PostgreSQL Klone mit Reflink-Kopien

(Und was wir dadurch über Backups lernen können!)

Julian Markwort
Senior Database Consultant

pgconf.de 2026

Speaker Introduction

Julian Markwort
Senior Database Consultant

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

- PostgreSQL Consulting
- PostgreSQL Support
- PostgreSQL Remote DBA
- und mehr…

AUSTRIA (HQ)

CYBERTEC POSTGRESQL
INTERNATIONAL (HQ)

SWITZERLAND

CYBERTEC POSTGRESQL
SWITZERLAND

URUGUAY

CYBERTEC POSTGRESQL
SOUTH AMERICA

ESTONIA

CYBERTEC POSTGRESQL
NORDIC

POLAND

CYBERTEC POSTGRESQL
POLAND

INDIA

CYBERTEC POSTGRESQL
INDIA PRIVATE LIMITED

SOUTH AFRICA

CYBERTEC POSTGRESQL
SOUTH AFRICA

Database Services

- 24/7 Support
- High Availability
- Consulting
- Performance Tuning
- Clustering
- Migration
- Compliance

ISO 27001 CERTIFIED

Motivation

Backups sind nützlich.

Motivation

Zusätzliche Backups der Datenbank können genutzt werden:

- für Disaster Recovery
- als Fallback für schief gegangene Wartungen
- als Übungsplatz für neue Features oder Migrationen

Folien

Dieser Foliensatz ist schon hochgeladen!

Übersicht

Es gibt hauptsächlich drei Wege, an eine zusätzliche Kopie zu gelangen:

- Backups
- Snapshots
- Reflink Klone

Backups

- dauern lange, da alles kopiert werden muss
- haben doppelten Speicherbedarf, da alles kopiert werden muss

Snapshots

- sind schnell fertig, da sie nur eine Referenz auf eine Version einer Datei zu dem Zeitpunkt festhalten müssen
- brauchen nur wenig Speicher
  - initial nur Metadaten für Referenzen
  - wenn das Original modifiziert wird, wächst der Speicherbedarf aufgrund von Copy-on-Write
- sind normalerweise atomar, sodass alle Versionen aller Dateien den gleichen Zeitpunkt wiederspiegeln

Reflink Klone

- sind ähnlich wie Snapshots:
  - schnell fertig
  - wenig Speicherbedarf
  - Copy-on-Write
- sind aber *nicht* atomar

Glossar

Reflink Kopie

Eine leichtgewichtige Kopie einer Datei, implementiert durch eine Referenz auf eine bestimmte Version der Datei zum Zeitpunkt des Kopierens. Werden Original oder Kopie modifiziert, so entstehen per Copy-on-Write neue Versionen.

WAL (Write Ahead Log)

Der Name des Transaktionslogs in PostgreSQL.

LSN (Logical Sequence Nummer)

Eine Kombination aus WAL-Datei-Sequenznummer und Offset innerhalb der WAL-Datei, die zur Identifikation jedes Transaktionslog-Eintrags dient.

Backups

Probleme von Backups

Backups sind grundsätzlich inkonsistent.

Wieso?

Weil die Datenbank weiterhin Transaktionen verarbeitet, während die Dateien kopiert werden.

Wenn wir die Kopien Datei für Datei anlegen, so besteht die Möglichkeit, dass die Dateien Zustände nach Abschluss verschiedener Transaktionen darstellen und somit inkonsistent miteinander wären.

Backups sind grundsätzlich inkonsistent

Starte ein Backup.

Table 1:

xmin	xmax	id
41		1

Table 2:

xmin	xmax	id
42		1

WAL

```
TX 41 INSERT TABLE 1
TX 41 COMMIT
TX 42 INSERT TABLE 2
TX 42 COMMIT
Backup Start
```


Backups sind grundsätzlich inkonsistent

Kopiere Tabelle 1.

Table 1:

xmin	xmax	id
41		1

Table 2:

xmin	xmax	id
42		1

WAL

```
TX 41 INSERT TABLE 1
TX 41 COMMIT
TX 42 INSERT TABLE 2
TX 42 COMMIT
Backup Start
```

Copy of Table 1:

xmin	xmax	id
41		1

Backups sind grundsätzlich inkonsistent

Transaktion 43 modifiziert Tabelle 1 (schon kopiert) und Tabelle 2.

