Backups vonstattengeht und welche Probleme oder Engpässe sich dabei auf-tun.
- Schritt 2):
Im Anschluss arbeitet in der PostgreSQL Dokumentation das Kapitel "Backup an Restore" durch und ergänzt entsprechend eure Ausarbeitung Backups um die konkreten Umsetzungsmöglichkeiten in PostgreSQL.
- Schritt 3):
Als abschließender Schritt soll nun ein umfassendes Beispiel eurer Dokumentation hinzugefügt werden, in dem Ihr die zuvor erlernten Konzepte praktisch mittels einer Beispieldatenbank umsetzt und erläutert.

1.2 Bewertung

Je mehr desto 1

2 Aufgabendurchführung

2.1 Backup Strategien - Theorie [1]

Es gibt 3 verschiedene Arten Daten zu sichern.

- Inkrementelles Backup
- Differentielles Backup
- Vollbackup

Jede Art hat seine Vor- und Nachteile. Welche das sind werden im folgenden beschrieben.

Welche Art man wählt hängt vom Anwendungsfall ab. Häufig wird auch eine Kombination aus den Möglichkeiten gewählt. (Beispielsweise unter der Woche inkrementelle Backups und am Wochenende Vollbackups.)

2.1.1 Inkrementelles Backup

Beim inkrementellen Backup wird als erster Schritt ein Vollbackup der Daten gemacht. Danach werden immer nur Backups der geänderten Daten nach dem letzten Backup gemacht. Somit sind die Backups sehr klein. Allerdings darf davon keines verloren gehen, da sie aufeinander aufbauen und man für die Wiederherstellung der Daten jedes einzelne Backup benötigt.

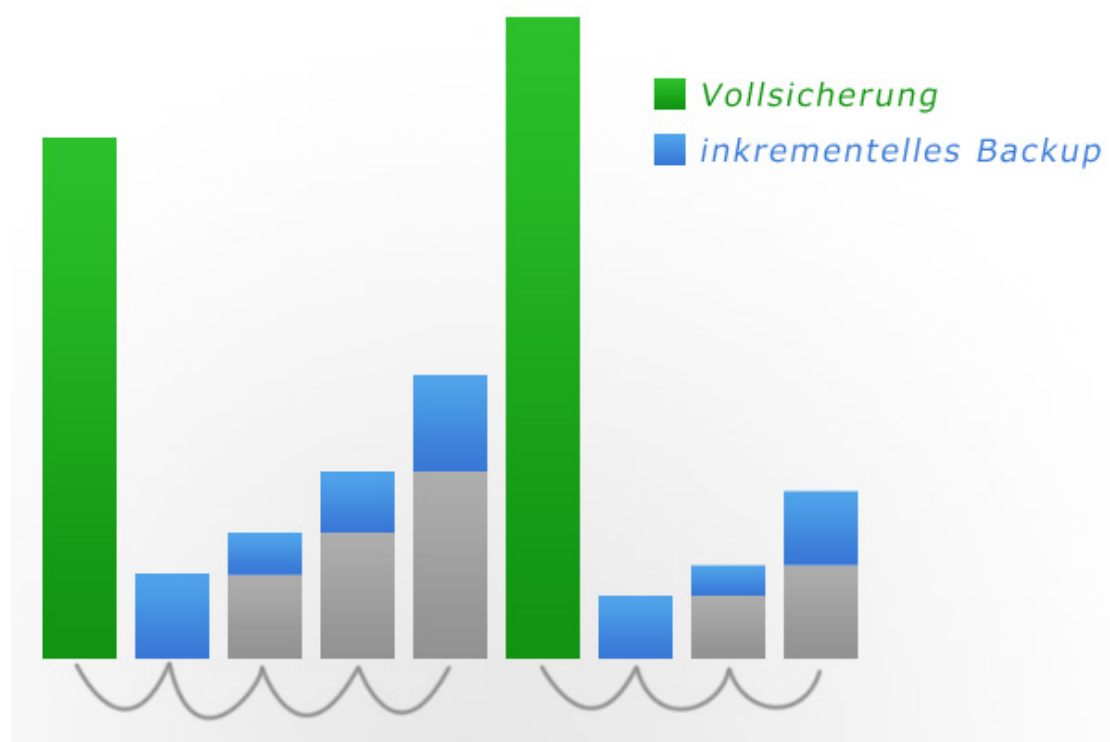


Abbildung 1: Inkrementelles Backup

Vor/Nachteile

- **Vorteile**

- Einfaches Verfahren
- Niedriger Speicherbedarf aufgrund der kleinen inkrementellen Backups
- Wiederherstellung der Daten zu jedem Backupzeitpunkt möglich

