Replikation mit PostgreSQL 9.2

CLT 2013 – 16./17.03.2013

Andreas 'akretschmer' Kretschmer, Andreas 'ads' Scherbaum

Web: http://a-kretschmer.de
http://andreas.scherbaum.la/ / http://andreas.scherbaum.biz/

16./17.03.2013

Was ist PostgreSQL?

- Relationale Datenbank unter BSD Lizenz
- Weltweit aktive Community
- Zahlreiche Features und Funktionen (Foreign Keys, Transaktionen, Trigger)
- Läuft auf zahlreichen Betriebssystemen und diverser Hardware
- Weitgehendes Einhalten der SQL-Standards – Dokumentation der Abweichungen
- Im Schnitt pro Jahr ein Minor-Release mit neuen Features
- Version 9.3 wird derzeit fleißig entwickelt

Der Redner

- Name: Andreas Kretschmer
- Angestellt bei Internet24 (Dresden)
- Arbeit mit PostgreSQL seit über 10 Jahren
- Aktiv in der deutschsprachigen PostgreSQL-Community

Der Redner

- Name: Andreas Scherbaum
- Selbstständig im Bereich Datenbanken, Linux auf Kleingeräten, Entwicklung von Webanwendungen
- Arbeit mit Datenbanken seit 1997, mit PostgreSQL seit 1998 oder 1999
- Gründungsmitglied der Deutschen und der Europäischen PostgreSQL User Group
- Board of Directors – European PostgreSQL User Group
- Seit 2011 mit "Big Data" bei EMC/Greenplum beschäftigt

Ziele dieses Workshops

- Installation von PostgreSQL 9.2.3 (automatisiert)
- Konfiguration der Master Datenbank
- Erstellen eines Base Backups der Master Datenbank
- Einspielen des Base Backups als Slave (2 Slaves)
- Konfiguration der Slave Datenbank und der Replikation
- Starten des Slave
- Testen der Replikation

Datenbank kompilieren

- Unter folgender URL gibt es ein Paket zum Workshop
- http://andreas.scherbaum.la/writings/clt-2013-skripts.tar.bz2

Datenbank kompilieren

- Auspacken mit:

Beispiel (Entpacken)

```
cat clt-2013-skripts.tar.bz2 | bunzip2 | tar x
cd build/
```

Beispiel (Kompilieren)

```
make build
```

Datenbank kompilieren

- Im Verzeichnis `install` wird PostgreSQL 9.2.3 installiert
- Im Verzeichnis `master/data` und `slave1/data` sowie `slave2/data` wird jeweils eine Datenbank initialisiert
- Beide Datenbanken bleiben weitgehend unkonfiguriert
- Nur Änderungen für die Replikation werden vorgenommen

Datenbank initialisieren

Beispiel (Alles erst mal löschen)

```
make clean-slave clean-master
```

Beispiel (Master initialisieren)

```
make init-master
```

Streaming Replication + Hot Standby

- PostgreSQL 9.0 enthält Streaming Replication
- PostgreSQL 9.0 enthält Hot Standby

- PostgreSQL 9.1 und 9.2 brachten Verbesserungen
- PostgreSQL 9.1 brachte synchrone Replikation
- PostgreSQL 9.2 brachte kaskadierende Replikation

Streaming Replication – der Master

- Archive Logs (16 MB) werden wie gehabt zum Slave übertragen (nicht auf dem Master lagern)
- Zusätzlich pollt der Slave aktuelle Änderungen vom Master
- Bei zu großem Lag werden (zuerst) die Logfiles eingespielt (`wal_keep_segments`)

- Im Master wird weiterhin der `archive_mode` aktiviert
- `archive_command` kopiert die Logfiles
- Die Anzahl Clients über `max_wal_senders = n` konfiguriert
- `wal_level` muss auf `archive` oder besser `hot_standby` gesetzt werden

Datenbank konfigurieren – Master

- Datei `master/data/postgresql.conf` im Editor öffnen
- Folgende Änderungen vornehmen:

Beispiel (Einstellungen im Master)

```
listen_addresses = '*'
port = 5440
archive_mode = on
archive_command = '/bin/cp %p ../../wal/%f'
archive_timeout = 300
wal_level = hot_standby
max_wal_senders = 3
```


Streaming Replication – der Master – pg_hba.conf

- Benötigt einen Account mit "replication" Rechten

Beispiel (Superuser Zugang für Replikation – `pg_hba.conf`)

```
# TYPE   DATABASE      USER        CIDRADDRESS     METHOD
local    replication   postgres                    trust
host     replication   postgres    127.0.0.1/32    trust
host     replication   all         0.0.0.0/0       reject
```

