Teil 3: Backup und Recovery – die Daten der PostgreSQL sichern

Sicher ist sicher

Unsere Daten sind unser Kapital. Deshalb ist es essenziell wichtig, dafür zu sorgen, dass sie nicht verloren gehen. Die PostgreSQL bietet von Haus aus gute Möglichkeiten, um einem Datenverlust entgegenzutreten. Lernen Sie hier die Grundlagen, um für Ihre Anforderungen eine passende Backupstrategie zu entwickeln.

von Andreas Wenk

Ah – Sie sind wieder da. Schön, dass Sie auch den dritten Teil der Artikelserie zu PostgreSQL [1] lesen. Aber auch wenn dies der erste Artikel ist, den Sie aus der Serie lesen – kein Problem. Das Thema ist genauso spannend, wie das der anderen Artikel. Es geht um Backup, zu Deutsch: Datensicherung. In diesem Artikel werde ich die On-Board-Möglichkeiten der PostgreSQL beleuchten und Ihnen zeigen, wie Sie Ihre Daten sichern. Allerdings ist es wichtig, sich (wie bei allen Vorhaben in der IT-Welt) vorab Gedanken zu machen, welches die beste Strategie ist. Die Fragen, die sich dabei stellen, betreffen den Grund für ein Backup, die Häufigkeit der Backups und den Ort, an dem die Daten gesichert werden sollen. Die Programme, die ich Ihnen dann im weiteren Verlauf des Artikels zeigen werde, sind *pg_dump*, *pg_dumpall* und *pg_restore*.

Warum ein Backup durchführen?

Was für eine Frage, denken Sie? Kennen Sie die unterschiedlichen Gründe für die Wichtigkeit eines Backups? Sehen wir

uns das etwas genauer an. Ich war mir lange Zeit nicht bewusst darüber, welche Teile eines Rechners (Servers) am ehesten kaputt gehen. Vor allem nicht, wie häufig. Seitdem ich allerdings bei einem ISP arbeite und oft bei den Sysadmins vorbeischaue, ist mir klar, dass eines der anfälligsten Hardwareteile die HD (Hard Disc – Festplatte) ist. Der Vollständigkeit halber müssen hier noch das Netzteil und der Ventilator genannt werden. Eigentlich ist das auch sehr gut nachvollziehbar, weil auf eine HD mechanisch zugegriffen wird. Verschleiß ist vorprogrammiert. Und da wir nun einmal die Daten unserer Datenbank auf einer HD vorhalten, ist das ein Problem, dem wir entgegentreten müssen. Dann darf der menschliche Faktor auf keinen Fall vergessen werden. Was machen Sie, wenn ein Kollege versehentlich ein DROP DATABASE abfeuert … ups! Natürlich sollte das nicht passieren, aber es kann nun einmal der Fall sein. Ein weiterer Grund, ein Backup anzulegen. Schließlich haben wir da noch die Software selbst, die wir nutzen, um auf unsere Daten zuzugreifen. Zum einen ist das das Betriebssystem, die Clientsoftware oder PostgreSQL selbst. Software hat Bugs. Und Bugs können im hier besprochenen Szenario zu Datenverlust führen. Also noch ein Grund für Backups.

In welchem Intervall sollen Backups angelegt werden?

Eine wichtige Frage, die nicht so einfach zu beantworten ist. Natürlich könnten wir einfach sagen, so oft wie möglich. Allerdings stellt sich die Frage, wie oft es denn möglich ist. Ein Backup ist ein zusätzlicher Job, der auf dem Server ausgeführt wird. Das bedeutet, es werden Ressourcen benötigt, die anderen Prozessen nicht zur Verfügung stehen. Insofern muss hier geklärt werden, wie oft ein Backup überhaupt erstellt werden kann, ohne dass die Performance der Applikation in Mitleidenschaft gezogen wird. Außerdem muss auch die Frage gestellt werden, wie oft ein Backup überhaupt Sinn macht.

Bei der PostgreSQL wird durch die zugrunde liegende MVCC-(Multi-Version-Concurrency-Control-)Architektur sichergestellt, dass die Daten immer in einem konsistenten Zustand vorliegen, es also keine kaputten Datenstände bzw. Transaktionen gibt. Daher kann man davon ausgehen, dass immer die Datenstände verloren gehen werden, die seit dem letzten Backup geschrieben (oder gelöscht) wurden. Eine kaputte Datenbank muss nicht befürchtet werden. Es wird folglich eine Entscheidung getroffen, wie alt die Daten sein dürfen, wenn ein Backup wieder eingespielt wird (mehrere Stunden oder nur Minuten).

An dieser Stelle sei der Hinweis erlaubt, dass die vorgestellten Backupmethoden eventuell nicht die richtigen für Ihre Anforderungen sind, sondern nur einen Teil einer Backupstrategie ausmachen. Gegebenenfalls muss auch mit WAL-Archivierung, Replikation oder mit einem Stand-by-System gearbeitet werden. Allerdings sind das keine klassischen Backupmethoden (außer der WAL-Archivierung) und werden deshalb nicht weiter besprochen. Eine kurze Begriffserklärung können Sie trotzdem im Kasten „Datensicherheit ohne Backup“ nachlesen.

Wohin soll das Backup geschoben werden?

Diese Frage ist eventuell die wichtigste überhaupt beim Erstellen einer Backupstrategie. Prinzipiell spielt der finanzielle Aspekt sicher eine große Rolle.