Table 1:

xmin	xmax	id
41		1
43		2

Table 2:

xmin	xmax	id
42		1
43		2

WAL

```
TX 41 INSERT TABLE 1
TX 41 COMMIT
TX 42 INSERT TABLE 2
TX 42 COMMIT
Backup Start
TX 43 INSERT TABLE 1
TX 43 INSERT TABLE 2
TX 43 COMMIT
```

Copy of Table 1:

xmin	xmax	id
41		1

Backups sind grundsätzlich inkonsistent

Kopiere Tabelle 2 und stoppe das Backup.

Table 1:

xmin	xmax	id
41		1
43		2

Table 2:

xmin	xmax	id
42		1
43		2

WAL

```
TX 41 INSERT TABLE 1
TX 41 COMMIT
TX 42 INSERT TABLE 2
TX 42 COMMIT
Backup Start
TX 43 INSERT TABLE 1
TX 43 INSERT TABLE 2
TX 43 COMMIT
Backup Stop
```

Copy of Table 1:

xmin	xmax	id
41		1

Copy of Table 2:

xmin	xmax	id
42		1
43		2

von einem Backup starten

- Da wir Datei für Datei kopieren, während der Server läuft, sind die Backups nicht atomar und die Dateien untereinander nicht konsistent.
- Wir müssen eine spezielle *Backup Recovery* durchführen, Wenn wir von einem Backup starten.
  - Recovery läuft mindestens von Start LSN bis Stop LSN
- Das kann eine Weile dauern, wenn die Distanz zwischen den LSN groß ist.
  - Das Anwenden des Transaktionslogs läuft mit höchstens 5-10GB pro Minute.
- Die Datenbank kann erst nach Erreichen der Stop LSN in den *Hot Standby* Modus wechseln und lesende Verbindungen bedienen.

Snapshots

Probleme von Snapshots

- Gibt es eine Garantie, dass die Snapshots wirklich atomar sind?
  - Falls nicht, so bestehen die gleichen Probleme wie bei Backups.
- Manche Snapshot-Mechanismen sind langsam und reduzieren die IO Performance stark.
- Es existiert kein Dateisystem, das ubiquitär ist und Snapshots ermöglicht.
  - Wir wollen etwas, das überall benutzt werden kann.
  - Wir wollen etwas, wo die Verwendung überall gleich ist.

von einem Snapshot starten

- Von einem Snapshot zu starten ist genau wie normale Crash Recovery.
- Die Recovery beginnt bei der Start-LSN des letzten Checkpoints und läuft bis zum Ende vom WAL.

Reflink Klone

Probleme von Reflink Klonen

- Grundsätzlich die gleichen Probleme wie bei Backups.
- Da wir Datei für Datei Reflink-kopieren, sind die Klone nicht atomar und die Dateien untereinander nicht konsistent.

von einem Reflink Klone starten

- Wir müssen eine spezielle *Backup Recovery* durchführen, Wenn wir von einem Klon starten.
  - Recovery läuft mindestens von Start LSN bis Stop LSN
- Wenn wir annehmen, dass das Klonen schnell erfolgt, so wäre nur wenig Transaktionslog anzuwenden.

Ein inkonsistentes Backup konsistent machen

Backup Recovery ist essenziell

- Da Backups grundsätzlich inkonsistent sind, verlassen wir uns auf die *Backup Recovery*, um sie wieder konsistent zu machen.
- Wir müssen die Start LSN und Stop LSN kennen, um sicherzustellen, dass wir *genug* Transaktionslog anwenden.
- Hierfür gibt es in PostgreSQL ein API.

Low Level Backup API

1. Rufe pg_backup_start() auf, und halte die *Verbindung* aufrecht.
   - Wartet auf den nächsten Checkpoint, oder löst direkt einen aus.
   - Gibt uns die Start LSN zurück.
2. Kopiere oder Klone die Dateien.
3. Rufe pg_backup_stop() in der gleichen *Verbindung* auf
   - Gibt uns die Stop LSN des Backups zurück.
   - Gibt uns ein *Backup Label* zurück.
   - Gibt uns eine Tablespace Map zurück.
4. Sichere alles Transaktionslog, das zwischen Start und Stop LSN anfiel.

Transaktionslog sichern

Es gibt drei Möglichkeiten:

1. Erzeuge einen Replikationsslot und zwinge damit die Primary, das WAL aufzuheben, hole es nach dem pg_backup_stop() ab.
2. Starte eine Replikationsverbindung und empfange das WAL während das Backup läuft.
3. Das WAL per `archive_command` von der Primary in ein Archiv schieben lassen und während der *Backup Recovery* per `restore_command` von dort abrufen.