- **Nachteile**

- Man braucht alle Backups für die Wiederherstellung (keines darf fehlen)

2.1.2 Differenzielles Backup

Das differenzielle Backup ist dem inkrementellen sehr ähnlich. Auch hier wird als erster Schritt ein Vollbackup der Daten gemacht.

Anders ist aber, dass bei einem Backup nicht die Änderungen zum letzten Backup, sondern die Änderungen zum Vollbackup gespeichert werden.

Somit braucht man für eine Wiederherstellung nicht mehr alle Teil-Backups sondern nur noch das Vollbackup und das Teilbackup des gewünschten Backup Zeitpunkts.

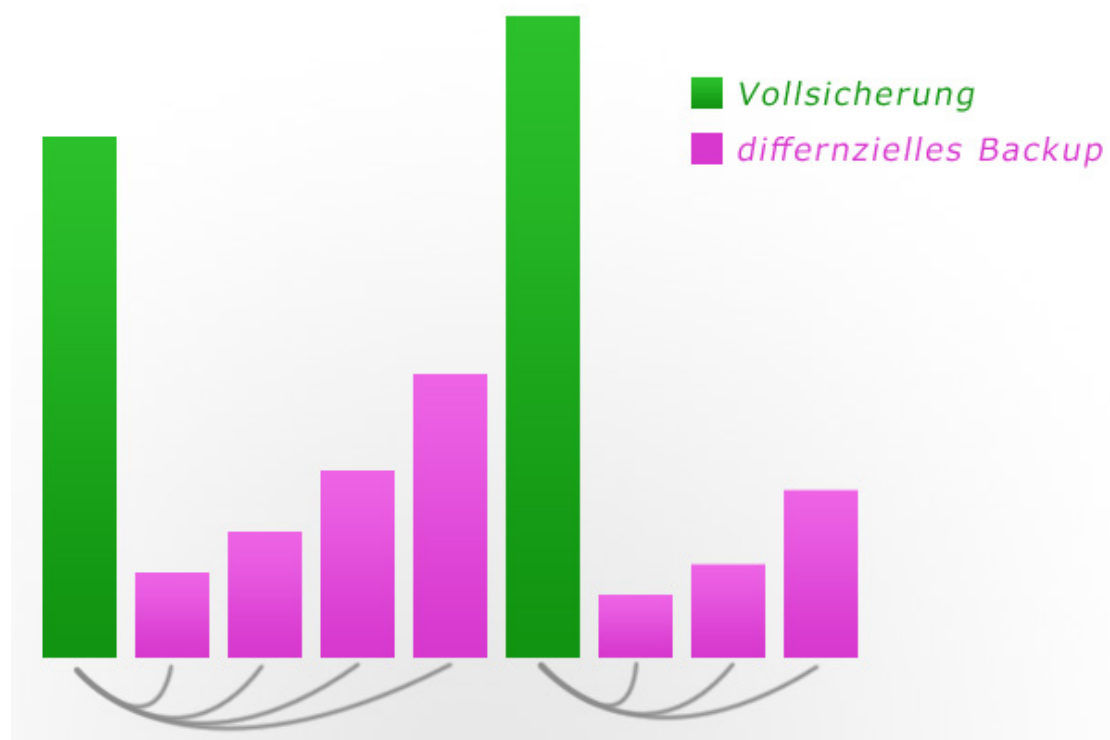


Abbildung 2: Differenzielles Backup

Vor/Nachteile

• Vorteile

- Weniger Speicherbedarf als beim Vollbackup
- Nur Vollbackup und die differenzielle Sicherung zum gewünschten Zeitpunkt notwendig

• Nachteile

- Mehr Speicherbedarf als beim inkrementellen Backup
- Dateien, die einmal verändert werden, müssen bei jedem differenziellen Backup neu gesichert werden. Dadurch steigt der Speicherbedarf

2.1.3 Vollbackup

Hierbei wird der komplette Datenbestand bei jedem Backup gesichert. Um die Daten zurückzusetzen benötigt man demnach nur das Backup zum gewünschten Zeitpunkt.