> Software hat Bugs. Und Bugs können in diesem Szenario zu Datenverlust führen.

Eventuell lässt Ihr Budget nicht zu, einen weiteren Server als Backupmaschine anzuschaffen. Es macht allerdings wenig Sinn, ein Backup auf dem gleichen Server abzulegen, auf dem Ihr Datenbankcluster läuft – womöglich noch auf der gleichen Festplatte. Damit produzieren Sie einen Single Point of Failure, den Sie auf alle Fälle vermeiden sollten. Gehen wir kurz die Möglichkeiten durch.

Um irgendeine Art von Sicherheit zu haben, sollten Sie zumindest ein RAID-System [5] auf Ihrem Server installieren. Dabei sollte es sich nicht um ein Software-, sondern um ein Hardware-RAID handeln. Dies besteht zumindest aus zwei HDs, wobei beide HDs parallel laufen und über den gleichen Datenbe-

Datensicherheit ohne Backup

WAL-Archivierung

WAL bedeutet „Write Ahead Log“ und ist das Log, das alle Informationen über Transaktionen der Datenbank beinhaltet. Dieses Log wird in regelmäßigen Abständen gelöscht, da nur die aktuellen Transaktionen benötigt werden. Zuvor werden die Transaktionen natürlich in den Datenbestand übernommen. Da das WAL immer einen konsistenten Zustand der Datenbank zu einem gewissen Zeitpunkt enthält, können Sie daraus auch ein Backup erstellen. Allerdings muss auf jeden Fall ein initiales Backup der Datenbank erstellt werden. Bei der Wiederherstellung würde man aus diesem initialen Backup die Datenbank wiederherstellen und dann das WAL-Backup sozusagen „oben drauf“ packen.

Replikation

Bei der Replikation wird durch eine Replikationssoftware wie Slony-I [2] eine Master-multiple-Slaves-Infrastruktur angelegt. Sobald die Datenstände auf dem Master Node (Knoten) geändert werden, sorgt die Software dafür, dass diese Änderungen auf die einzelnen Slaves repliziert werden. Sinnigerweise befinden sich die Slaves auf physikalisch anderen Servern. Die PostgreSQL selbst bietet keine Replikation, sondern es gibt nur Produkte von Drittanbietern (wie Slony).

Stand-by-System

Ein Stand-by-System kann auch als Failover-System bezeichnet werden. Wenn also der Hauptserver wegbricht, übernehmen ein (oder mehrere) Server, die im Stand-by-Status sind. Es gibt verschiedene Arten wie *Hot Standby* oder *Warm Standby* (auch als *Logshipping* bezeichnet). Zu letzterer Art erhalten Sie mehr Informationen unter [3] in der Onlinedokumentation der PostgreSQL. Die Version 9 der PostgreSQL wird übrigens auch ein Hot-Stand-by-System haben. Lesen Sie mehr dazu in der Dokumentation unter [4].

stand verfügen. Sollte nun eine HD kaputtgehen, können eine (oder mehrere) andere HDs im RAID-System übernehmen. Sprich, ein geschriebenes Backup ist damit immer noch nutzbar, wobei ein RAID kein Backup ersetzt. Besser ist auf alle Fälle das Schreiben des Backups auf einen zweiten Server. Wenn wir davon ausgehen, dass Sie über zwei Server verfügen, z. B. einen Webserver und Datenbankserver, besteht die Möglichkeit, das Backup vom Datenbankserver auf den Webserver zu schreiben. Damit haben Sie zumindest den Single Point of Failure umgangen. Diese Variante wird gar nicht so selten genutzt. Das Backup würde dabei lokal erstellt werden und dann z. B. per rsync [6] oder scp [7] auf den anderen Server geschoben werden.

Die luxuriöseste Variante ist ähnlich der eben beschriebenen, allerdings wird das Backup von einem Backupsystem wie Bacula [8] abgeholt. Das Backup wird lokal geschrieben und an einen festgelegten Ort auf dem Server geschrieben – z. B. */var/backup/postgresql*. Das Backupsystem kommt dann zu gegebener Zeit vorbei, holt das Backup ab und transportiert es auf einen externen Backupserver. Von diesem Backupserver kann das Backupsystem dann eine Wiederherstellungsroutine starten.

Weitere Optionen für die CLI-Programme

Es gibt noch weitere PostgreSQL-CLI-Programme. Geben Sie einfach mal *pg_*, gefolgt von TAB TAB auf der Shell ein, um eine Übersicht zu erhalten. Diese Programme nehmen immer folgende Optionen an:

-p port: gibt an, auf welchem Port die zu sichernde DB läuft

-U username: der Benutzername, mit dem Sie sich an der DB anmelden

-h hostname: gibt an, auf welchem Host die Datenbank läuft

Für *pg_dump* können Sie außerdem diese beiden Optionen nutzen:

-t tabellenname: Möglichkeit, nur eine einzige Tabelle zu sichern

-a: es sollen nur die Daten der Datenbank gesichert werden

Eine einfache Backupstrategie

Das folgende Szenario stellt eine denkbar einfache Backupstrategie dar, hilft aber auf alle Fälle größere Schmerzen, verursacht durch Datenverlust, zu vermeiden. Mit dem Programm *pg_dump* erstellen wir zwei tägliche Backups von jeder einzelnen Datenbank in unserem Cluster. Wenn wir davon ausgehen, dass tagsüber am meisten los ist, macht es Sinn, früh morgens um 6 Uhr und gegen Abend um 18 Uhr ein lokales Backup zu erstellen. Erfahrungsgemäß sind die Hauptbesuchszeiten einer Website gegen Mittag und nach dem Abendessen. 18 Uhr ist dabei theoretisch auch relativ stark frequentiert. Sie sollten beobachten, ob die Performance zu sehr leidet und den Zeitpunkt ggf. verschieben. Des Weiteren bitten wir unsere Sysadmins freundlich, unsere erstellten Backups einmal am Tag um ca. 19 Uhr abzuholen. Damit werden beide Backups mitgenommen und außerdem bleibt bis dahin genug Zeit, das Backup um 18 Uhr zu beenden. Natürlich besteht die Gefahr, dass irgendwann zwischendurch etwas kaputtgeht – aber das ist nun einmal das Risiko, mit dem wir leben müssen.

LISTING 1

```
root:~# psql -U postgres
psql (8.4.2)
Type "help" for help.
CREATE DATABASE phpmag_0410;
CREATE DATABASE
postgres=# \c phpmag_0410
psql (8.4.2)
You are now connected to database
                                    "phpmag_0410".
phpmag_0410=# CREATE TABLE bob (id serial,
                                    bauarbeit text);
NOTICE:  CREATE TABLE will create implicit sequence
          "bob_id_seq" for serial \  column "bob.id"
CREATE TABLE
phpmag_0410=# CREATE TABLE zakk (id serial,
                                    guitars text);
NOTICE:  CREATE TABLE will create implicit sequence
          "zakk_id_seq" for serial \  column "zakk.id"
CREATE TABLE
phpmag_0410=# INSERT INTO bob (bauarbeit)
    VALUES ('schiefer Turm von Pisa'), \  ('Eifelturm'),
                                    ('chinesische Mauer');
INSERT 0 3
phpmag_0410=# INSERT INTO zakk (guitars)
                        VALUES ('Pinnball Les Paul'), \
 ('Cofin Fiddle'), ('Gibson Accoustic');
INSERT 0 3
```

Außerdem erstellen wir einmal in der Woche ein Backup des gesamten Datenbankclusters. Der Hintergrund ist folgender: Wie Sie wissen, gibt es in der PostgreSQL globale Datenbankobjekte, z. B. Rollen und deren Privilegien. Diese werden bei einem Backup mit *pg_dump* nicht mitgesichert. Wenn Sie ein Backup einer einzelnen Datenbank also wiederherstellen wollen, geschieht das immer in eine bestehende Datenbank. Jetzt kommt *pg_dumpall* ins Spiel. Damit erstellen Sie ein Backup des gesamten Datenbankclusters. Die Schritte zum Wiederherstellen eines kaputten Clusters ergeben sich dann von selbst. Zuerst wird das globale Backup wiederhergestellt und danach die Backups der einzelnen Datenbanken. Da aller Wahrscheinlichkeit nach Änderungen an globalen Einstellungen des Clusters wesentlich seltener als normale Operationen vorkommen, haben wir entschieden, dass ein globales Backup einmal in der Woche ausreicht. Ach ja – den Sysadmins geben wir dann auch wieder Bescheid, dieses Backup abzuholen und zu sichern. Was würden wir nur ohne die Jungs tun?

Praxis – ein Backup erstellen

In diesem Abschnitt zeige ich Ihnen nun, wie Sie mit den Programmen *pg_dump* und *pg_dumpall* ein Backup erstellen. Und weil es so essenziell wichtig ist, zu wissen, das Wichtigste zuerst: Sie können mit beiden Programmen ein Backup während des laufenden Betriebs vornehmen. Das ist doch mal was! Um die Beispiele mit durchspielen zu können, sollten Sie nun als Erstes eine Datenbank erstellen und eins, zwei Tabellen einfügen. Nutzen Sie dafür das wunderbare Programm *psql* (Listing 1).

Für unsere Testzwecke sollte das genügen. Verlassen Sie nun *psql* (*\q*) und kehren Sie zurück auf die Shell. Um es etwas einfacher zu haben, wechseln Sie nun mit *su postgres* zum Systembenutzer *postgres* und springen automatisch in sein Home-Verzeichnis.

Erstellen wir nun ein Backup: *postgres:~$ pg_dump phpmag_0410 > phpmag_0410.sql*. Sehr einfach. Das Standardverhalten von *pg_dump* ist das

NEU! Jetzt mit
Nehalem-Prozessoren

serverloft

Sie haben die Wahl:
AMD oder Intel

NEU: Nehalem-Prozessoren
Höhere Leistung
Effiziente Energienutzung
Schnelle SSD-Festplatten

intel

Kein Setup!
149,–
Euro gespart!

Willkommen bei serverloft

serverloft steht für professionelle Root-Server ohne Kompromisse. Wir richten uns an Anwender, die Wert auf das Wesentliche legen: Performante Hardware mit Server-Prozessoren von AMD oder Intel und PRIMERGY Servern von FUJITSU, ein schnelles, stabiles Netzwerk und transparente Angebote ohne Mindest-laufzeit, monatlicher Abrechnung und einem außergewöhnlich guten Preis-Leistungsverhältnis.