(Backup) Recovery

- Beim Start liest PostgreSQL die `pg_control` Datei um herauszufinden, ob Crash Recovery nötig ist.
- In einem Backup zeigt diese Datei meist nur auf den letzten Checkpoint vor dem Ende des Backups.
  - Wenn wir nur bei diesem Checkpoint mit der Recovery anfangen, so ist es wahrscheinlich, dass wir zu wenig WAL anwenden, was dann wieder Inkonsistenz und Datenkorruption nach sich ziehen kann.
  - PostgreSQL kann dieses Problem beim Start nicht alleine erkennen.
- **Backup Recovery** wird ausgelöst durch das Vorhandensein einer `backup_label` Datei, welche die Start LSN enthält.

Crash Recovery mit pg_control

```
/usr/pgsql-18/bin/pg_controldata -D /tmp/data/

[..]
Database cluster state:               in production
pg_control last modified:             Thu 16 Apr 2026 21:12:58 CEST
Latest checkpoint location:           0/2468B28
Latest checkpoint's REDO location:    0/2468AD0
[..]
```

- Die `REDO location` zeigt die Start-LSN, mit der wir beginnen müssen.
- Die `checkpoint location` zeigt die LSN des Checkpoint Records.
- Erreicht die Recovery diese LSN, so ist das Data-Verzeichnis wieder konsistent.

Backup Recovery mit Backup Label

Eine `backup_label` Datei im Data-Verzeichnis informiert PostgreSQL, wo es mit der Backup Recovery beginnen muss.

```
START WAL LOCATION: 0/4000028 (file 000000010000000000000004)
CHECKPOINT LOCATION: 0/4000080
BACKUP METHOD: streamed
BACKUP FROM: primary
START TIME: 2026-04-16 21:15:53 CEST
LABEL: pg_basebackup base backup
START TIMELINE: 1
```

Das Ende der Backup Recovery findet PostgreSQL dann nach einer Weile im Transaktionslog.

WAL während eines Backup

```
/usr/pgsql-18/bin/pg_waldump /tmp/backup/pg_wal/000000010000000000000004

rmgr: XLOG [..]   lsn: 0/04000028 [..] desc: CHECKPOINT_REDO wal_level replica
[..]
rmgr: XLOG [..]   lsn: 0/04000080 [..] desc: CHECKPOINT_ONLINE redo 0/4000028;
    tli 1; prev tli 1; fpw true; wal_level replica; xid 0:789; oid 24576;
    multi 1; offset 0; oldest xid 744 in DB 1; oldest multi 1 in DB 1;
   oldest/newest commit timestamp xid: 0/0; oldest running xid 789; online
[..]
rmgr: XLOG [..]    lsn: 0/040000F8 [..] desc: BACKUP_END 0/4000028
```


Ein Skript zum Klonen. . .

Vorüberlegungen

Wir wollen:

- `cp --reflink=always pg_data pg_clone`
- das Low-Level-Backup-API benutzen
- das WAL einsammeln
- eine Backup Recovery starten
- nichts kaputt machen

Sicherungsmechanismen für Klone

- Wenn wir das Data-Verzeichnis mit einem Aufruf komplett kopieren, können wir uns nicht aussuchen, dass `pg_control` zuletzt kopiert werden soll.
- Lege im Ziel ein Dummy Backup Label als Platzhalter an, noch bevor das Klonen beginnt, damit ist der Klon fürs erste nicht startbar.
- Entferne niemals ein backup_label aus dem Data-Verzeichnis, wenn PostgreSQL den Start damit abbricht, sonst gibt es keine Garantie mehr.

Pseudocode...initialization

```
original_dir = argv[1]
clone_dir = argv[2]

abort if clone_dir is not empty
abort if original_dir and clone_dir are not on same FS (Reflink copy won't work)
abort if there are any symlinks in original_dir/pg_tblspc

create clone_dir if not exists
create placeholder backup_label in clone_dir
```


Pseudocode. . . start the backup

```
get connection details for PG running in original_dir
open a connection
create replication slot "pg_clone", reserving WAL immediately

start backup, save return value start_lsn

do the Reflink copy
(skipping pg_wal, postmaster.pid, postmaster.opts, pg_control)

ensure standby.signal file exists in clone_dir
```


Pseudocode. . . stop the backup

```
stop backup, save return values stop_lsn, labelfile, spcmapfile

create the backup_label from labelfile contents
create tablespace_map file from spcmapfile contents
```