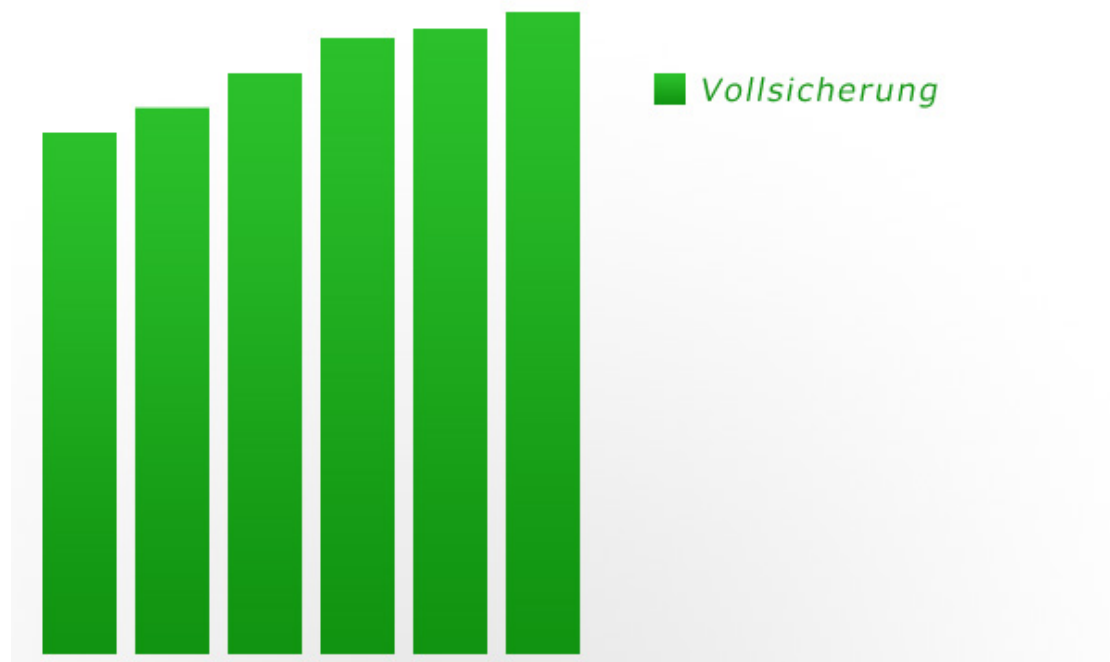


Abbildung 3: Vollbackup

Vor/Nachteile

- **Vorteile**

- Einfache Wiederherstellung

- **Nachteile**

- Sehr hoher Speicherbedarf
- um mehrere Versionen zu haben, müssen mehrere Sicherungsbänder aufbewahrt werden

2.2 Backupstrategien in PostgreSQL

2.2.1 pg_dump- Vollbackup [2]

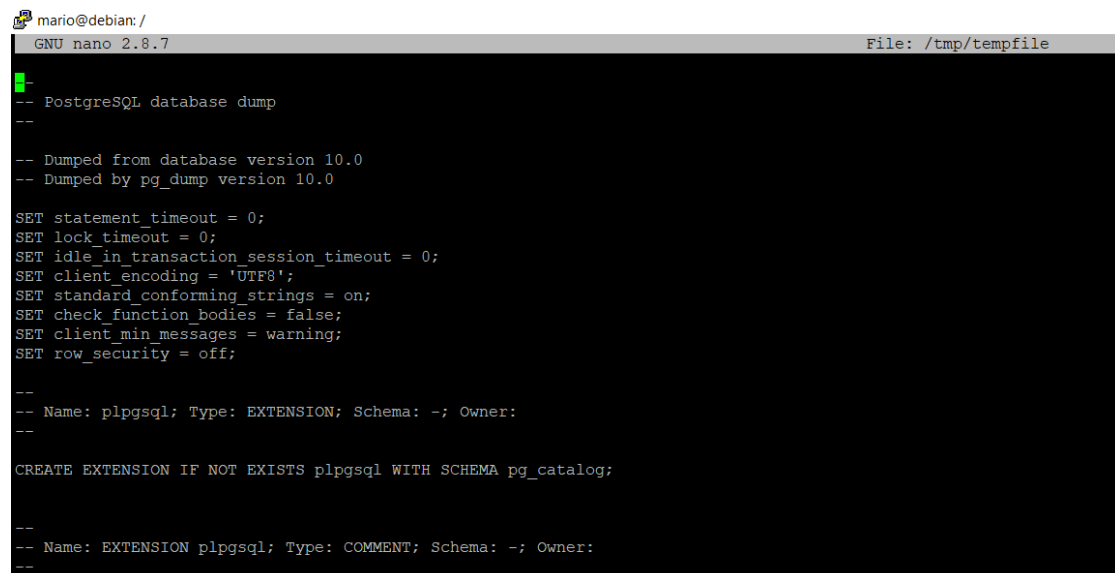
Für das Vollbackup wird die Funktion "pg_dump" verwendet. Dieses erstellt ein SQL File mit allen Commands um die Datenbank zum Zeitpunkt des Commands wiederherzustellen. Dabei muss man allerdings beachten, dass man diesen Command als Superuser ausführen sollte, da nur die Tabellen ausgelesen werden können auf die man auch Leserechte hat.