Sie erhalten vollen Root-Zugriff und verwalten Ihren Server selbst. Natürlich können Sie Ihren Server jederzeit per Webinterface resetten, neu installieren oder im Rescue-Modus starten.
Ihr Server ist innerhalb von 24 Stunden eingerichtet und sollte es einmal zu einem Hardware-Schaden kommen, wechseln wir den Server innerhalb von vier Stunden aus – rund um die Uhr.

Gehen Sie keine Kompromisse ein.
Ihr

Thomas Strohe
Thomas Strohe, Geschäftsführer

Powered by plusserver

NEU: PerfectServer XL – Xeon 2.0
Fujitsu PRIMERGY RX200
NEU: 2x Xeon E5520, 2x Quadcore-Nehalem
NEU: 8 GB DDR3-Arbeitsspeicher ECC
2x 500 GB SATA II-Festplatten, 7.2 k
10.000 GB Datentransfer inklusive

intel

Keine Mindestlaufzeit!
Keine Einrichtungsgebühr. Preis pro Monat: **159,–**

PerfectServer L – Opteron 1.0
Fujitsu PRIMERGY RX330
1x AMD Opteron 2344 HE, 1x Quad-Core
4 GB DDR2-Arbeitsspeicher ECC
2x 250 GB SATA II-Festplatten, 7.2 kk
5.000 GB Datentransfer inklusive

AMD

Keine Mindestlaufzeit!
Keine Einrichtungsgebühr. Preis pro Monat: **79,–**

Bei allen Angeboten haben Sie freie Wahl zwischen openSuSE 11.1, Ubuntu 8.04 LTS, Debian 4.0 & 5.0 LAMP, CentOS 5 oder Windows Webserver 2008 (zzgl. 30 EUR/Monat). Zur Administration ist Plesk 8.x oder 9.x „10 Domains" auf Wunsch kostenlos erhältlich.

Keine Einrichtungsgebühr bis 31.5.2010. Jedes zusätzliche 1 GB Datentransfer über Frei-kontingent EUR 0,19. Alle Preise in Euro inklusive 19 % MwSt.

Tel. 0800 100 4082

www.serverloft.de

Erstellen einer Datei im reinen Textformat. Das Gleiche erreichen Sie durch die Verwendung der Option *-Fp* (F = Format, p = plain). Nun bietet *pg_dump* allerdings auch noch die Nutzung anderer Formate. Zum einen ist das das *tar*-Archivformat. Beim Wiederherstellen mit *pg_restore* können Sie das Inhaltsverzeichnis dieses Archivformats ausgeben lassen und ggf. Einträge markieren, die nicht wiederhergestellt werden sollen, oder Sie können die Reihenfolge der Einträge ändern. Mehr dazu im Abschnitt zu *pg_restore*. Der Aufruf ist analog: *postgres:~$ pg_dump -Ft phpmag_0410 > phpmag_0410.tar*. Und dann gibt es noch das PostgreSQL-interne Format, das als custom-Format bezeichnet wird. Die Option ist hierbei *-Fc*. Es handelt sich hierbei um ein binäres Format und hat den Vorteil, dass es wesentlich schneller als die anderen Backups in den anderen beiden Formaten wiederhergestellt werden kann: *postgres:~$ pg_dump -Fc phpmag_0410 > phpmag_0410.bak*. Auffällig sind hierbei die unterschiedlichen Dateigrößen der *.bak*-, *.sql*- und *.tar*-Dateien. Um das erheblich größere *tar*-Archiv noch zu verkleinern, hilft *gzip*:

```
postgres:~$ pg_dump -Ft phpmag_0410 >
phpmag_0410-2.tar | gzip > \ phpmag_0410.tar.gz
```

Oha: von 11 264 Bytes auf 20 Bytes (in diesem Beispiel) – das ist wirklich viel kleiner. Sie haben hier also gesehen, wie ein Backup mit *pg_dump* erstellt wird. Werfen Sie nun noch kurz einen Blick auf den Kasten „Weitere Optionen für die CLI-Programme“. Diese Optionen sind gerade beim Einsatz von mehreren Clustern parallel sehr hilfreich. Welche weiteren Optionen es gibt, lesen Sie in der Onlinedokumentation [9].

LISTING 2

```
postgres:~$ pg_dumpall -g > cluster8.4.2-global

Sehen wir uns den Inhalt dieser Datei an:
postgres:~$ less cluster8.4.2-global
--
-- PostgreSQL database cluster dump
--
\connect postgres
SET client_encoding = 'UTF8';
SET standard_conforming_strings = off;
SET escape_string_warning = off;
--
-- Roles
--
CREATE ROLE kontor;
ALTER ROLE kontor WITH NOSUPERUSER INHERIT
    NOCREATEROLE NOCREATEDB LOGIN \  PASSWORD
      'md598697195c57236c6ba1b71e35eec6a99';
CREATE ROLE odie;
ALTER ROLE odie WITH NOSUPERUSER INHERIT
                  NOCREATEROLE NOCREATEDB LOGIN \
 PASSWORD 'md55e0ed1227e218a882850cf20
                                      c8f96e49';
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT
                        CREATEROLE CREATEDB LOGIN;
--
-- PostgreSQL database cluster dump complete
–
```

Weitere Optionen für pg_dumpall

-r: nur Rollen sichern

-s: nur Schemata sichern

-t: nur Tablespaces sichern

-a: nur Daten speichern, keine Schemata

Beachten Sie, dass Sie die Optionen nur einzeln nutzen können. So etwas wie *-rt* funktioniert nicht. Rufen Sie *pg_dumpall -help* auf, um alle Optionen zu sehen.