Pseudocode…receive WAL

```
create empty pg_wal directory

advance replication slot to start_lsn

call pg_receivewal, have it use the slot and stop at stop_lsn
```


Pseudocode...cleanup

```
determine stop_walfile, the one containing stop_lsn
rename {stop_walfile}.partial to stop_walfile

copy pg_control from original_dir to clone_dir

drop replication slot
```


Warnung

- Das Skript ist als experimentell zu betrachten.
  - `pg_basebackup.c` ist 2881 Zeilen lang (und `basebackup.c` 2137)
  - `pg_clone.sh` ist 86 Zeilen lang…
  - `pg_clone.py` ist 234 Zeilen lang…
- Es gibt nicht viele Sicherheitsmechanismen
- Um die Vorgänge zu verstehen und etwas zu lernen, sollte man die Schritte des Skripts testweise mal manuell ausführen, damit man den Zustand zu jedem Zeitpunkt analysieren kann.
- Der Slot könnte nicht aufgeräumt werden, wenn irgendwas schief geht (z.B. PostgreSQL abstürzt)
- Das Skript wird (zumindest ohne weitere Anpassungen) nicht mit Tablespaces funktionieren

Wo ist das Skript?

Muss ich noch hochladen, entweder:

github.com/cybertec-postgresql/pg_clone

oder:

github.com/markwort/pg_clone

Zusammenfassung

Es ist eigentlich gar nicht so schwer, Klone zu erzeugen.

Die Hauptsache ist, dass wir Recovery betreiben, damit der Klon konsistent wird.

Diese Klone ersetzen aber ganz sicher keine ordentlichen Backups und regelmäßige Tests derselbigen.

Danke!

Feedback

- Feedback hilft, Vorträge zu verbessern
- Feedback hilft, die Konferenz noch toller zu machen
- Login in mit postgresql.org account

Diskussion

Tablespaces...

- Im “besten” Fall würde das Skript die Tablespaces komplett “normal” kopieren.
- Würde das Skript einfach die Symlinks übernehmen, so wäre Korruption vorprogrammiert.
- Man müsste jeden Tablespace für sich Klonen (die könnten ja alle auf anderen Mounts liegen) und dann die Symlinks selbst anpassen. Diese Umsetzung möchte ich gerne den Lesenden überlassen.

Wie läuft das mit pg_basebackup?

```
/usr/pgsql-18/bin/pg_basebackup -D /tmp/backup -c fast -X stream -R -v -P

pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/7000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created temporary replication slot "pg_basebackup_505900"
39934/39934 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 0/7000120
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
```


Wie läuft das in `src/backend/backup/basebackup.c` ?

```
/* In the main tar, include the backup_label first... */
backup_label = build_backup_content(backup_state, false);
sendFileWithContent(sink, BACKUP_LABEL_FILE,
                    backup_label, -1, &manifest);

/* Then the bulk of the files... */
sendDir(sink, ".", 1, false, state.tablespaces,
        sendtblspclinks, &manifest, InvalidOid, ib);

/* ... and pg_control after everything else. */
sendFile(sink, XLOG_CONTROL_FILE, XLOG_CONTROL_FILE, &statbuf,
            false, InvalidOid, InvalidOid,
            InvalidRelFileNumber, 0, &manifest, 0, NULL, 0);
```

Aus `perform_base_backup()` @ `ea94d2e6734` (mit einigen Auslassungen).

Snapshots...

1. Manche Virtualisierungslösungen bieten Snapshots an, müssen aber zu diesem Zwecke dann auf ein Journal umstellen, dass die Performanz leiden lässt.
2. ZFS ist toll, kann ich aber keinem Kunden aus der Enterprise Linux Welt andrehen (Lizenzproblematik, Kernel patchen...).
3. BTRFS ist nix für Server.
4. LVM Snapshots erscheinen mir nicht sehr praktikabel.
5. Alle Methoden, die Snapshots auf einer hardwarenäheren Ebene als dem Dateisystem erstellen, ermöglichen es nicht, auf dem Snapshot direkt eine Datenbank zu starten.
6. In ZFS geht sowas: `zfs clone pool/data@snapshot pool/data_clone`.

Reflink Kopien. . .

1. Verfügbar in XFS, ZFS, BTRFS, . . .
2. Sind in der Anwendung trivial.
3. Brauchen keine Interaktion mit anderen Komponenten.
4. Brauchen keine Vorbereitung
5. Funktionieren also immer, solange das Dateisystem Reflinks unterstützt.
6. Aufräumen ist auch ganz einfach.