- Hinweis: als erste Zeilen eintragen
- Hinweis: ggf. eigenen User für Replikation erstellen
- Hinweis: statt "postgres" ggf. anderen Nutzer (eigenen Unix Account) einsetzen

Datenbank starten – Master

- Den Master starten
- Hinweis: eigene Shell/Konsole/Terminal verwenden

Beispiel (Datenbank starten)

```
make master-db
```

Beispiel (Datenbankshell öffnen)

```
make master-sh
```


Datenbank verwenden

- Einige schreibende Änderungen durchführen
- Z. B. eine Tabelle erstellen und mit Daten befüllen
- Danach die WAL-Datei wechseln:

Beispiel (neue WAL-Datei beginnen)

```
SELECT pg_switch_xlog();
```

- Im Verzeichnis `wal` sollten sich nun Dateien befinden

Streaming Replication – der Slave

- `hot_standby = on`
- Archivierung deaktivieren (`archive_mode`, `wal_level`, `max_wal_senders`)
- Ggf. Port ändern (wenn Master und Slave auf der gleichen Maschine laufen)

Hot Standby

- Der Slave kann für (lesende) Anfragen genutzt werden
- Transaktionen auf dem Slave sind komplett, aber ggf. einige Zeit hinter dem Master
- Einige Lockingmodi können verwendet werden, andere nicht
- Two-Phase Commits sind nicht möglich (erfordern einen WAL-Eintrag)
- Nicht möglich: `LISTEN`, `NOTIFY`, `UNLISTEN`
- Möglich: Backup

Hot Standby – Konfliktlösungen

- Der Slave wartet `max_standby_archive_delay` bzw. `max_standby_streaming_delay` Sekunden bei einem Konflikt (Default: 30 Sekunden)
- Transaktionen und Abfragen, die einen Lock halten, werden beendet
- Idle Transaktionen werden beendet
- Konflikte durch Aufräumarbeiten (Entfernen veralteter Records auf dem Master) informieren die Anfrage, diese beendet sich selbst (ansonsten: falsche Ergebnisse)

Hot Standby – Failover

- Im Failover Fall (Slave beendet Recovery) bleiben alle Verbindungen bestehen
- Transaktionen werden schreibbar

Slave erstellen

- Der Slave wird wie bei PITR mit einem Online-Backup aufgesetzt
- Der Master wird mittels `pg_start_backup()` und `pg_stop_backup()` in den Online-Backup Modus versetzt
- Seit PostgreSQL 9.1: `pg_basebackup`
- Der Slave darf noch nicht laufen!
- Im Workshop Paket befindet sich ein entsprechendes Makefile Ziel

Beispiel (Slave erstellen)

```
make backup
```


Datenbank konfigurieren – Slave

- Datei `slave1/data/postgresql.conf` im Editor öffnen
- Folgende Änderungen zurücksetzen bzw. vornehmen:

Beispiel (Slave konfigurieren)

```
port = 5441
archive_mode = off
#archive_command = ''
#archive_timeout = 30
```

Beispiel (Slave konfigurieren)

```
hot_standby = on
```

Streaming Replication – der Slave

- Normale Konfiguration in `recovery.conf`
- Zusätzlich: `standby_mode = 'on'`
- Zusätzlich: `primary_conninfo = 'host=masterdb port=5432'`
- Zusätzlich: `trigger_file = '/tmp/trigger.hs'`

Streaming Replication – der Slave – Konfiguration

- `slave1/data/recovery.conf` im Editor erstellen

Beispiel (Beispielkonfiguration)

```
restore_command = 'cp ../../wal/%f %p'
standby_mode = 'on'
primary_conninfo = 'host=127.0.0.1 port=5440 user=postgres'
trigger_file = '/tmp/replikation.trigger'
```

- Hinweis: statt "postgres" ggf. anderen Nutzer (eigenen Unix-Account) einsetzen

Datenbank starten – Slave

- Den Slave starten
- Hinweis: eigene Shell/Konsole/Terminal verwenden

Beispiel (Datenbank starten)

```
make slave-db1
make slave-db2
```

Beispiel (Datenbankshell öffnen)

```
make slave-sh1
make slave-sh2
```


Datenbank verwenden

- Änderungen auf dem Master vornehmen
- Änderungen sofort danach auf dem Slave betrachten
- Schreibende Änderungen auf dem Slave versuchen

Failover – Activate a slave

- Wenn das Triggerfile auftaucht, wird der Slave "ausgekoppelt"

Beispiel (Triggerfile erzeugen)

```
touch /tmp/trigger.slave2
```


Failover – Activate a slave

- Der vorherige "Slave 2" ist jetzt eine eigenständige Datenbank
- Hinweis: "Split-Brain" Szenarien sind unbedingt zu vermeiden!