LISTING 3

```
postgres:~$ dropdb phpmag_0410
postgres:~$ createdb phpmag_0410
postgres:~$ psql phpmag_0410 < phpmag_0410.sql
SET
[...]
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
 setval
--------
      3
(1 row)
[...]
REVOKE
REVOKE
GRANT
GRANT
```

Wie oben erwähnt, wird mit *pg_dumpall* ein Backup des gesamten Datenbankclusters erstellt. Und so funktioniert das: *postgres:~$ pg_dumpall -U postgres > cluster8.4.2*

Damit haben wir einen Datenbank-Dump von allen vorhandenen Datenbanken erstellt. Rein technisch erstellt das Programm nacheinander von jeder einzelnen Datenbank und von allen globalen Objekten einen Dump. Folgen wir nun unserer beschriebenen Backupstrategie, brauchen wir allerdings die einzelnen Datenbanken in diesem Backup gar nicht zu sichern. Es genügt, wenn wir die globalen Objekte wie Rollen und Privilegien, Tablespaces und Schemata sichern. Die Option *-g* tut genau das (Listing 2).

Das ist natürlich nur eine Beispieldatei von meinem Testcluster. Der Dump enthält die globalen Einstellungen für das Encoding des Clusters und ein paar Rollen. Das würde uns also für das Backupszenario völlig ausreichen. Übrigens, diese Datei hat gerade einmal eine Größe von 897 Byte, im Gegensatz zum kompletten Dump des Clusters, der eine Größe von (hier) 24 MB hat. Das sollte berücksichtigt werden im Hinblick auf die Dauer eines solchen Backups und auch des Transfers des Backups auf eine andere Maschine. Und es gibt noch einige weitere, hilfreiche Optionen für das Programm *pg_dumpall*. Drei aus meiner Sicht wichtige Optionen sind im Kasten „Weitere Optionen für pg_dumpall“ oder in der Onlinedokumentation unter [10] zu finden.

In diesem Abschnitt haben Sie gelernt, wie einfach es ist, ein Backup von einer Datei oder vom gesamten Cluster zu erstellen. Im nächsten Abschnitt zeige ich Ihnen, wie Sie die Daten wieder in einen Datenbankcluster zurückspielen können.

Wenn's geknallt hat – Daten wiederherstellen

Wir bekommen eine Nachricht (per Nagios, per E-Mail, per Telefonat), dass sich die HD eines unserer Datenbankserver verabschiedet hat. Leider war kein RAID-System vorhanden, deshalb sind die lokalen Daten alle futsch. Also tappern wir mit einer neuen HD unterm Arm ins RZ (Rechenzentrum) und tau-

schen die Platte aus. Zurück im Büro müssen wir jetzt zusehen, dass die Daten aus unserem Backup wieder auf die Festplatte kommen. Also führen wir einen so genannten Recovery-Prozess aus. Kann uns ja nix passieren, wir haben ein Backup. Well done!

Sehen wir uns im Folgenden an, wie ein Backup wieder eingespielt wird. Im eben beschriebenen Fall finden wir ein System vor, in dem es noch keinen Datenbankcluster gibt. Das heißt, zuerst muss ein Datenbankcluster erstellt werden. Das kann auf verschiedenen Wegen geschehen – siehe hierzu den ersten Teil dieser Artikelserie. Beachtet werden sollte, dass es die gleiche Version der PostgreSQL ist (also 8.4 und nicht 8.3 oder 9.x). Sollte dem nicht so sein, aus welchen Gründen auch immer, wird es etwas komplizierter, denn eine Migration ist nicht so trivial. Das näher zu beschreiben, würde allerdings den Rahmen dieses Artikels sprengen. Es sei nur gesagt, dass die Daten aus dem Backup auf jeden Fall in einen Cluster mit der gleichen Version gespielt werden müssen, bevor zu einer höheren Version migriert werden kann. Nachdem der Cluster erstellt ist, macht es Sinn, zuerst den Dump mit den globalen Objekten einzuspielen: *postgres:~$ psql -d postgres < cluster8.4.2-global*. Die Ausgabe zeigt Ihnen, was gerade alles geschehen ist. Und dass es funktioniert, ist insofern klar, als die Datenbank *postgres* immer in einem PostgreSQL-Datenbankcluster existiert. Ziemlich einfach. Die Alternative dazu wäre, einen kompletten Datenbank-Dump, der nicht nur die globalen Objekte, sondern auch alle Datenbanken enthält, einzuspielen. Aber Vorsicht: Sie werden mit Sicherheit nicht zusätzlich und nicht in der gleichen Häufigkeit solch ein Backup fahren, wie Sie es von den einzelnen Datenbanken tun. Das wäre viel zu aufwändig (gerade im Hinblick auf große bis sehr große Datenmengen im GB-Bereich – und nein, 30 GB für eine Datenbank ist nicht sehr groß, sondern nur groß). Außerdem würde die Möglichkeit, nur einzelne Datenbanken wiederherzustellen, fehlen. Ich rate Ihnen deshalb davon ab.