Es werden hierbei allerdings nicht die Rollen oder Tablespaces gespeichert, da diese eher Clusterweise gespeichert werden.

Zum **speichern** einer einzelnen Datenbank:

```
1 pg_dump dbname > dumpfile
2 pg_dump -h localhost -U mario restaurant > /tmp/tempfile
```

Das Dumpfile ist dann im Ordner **/tmp/tempfile** verfügbar. Es sieht folgendermaßen aus:



```
mario@debian: /
GNU nano 2.8.7 File: /tmp/tempfile
--
-- PostgreSQL database dump
--

-- Dumped from database version 10.0
-- Dumped by pg_dump version 10.0

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;
SET row_security = off;

--
-- Name: plpgsql; Type: EXTENSION; Schema: -; Owner:
--

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;

--
-- Name: EXTENSION plpgsql; Type: COMMENT; Schema: -; Owner:
--
```

Abbildung 4: pg_dump Dumpfile

Als nächster Schritt wird nun die Datenbank mit folgendem Befehl gelöscht und dann überprüft ob sie noch da ist.

```
1 psql
2 DROP DATABASE restaurant;
3 -- List databases
4 \l
```


Die Datenbank existiert nun nicht mehr.

```
postgres=# DROP DATABASE restaurant;
DROP DATABASE
postgres=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
dvdrental	mario	UTF8	en_US.UTF-8	en_US.UTF-8	=Tc/mario +
insytestdb	mario	UTF8	en_US.UTF-8	en_US.UTF-8	=Tc/mario +
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
schokofabrik	mfentler	UTF8	en_US.UTF-8	en_US.UTF-8	=Tc/mfentler +
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +

```
(6 rows)
postgres=#
```

Abbildung 5: Restaurant Datenbank ist nun nicht mehr da.

Diese Datenbank **wiederherstellen**. Man kann die Option "ON_Error_STOP=on" setzen. Diese stoppt die Wiederherstellung falls Fehler auftreten. Das ist nützlich, da man sonst eine fehlerhafte Datenbank haben könnte.

```
1  psql
2  //Datenbank muss natürlich auch wieder erstellt werden
3  CREATE DATABASE restaurant OWNER mario;
4
5  psql --set ON_ERROR_STOP=on dbname < dumpfile
6  psql --set ON_ERROR_STOP=on restaurant < /tmp/tempfile
```

Um zu überprüfen ob die Datenbank jetzt wiederhergestellt wurde kann man einfach ein Select machen.

```
postgres@debian:/$ psql
psql (10.0)
Type "help" for help.

postgres=# \c restaurant
You are now connected to database "restaurant" as user "postgres".
restaurant=# \dt
          List of relations
 Schema |      Name      | Type | Owner
-----+-----+-----+-----
 public | bestellung     | table | postgres
 public | kellner        | table | postgres
 public | rechnung       | table | postgres
 public | speise         | table | postgres
 public | tagesumsatzhelper | table | postgres
(5 rows)

restaurant=# select * from kellner;
 knr |   name
-----+-----
   1 | Kellner1
   2 | Kellner2
   3 | Kellner3
(3 rows)

restaurant=#
```

Abbildung 6: Datenbank wiederhergestellt

Will man eine Datenbank vom **einen auf den anderen Server kopieren** dann kann man das mit Pipes machen.