Hot Standby – Zustand der Replikation

- `pg_is_in_recovery()`: liefert true auf einem Standby-System zurück

Beispiel (pg_is_in_recovery())

```
postgres=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 Zeile)
```


Hot Standby – aktueller WAL-Record (Master)

- `pg_current_xlog_location()`: liefert den aktuellen WAL-Record auf dem Master

Beispiel (pg_current_xlog_location())

```
postgres=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
------------------------------
 0/DD000000
(1 Zeile)
```

Hot Standby – empfangener WAL-Record

- `pg_last_xlog_receive_location()`: liefert den letzten empfangenen WAL-Record

Beispiel (pg_last_xlog_receive_location())

```
postgres=# SELECT pg_last_xlog_receive_location();
 pg_last_xlog_receive_location
-------------------------------
 0/DD000000
(1 Zeile)
```


Hot Standby – eingespielter WAL-Record

- `pg_last_xlog_replay_location()`: liefert den letzten eingespielten WAL-Record

Beispiel (pg_last_xlog_replay_location())

```
postgres=# SELECT pg_last_xlog_replay_location();
 pg_last_xlog_replay_location
------------------------------
 0/DD000000
(1 Zeile)
```

Hot Standby – Status View

- `pg_stat_replication`: liefert eine Übersicht über alle Slaves

Beispiel (pg_stat_replication)

```
postgres=# SELECT * FROM pg_stat_replication;
pid  | usesysid | usename  | application_name | ...
...
```


Kaskadierende Slaves

- Ein Slave kann gleichzeitig Daten zu anderen Slaves streamen
- Konfiguration wie ein Master
- Aktuell nur asynchron
- Jeder Slave verbindet sich zu genau einem anderen Server (direkt zum Master, oder zu einem anderen Slave)
- Wird ein Slave aktiviert unterbricht das die Replikation zu seinen Slaves (Timeline ändert sich)

Synchrone Replikation

- Ein Slave kann synchron Daten vom Master empfangen
- Änderungen werden auf dem Slave geschrieben bevor der Master das Commit an die Applikation gibt
- Kann per Transaktion aktiviert werden
- Fällt der Slave aus wartet der Master ewig!

Synchrone Replikation

- Konfiguration: `application_name` muss gesetzt sein (in `recovery.conf`)
- `synchronous_standby_names` spezifiziert eine Liste mit Slaves
- `synchronous_commit` wird auf `remote_write` gesetzt

Beispiel (recovery.conf)

```
primary_conninfo = 'host=127.0.0.1 port=5440
user=postgres application_name=slave1'
```

Synchrone Replikation

- Slave stoppen, Master stoppen, Synchrone Replikation aktivieren, Master starten

Beispiel (alles stoppen)

```
make stop-all
```

- im Editor öffnen: `master/data/postgresql.conf`
- Nach "Aktivieren für synchrone Replikation" suchen

Beispiel (Master starten)

```
make master-db
```


Synchrone Replikation

- Transaktion im Master starten
- Etwas ändern
- Transaktion committen

Beispiel (Transaktion)

```
BEGIN;
CREATE TABLE z();
COMMIT;
```

Synchrone Replikation

- Slave starten

Beispiel (Transaktion)

```
make slave-db1
```


Nacharbeiten

- `max_locks_per_transaction` erhöhen, wenn `Prepared Transactions` verwendet werden
- `max_connections`, `max_prepared_transactions` und `max_locks_per_transaction` müssen auf dem Slave gleich oder größer als auf dem Master sein
- `vacuum_defer_cleanup_age` verzögert `VACUUM` auf dem Master (um die angegebene Anzahl Transaktionen) und erhöht damit die Laufzeit für Anfragen auf dem Slave (alternativ: Anfrage rückwärts zum Master)
- `hot_standby_feedback` informiert den Master über laufende Anfragen auf den Slaves

Wichtiger Termin

Wichtiger Termin
`pgconf.de 2013`

- 8. November
- in Oberhausen
- gefolgt von der ORR

Webseite: `http://www.pgconf.de/`

Wichtiger Termin

Wichtiger Termin
`pgconf.eu 2013`

- wahrscheinlich Ende September oder Anfang Oktober
- in einem europäischen Land

Webseite: `http://www.pgconf.eu/`

Ende

`http://www.a-kretschmer.de/`

`http://andreas.scherbaum.la/`

Fragen?

Andreas 'akretschmer' Kretschmer <???@???.???>
Andreas 'ads' Scherbaum <andreas@scherbaum.biz>
PostgreSQL User Group Germany
European PostgreSQL User Group