Als Nächstes zeige ich Ihnen, wie Sie einen einzelnen Datenbank-Dump einspielen. Dabei eine Sache gleich vorweg: In diesem Szenario haben wir einen jungfräulichen Datenbank-Cluster. Das wird in der Regel nicht so sein, sondern der Fall, dass das System noch lebt und die Daten aufgrund von „menschlichem Versagen“ kaputt sind, ist eher anzunehmen. Das heißt, es befinden sich noch Daten, Functions, Sequences, Indices usw. in der Datenbank. Wenn Sie nun Daten aus dem Backup darüber buttern, wird es zu etlichen Fehlermeldungen kommen wie *"ERROR: relation "bob" already exists"*. Deshalb behelfen Sie sich eines einfachen Tricks: Sie löschen die Datenbank komplett und erstellen mit dem Programm *createdb* eine leere neue. Ups?

LISTING 4

```
postgres@debian:~$ pg_restore -d phpmag_0410
                                    phpmag_0410.bak
--
-- PostgreSQL database dump
--
SET statement_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = off;
SET check_function_bodies = false;
SET client_min_messages = warning;
SET escape_string_warning = off;
SET search_path = public, pg_catalog;
[...]
--
-- Name: bob; Type: TABLE; Schema: public; Owner:
                                    postgres; Tablespace:
--
CREATE TABLE bob (
  id integer NOT NULL,
  bauarbeit text
);
ALTER TABLE public.bob OWNER TO postgres;
--
-- Name: bob_id_seq; Type: SEQUENCE; Schema: public;
                                    Owner: postgres
--
CREATE SEQUENCE bob_id_seq
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO MINVALUE
  CACHE 1;
[...]
--
-- Data for Name: bob; Type: TABLE DATA;
                        Schema: public; Owner: postgres
--
COPY bob (id, bauarbeit) FROM stdin;
1       schiefer Turm von Pisa
2       Eifelturm
3       chinesische Mauer
\.
[...]
--
-- PostgreSQL database dump complete
--
```

LISTING 5

```
postgres:~$ pg_restore -l phpmag_0410.tar >
                                    phpmag_0410.toc
postgres:~$ cat
;
; Archive created at Wed Mar 31 01:01:20 2010
;   dbname: phpmag_0410
;   TOC Entries: 18
;   Compression: 0
;   Dump Version: 1.11-0
;   Format: TAR
;   Integer: 4 bytes
;   Offset: 8 bytes
;   Dumped from database version: 8.4.2
;   Dumped by pg_dump version: 8.4.2
;
;
; Selected TOC Entries:
;
3; 2615 2200 SCHEMA - public postgres
1784; 0 0 COMMENT - SCHEMA public postgres
1785; 0 0 ACL - public postgres
1497; 1259 73839 TABLE public bob postgres
1496; 1259 73837 SEQUENCE public bob_id_seq
                                    postgres
1786; 0 0 SEQUENCE OWNED BY public bob_id_seq
                                    postgres
1787; 0 0 SEQUENCE SET public bob_id_seq postgres
1499; 1259 73848 TABLE public zakk postgres
1498; 1259 73846 SEQUENCE public zakk_id_seq
                                    postgres
1788; 0 0 SEQUENCE OWNED BY public zakk_id_seq
                                    postgres
1789; 0 0 SEQUENCE SET public zakk_id_seq postgres
1777; 2604 73842 DEFAULT public id postgres
1778; 2604 73851 DEFAULT public id postgres
1779; 0 73839 TABLE DATA public bob postgres
1780; 0 73848 TABLE DATA public zakk postgres
```

Ja! Das ist auch die offizielle Empfehlung der PostgreSQL-Community. Die einzelnen Schritte sehen Sie in Listing 3.

Das sieht gut aus. Wir haben die Datenbank erfolgreich aus dem Backup wiederhergestellt. An dieser Stelle sei erwähnt, dass es für *pg_dump* die Optionen *--clean* und *--create* gibt, die obige Schritte vereinfachen können. Informativ auch ein Blick in die Manual-Seiten via *man pg_dump*. Natürlich ist an dieser Stelle zu empfehlen, alle Datenbanken mit einem Skript automatisiert wieder einzuspielen. Die Erstellung überlasse ich Ihnen als Übung. Das geht natürlich auch mit einem PHP-CLI-Programm. Vielleicht wird das irgendwann ein weiterer kleiner Artikel meinerseits.

Ein Backup mit pg_restore wieder einspielen

pg_restore ist ein weiteres Programm aus der Reihe Backup und Recovery. Mithilfe dieses Programms haben Sie die Möglichkeit, einen Datenbank-Dump aus einem Archivformat wieder einzuspielen. Einen Dump im Plain-Text-Format können Sie allerdings nicht einspielen. Der einfachste Fall ist also, mit *pg_restore* den Dump *phpmag_0410.bak* oder *phpmag_0410.tar* wieder einzuspielen. Den Aufruf und die Ausgabe sehen Sie in Listing 4.

Das hat einwandfrei geklappt. Allerdings bietet *pg_restore* noch weitere nette Features. Unter anderem ist das die Möglichkeit, einen Dump im Archivformat *.tar* zu verändern. Dabei können Sie Einträge hinzufügen (was nicht wirklich Sinn macht), ändern oder löschen. Um das tun zu können, wird aus dem Dump eine TOC-Datei generiert. Dabei steht TOC für *Table of Contents*. Das Erstellen ist trivial und wird einfach durch das Setzen der Option *-l* erledigt. Den Aufruf und die Ausgabe sehen Sie in Listing 5.

pg_restore erlaubt uns, den Datenbank-Dump aus einem Archivformat wieder einzuspielen.