```
1 pg_dump -h host1 dbname | psql -h host2 dbname
```

Will man ein **Backup vom gesamten Datenbankcluster** mitsamt den **Rollen** und **Tablespaces** erstellen dann benützt man die Funktion "pg_dumpall"

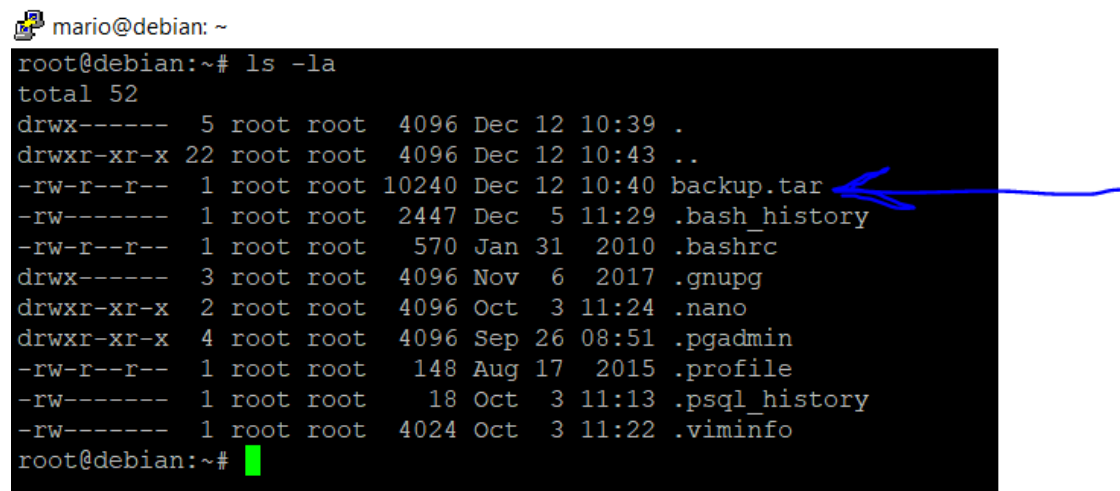
```
1 //Speichern
2 pg_dumpall > dumpfile
3
4 //Wiederherstellen
5 psql -f dumpfile postgres
```

2.2.2 Vollbackup - Alternative FLSB[3]

Eine schnellere **Alternative** zu `pg_dump` ist das **File System Level Backup**. Dabei werden einfach alle Files die Postgres benützt um die Daten zu speichern, kopiert. Die **Nachteile** bei dieser Methode sind, dass 1. der Server abgeschaltet werden muss für das Backup, 2. die Datei größer ist als die beim `pg_dump`, 3. man muss die gesamte DB speichern.

Es werden die Dateien dazu archiviert.

```
1 tar -cf backup.tar /usr/local/pgsql/data
2
3 //restore the database
4 tar -zxf backup.tar -C /usr/local/pgsql/data
```



```
mario@debian: ~
root@debian:~# ls -la
total 52
drwx----- 5 root root 4096 Dec 12 10:39 .
drwxr-xr-x 22 root root 4096 Dec 12 10:43 ..
-rw-r--r-- 1 root root 10240 Dec 12 10:40 backup.tar
-rw----- 1 root root 2447 Dec 5 11:29 .bash_history
-rw-r--r-- 1 root root 570 Jan 31 2010 .bashrc
drwx----- 3 root root 4096 Nov 6 2017 .gnupg
drwxr-xr-x 2 root root 4096 Oct 3 11:24 .nano
drwxr-xr-x 4 root root 4096 Sep 26 08:51 .pgadmin
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
-rw----- 1 root root 18 Oct 3 11:13 .psql_history
-rw----- 1 root root 4024 Oct 3 11:22 .viminfo
root@debian:~#
```

Abbildung 7: Archiviertes Verzeichnis

Anschließend muss man für den **Restore**, diese Dateien nur ersetzen, wie im Codesnippet oben beschrieben.

2.2.3 Continuous Archiving and Point-in-Time Recovery (PITR) - Inkrementelles Backup [4]

Postgres speichert die Änderungen die an den Datenbankfiles gemacht wurden in sogenannten "write ahead logs (WAL)". Diese logs existieren primär für die Wiederherstellung. Aus diesem Grund eignen sie sich perfekt für die nächste Methode.

Für das inkrementelle Backup (mit dem FSLB) wird als erster Schritt ein Vollbackup erstellt und als weitere Schritte werden inkrementell die WAL Files gespeichert. Man sollte im Hinterkopf behalten, dass diese WAL Files von der DB überschrieben werden und man sie deswegen abspeichern sollte.

Um das WAL Archiving zu aktivieren schreib man folgende Line in das **postgresql.conf** File. **%p**

wird durch den Pathname des zu archivierenden Files ersetzt, %f steht für den Filename. Postgres setzt das dann automatisch, darum muss man sich also nicht kümmern außer man möchte den Command manuell in der Console ausführen.