Sie haben bestimmt bereits erkannt, dass das Semikolon einen Kommentar einleitet. Am Anfang der Datei werden einige Informationen über die Datei angegeben. Danach erfolgen alle im Archiv enthaltenen Elemente jeweils auf einer Zeile. Die Nummer am Anfang jeder Zeile ist eine Archiv-ID. Um nun das Backup nicht zu zerstören, bietet es sich an, nicht benötigte Inhalte einfach auszukommentieren, indem Sie an den Anfang der Zeile ein Semikolon schreiben. Natürlich besteht hier auch die Möglichkeit, die Reihenfolge der Inhalte zu ändern, denn die PostgreSQL arbeitet diese Liste von oben nach unten ab. In einigen Fällen ist es nicht möglich, die Datenbank zu löschen, sondern das Backup muss in die bestehende Datenbank eingespielt werden. Dabei kann es vorkommen, dass die Reihenfolge geändert werden muss, damit keine Fehler entstehen. Und das können Sie hier sehr einfach erledigen. Beachten Sie aber auch, dass dies hier nur eine sehr kleine Datei ist. Sie werden eher weniger Lust haben, eine solche TOC-Datei mit mehreren tausend Zeilen zu bearbeiten, zumindest nicht von Hand.

Nachdem Sie die Datei bearbeitet haben, spielen Sie die Daten aus dem Backup unter Verwendung dieser TOC-Datei in die Datenbank ein. Sie weisen *pg_restore* sozusagen an, die Objekte im Backup auf Grundlage und in der Reihenfolge, wie in der in der TOC-Datei angegeben, auszuführen. Die Option für *pg_restore* ist *-L* gefolgt vom Pfad zur TOC-Datei: *postgres@debian:~$ pg_restore -L phpmag_0410.toc -d phpmag_0410 phpmag_0410.bak*

Ein Backup mit pgAdmin III erstellen

Ich habe Ihnen bislang die Programme auf der Command Line gezeigt. PgAdmin III, das Standard-Verwaltungsprogramm für die PostgreSQL, bietet allerdings auch die Möglichkeit, Backups zu erstellen und wieder einzuspielen. Um ein globales Backup zu erstellen oder die globalen Objekte zu sichern, markieren Sie im Object Browser den gewünschten Server und wählen aus dem Toolsmenü *Backup globals ...* bzw. *Backup server ...* Beachten Sie, dass beide Arten von Backups nur als Plain text gesichert werden können.

Um ein Backup einer einzelnen Datenbank zu erstellen, wählen Sie diese im Object Browser aus und öffnen dann wiederum über den Navigationspunkt *Tools* den Eintrag Backup. Hier haben Sie die Möglichkeit, zwischen den Formaten binär (COMPRESS), Archiv (TAR) oder reinem Text (PLAIN) zu wählen. Beim Format PLAIN haben Sie zusätzlich noch die Möglichkeit, weitere Optionen anzugeben, z. B. nur die Daten (Only Data) zu speichern. Wählen Sie dann einen Speicherort aus, klicken Sie auf OK, und das Backup wird erstellt. Das Einspie-

LISTING 6

```
#!/bin/sh
# Andreas Wenk 03.04.2010
# pg_backup.sh
# Script um einen Dump von jeder Datenbank im Cluster
# herzustellen.
# Benutzung:
# ./postgresql_backup.sh
#
# Variablen
DATE='/bin/date "+%F-%T"'
BACKUPDIR="/var/backups/postgres/"
FILEFORMAT=".bak"
SERVER='hostname'
PORT = 5432

# welche Datenbanken sind im Cluster vorhanden?
psql -t -c "SELECT datname FROM
                    pg_catalog.pg_database WHERE
(datname NOT LIKE 'template%' AND datname !=
                                'postgres');" |
while read i; do
 if [ ! -z $i ]; then
 # das Backup erstellen oder mit einer Fehlermeldung
  # abbrechen
 pg_dump -p "$PORT" -Fc $i >
              "$BACKUPDIR$DATE-$i$FILEFORMAT" ||
 echo "error: backup $i not successfull." >&2
 fi; done

# Backup Dateien löschen, die älter als 7 Tage sind
find $BACKUPDIR -ctime +7 -exec rm {};
```

CREATE OR DIE

Das neue Portal für Pixelschubser und Codiernasen!

Inspired by Patricia Waller

www.createordie.de

len des Backups erfolgt ziemlich analog. Öffnen Sie über den Navigationspunkt Tools den Bereich Restore, wählen Sie eine Backupdatei aus und spielen Sie sie wieder ein.

Skript, um automatische Backups erstellen zu lassen

Weiter oben habe ich über eine einfache Backupstrategie gesprochen. Ich möchte Ihnen als Abschluss dieses Artikels nun noch ein relativ einfaches Shell Script vorstellen, das Sie mit dem Cron Daemon als Cronjob regelmäßig laufen lassen können. Dieses Skript können Sie natürlich auch in PHP schreiben. Allerdings ist es üblich, solche Skripte als Shell Script zu schreiben, sodass eine größtmögliche Portabilität auf Unix-artigen Systemen gewährleistet ist.

Das Skript erstellt von allen vorhandenen Datenbanken im Cluster ein Backup, allerdings nicht von der Datenbank *postgres* und von allen *template*-Datenbanken. Um herauszufinden, welche Datenbanken das sind, wird eine SQL-Query auf die Systemtabelle *pg_database* im Schema *pg_catalog* abgesetzt. Über eine Pipe wird dann auf die Ergebnismenge zugegriffen und in einer *while*-Schleife jeweils das Backup erstellt. Da wir außerdem nicht wollen, dass wir zu alte Backups aufbewahren, werden Backups, die älter als sieben Tage sind, durch die letzte Zeile in diesem Skript gelöscht. In Listing 6 finden Sie den gesamten Quellcode.

Um das Skript regelmäßig ausführen zu lassen, fügen Sie der Crontab folgenden Eintrag hinzu:

```
# backup postgresql
20 5 * * *    postgres cd /var/lib/postgresql/ && ./
                        postgresql_backup.sh
```

Damit wird jeden Tag um 05:20 Uhr das Skript *postgresql_backup.sh* aus dem Verzeichnis */var/lib/postgresql* ausgeführt. Um nun auch noch die globalen Objekte mit *pg_dumpall* zu sichern, reicht sogar ein einfacher Aufruf von *pg_dumpall* direkt in der Crontab: *40 5 * * * pg_dumpall > /var/backup/pg_backup-$(date +%F-%T).bak*. Um 05:40 Uhr an jedem Tag wird das Programm *pg_dumpall* aufgerufen, um eine Backupdatei mit dem Namen *pg_backup-2010-04-02-05:40:00.bak* zu erstellen. Somit haben Sie auch diesen Job erledigt.

Der Schlussakkord in Moll

Wir sind am Ende dieser Artikelserie angelangt. Sie haben aus meiner Sicht nun die wichtigsten Bereiche kennen gelernt, um mit der PostgreSQL-Datenbank arbeiten zu können. Im ersten Teil haben Sie gesehen, dass die grundlegende Installation eines PostgreSQL-Clusters nicht kompliziert ist. Im zweiten Teil habe ich Ihnen dann gezeigt, wie Sie der PostgreSQL in Sachen Performance unter die Arme greifen können und wie Sie Flaschenhälse in Ihren SQL-Queries finden. Dieser Teil rundet schließlich die Serie mit dem Wissen um Backups ab. Was mir bleibt, ist Ihnen viel Spaß mit der PostgreSQL zu wünschen.

An dieser Stelle möchte ich mich sehr herzlich bei Andreas Putzo bedanken. Andreas ist ein sehr guter und konstruktiver Kritiker meiner Artikel (und auch des Buchs). Sein Rat war extrem hilfreich beim Schreiben dieser Artikelserie. Danke, Andreas!

Wenn Sie weiter in das Thema PostgreSQL einsteigen möchten, lege ich Ihnen nochmals die drei aktuellsten Bücher für Ihre weitere Lektüre ans Herz: „PostgreSQL 8.4 – Installation, Grundlagen, Praxis“ von Thomas Pfeiffer und mir [11], „PostgreSQL. Datenbankpraxis für Anwender, Administratoren und Entwickler“ von Andreas Scherbaum [12] und „PostgreSQL-Administration“ von Peter Eisentraut und Bernd Helmle [13].

PostgreSQL ist cool! Diese Datenbank bietet Ihnen ein professionelles Werkzeug, um Ihre Daten sicher und konsistent zu speichern und performant abzurufen. Dabei greifen Sie auf ein relationales Datenbankmanagementsystem zu. Ist das immer das Richtige? Nein. Ich möchte Sie dazu ermutigen, über den Tellerrand zu blicken und sich zu überlegen, ob nicht eventuell eine andere Datenbankarchitektur für Ihre Anforderungen die richtig ist. Gerade für Webapplikationen, in denen viele monolithische Datensätze geschrieben werden, bietet sich eine dokumentbasierte Datenbank wie CouchDB [14] oder MongoDB [15] an. „Always use the right tool for the job!“. Werfen Sie doch mal einen Blick darauf!

Ist die PostgreSQL immer die richtige Wahl?

„Use the right tool for the right job!“

Links & Literatur

[1] http://www.postgresql.org
[2] http://www.slony.info/
[3] http://goo.gl/ygX5
[4] http://goo.gl/K6qK
[5] http://goo.gl/MhBg
[6] http://goo.gl/3Ng0
[7] http://goo.gl/jn9f
[8] http://goo.gl/cWql
[9] http://goo.gl/Yamu
[10] http://goo.gl/8fd0
[11] Pfeiffer, Thomas; Wenk, Andreas: „PostgreSQL 8.4 – Installation, Grundlagen, Praxis“, Galileo Computing, 2010
[12] Scherbaum, Andreas: „PostgreSQL – Datenbankpraxis für Anwender, Administratoren und Entwickler“, Open Source Press, 2009
[13] Eisentraut, Peter; Helmle, Bernd: „PostgreSQL-Administration“, O'Reilly, 2008
[14] http://couchdb.apache.org/
[15] http://www.mongodb.org/

Andreas Wenk

Andreas Wenk ist PostgreSQL-Fan und Softwareengineer bei SinnerSchrader in Hamburg. Außerdem lernt er täglich Neues über Programmierung, Internet, seine beiden Töchter und andere coole Dinge. Sie erreichen ihn unter andy@nms.de und können ihm unter @awenkhh folgen.