```
1 archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p  
↪ /mnt/server/archivedir/%f' # Unix
```

Die WAL Files werden dann im Directory (/mnt/server/archivedir) gespeichert.

WAL-Archiving kann die Edits in den Files postgresql.conf, pg_hba.conf and pg_ident.conf nicht restoren, da diese manuell gemacht werden und nicht von SQL Commands.

2.2.4 Base Backup

Um BaseBackups zu erstellen wird der Command "pg_basebackup" verwendet.

2.3 Base Backup using Low Level API

Diese Methode ist etwas umfangreicher als die eben genannte, ist aber dennoch sehr simpel. Hierbei ist es wichtig, dass die folgenden Schritte genau in dieser Reihenfolge gemacht werden und immer überprüft wird ob sie auch funktioniert haben.

1. WAL-Archiving muss aktiviert sein und laufen.
2. Mit der Datenbank als **Superuser** verbinden und folgenden Command ausführen. Label steht hierbei für einen beliebigen String, der das Backup identifizieren soll.

```
1 SELECT pg_start_backup('label', true);
```

3. Backup machen (nicht pg_dump, sondern beispielsweise mit tar)
4. Wieder mit der Datenbank als Superuser verbinden und folgenden Command ausführen um wieder aus dem Backupmode zu kommen.

```
1 SELECT pg_stop_backup();
```

2.3.1 Recovering using Continous Archiving

Wenn der Fall eintritt, dass man die Datenbank wiederherstellen muss, dann geht das so:

```
restore_command = 'cp /mnt/server/archivedir/%f %p'
```

1. Stoppe den Server, falls er läuft.
2. Wenn man über genügend Speicherplatz verfügen, kopiere das gesamte Cluster-Datenverzeichnis.
3. Entferne alle vorhandenen Dateien und Unterverzeichnisse unter dem Clusterdatenverzeichnis und unter den Stammverzeichnissen aller verwendeten Tablespace.
4. Stelle die Datenbankdateien von deiner Dateisystemsicherung wieder her. Stelle sicher, dass du mit dem richtigen Besitzer (Datenbankbenutzer, nicht root) und mit den richtigen Berechtigungen wiederhergestellt werden.
5. Entferne alle Dateien in pg_xlog / Diese stammen aus der Dateisystemsicherung und sind daher wahrscheinlich veraltet und nicht aktuell.
6. Wenn du WAL-Segmentdateien, die Sie in Schritt 2 gespeichert haben, nicht archiviert haben, kopiere sie in pg_xlog/.
7. Erstelle eine Wiederherstellungsbefehlsdatei recovery.conf im Clusterdatenverzeichnis.
8. Starte den Server. Der Server wechselt in den Wiederherstellungsmodus und liest die benötigten archivierten WAL-Dateien durch. Nach Abschluss des Wiederherstellungsvorgangs benennt der Server die Datei "recovery.conf" in "recovery.done" um und startet dann die normalen Datenbankvorgänge.

3 Quellen

- [1] <https://www.grundlagen-computer.de/backup/backup-strategien-inkrementell-differentiell-und-vollbackup>
- [2] <https://www.postgresql.org/docs/9.4/backup-dump.html>
- [3] <https://www.postgresql.org/docs/9.4/backup-file.html>
- [4] <https://www.postgresql.org/docs/9.4/continuous-archiving.html>