МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №3
по дисциплине
«Распределённые системы хранения данных»

Вариант №55893

Выполнил:
Студент группы P3334
Баянов Равиль
Динарович
Преподаватель:
Николаев Владимир
Вячеславович

Санкт-Петербург
2025

Содержание

Задание

Цель работы - настроить процедуру периодического резервного копирования базы данных, сконфигурированной в ходе выполнения лабораторной работы №2, а также разработать и отладить сценарии восстановления в случае сбоев.
Узел из предыдущей лабораторной работы используется в качестве основного. Новый узел используется в качестве резервного. Учётные данные для подключения к новому узлу выдаёт преподаватель. В сценариях восстановления необходимо использовать копию данных, полученную на первом этапе данной лабораторной работы.

Номер выделенного узла: pg112
Логин: postgres1
Пароль: kx9JiBg7

Описание этапов выполнения

Этап 1. Резервное копирование

Задание:

- Настроить резервное копирование с основного узла на резервный следующим образом:

  Периодические полные копии с помощью SQL Dump.
  По расписанию (cron) раз в сутки, методом SQL Dump с сжатием. Созданные архивы должны сразу перемещаться на резервный хост, они не должны храниться на основной системе. Срок хранения архивов на резервной системе - 4 недели. По истечении срока хранения, старые архивы должны автоматически уничтожаться.

- Подсчитать, каков будет объем резервных копий спустя месяц работы системы, исходя из следующих условий:
  - Средний объем новых данных в БД за сутки: 550МБ.
  - Средний объем измененных данных за сутки: 100МБ.
- Проанализировать результаты.

Выполнение:

Создадим скрипт ***dump_script.sh*** для резервного копирования, который будет запускаться по чёткому расписанию с помощью ***cron***.

Но для начала сгенерируем ssh ключи для передачи копий на резервный узел без ввода пароля при срабатывании скрипта.

Сгенерируем RSA ключи:

```
ssh-keygen -t rsa -b 4096
```

И перекинем на резервный узел приватный ключ:

```
ssh-copy-id postgres1@pg112
```

Скрипт dump_script.sh:

```
# Новая переменная окружения директории с дампами на основном узле
```

```
export LOCAL_DUMP_DIR=$HOME/dumps/postgres

# Новая переменная окружения директории с дампами на резервном узле
export REMOTE_DUMP_DIR=/var/db/dumps/postgres

# Создаём директорию для хранения дампов на основном узле
mkdir -p $LOCAL_DUMP_DIR

# Создам директорию для хранения дампов на резервном узле
ssh postgres1@pg112 "mkdir -p $REMOTE_DUMP_DIR"

# Выполнение логического резервного кодирования для всех баз данных на
сервере с помощью SQLDump
pg_dumpall -p 9745 -U postgres0 --exclude-database=template1 | gzip >
$LOCAL_DUMP_DIR/dump_$(date +%Y-%m-%d).sql.gz

# Проверка, что ЛК прошло успешно
if [ $? -eq 0 ]; then
 echo "Резервное копирование успешно завершено на основном узле:
$LOCAL_DUMP_DIR"

# Копирование только что созданных дампов на резервный узел
 scp -r $LOCAL_DUMP_DIR/* postgres1@pg112:$REMOTE_DUMP_DIR/

 if [ $? -eq 0 ]; then
   echo "Резервная копия успешно перенесена на резервный узел:
$REMOTE_DUMP_DIR"

# Удаление дампов после компирования на резервный узел, так как на основном
узле они  не должны храниться
  find $LOCAL_DUMP_DIR/* -delete

# Удаление дампов с резервного узла по истечению 4 недель
ssh postgres1@pg112 "find $REMOTE_DUMP_DIR -type f -name '*.gz' -mtime +28 -
exec rm -f {} \;"

   echo "Старые резервные копии удалены на резервном узле"
 else
   echo "Ошибка при переносе резервной копии на резервный узел"
   exit 1
 fi
else
 echo "Ошибка при выполнении резервного копирования на основном узле"
 exit 1

fi
```

Затем с помощью команды ***chmod +x dump_script.sh*** установим скрипту права на выполнения.

И через утилиту *cron* установим выполнение данного скрипта в 3 часа ночи.

crontab -e

0 3 * * * /var/db/postgres0/dump_script.sh

Теперь рассчитаем объём и анализ результатов:

1. Объём данных за сутки:

   - Новые данные – 550 Мб
   - Изменённые данные – 100 Мб
   - Всего данных – 550 + 100 = 650 Мб

2. Объём данных за месяц появления и изменения данных:

   650 Мб · 30 дней = 19500 Мб

3. Объём данных всех резервных копий за месяц:

   Нетрудно догадаться, что ответом будет сумма арифметической прогрессии, где первый член будет равен 650, а последний член будет равен 19500.

   Ответ: $\frac{(19500+6500)}{2} 30 = 302\,250$ Мб $= 302{,}250$ Гб

Этап 2. Потеря основного узла

Задание:

Этот сценарий подразумевает полную недоступность основного узла. Необходимо восстановить работу СУБД на РЕЗЕРВНОМ узле, продемонстрировать успешный запуск СУБД и доступность данных.

Выполнение:

Представим, что основной узел повреждён и теперь перейдём на резервный узел и попробуем восстановить БД.

Напишем скрипт, который при запуске автоматически подготовит среду для восстановления и восстановит кластер.
Скрипт recovery_script.sh:

```
# recovery_script.sh

export PGDATA=$HOME/postgres/zeb22
export TBDATA=$HOME/postgres/rez82
export PGPORT=9745

# Создаём директорию для инициализации нового кластера, в который мы будем восстанавливать наш оригинальный кластер
mkdir -p $PGDATA

# Создаём директорию для табличного пространства
mkdir -p $TBDATA

# Инициализируем новый кластер
initdb -D $PGDATA

# Запустим новый сервер
pg_ctl -D $PGDATA start

# Создаём само табличное пространство
psql -U postgres1 -p 9745 -d postgres -c "CREATE TABLESPACE rez82 LOCATION '/var/db/postgres1/postgres/rez82';"

# Разархивируем дамп, чтобы уже потом восстановить кластер по нему.
gunzip -c "$(ls -t dumps/postgres/dump_*.sql.gz | head -n1)" | psql -U postgres1 -d postgres -p 9745

# Проверка, что сервер отвечает и всё хорошо
pg_ctl -D $PGDATA status
```

После этого для проверки, что все наши объекты восстановились выполним команды:
psql -U postgres1 -p 9745 -d postgres -c "\l"
psql -U postgres1 -p 9745 -d postgres -c "\db+"
psql -U postgres1 -p 9745 -d postgres -c "\du+"

Этап 3. Повреждение файлов БД

Задание:

Этот сценарий подразумевает потерю данных (например, в результате сбоя диска или файловой системы) при сохранении доступности основного узла. Необходимо выполнить полное восстановление данных из резервной копии и перезапустить СУБД на ОСНОВНОМ узле.

Ход работы:

- Симулировать сбой:
  - удалить с диска директорию конфигурационных файлов СУБД со всем содержимым.
- Проверить работу СУБД, доступность данных, перезапустить СУБД, проанализировать результаты.
- Выполнить восстановление данных из резервной копии, учитывая следующее условие:
  - исходное расположение директории PGDATA недоступно - разместить данные в другой директории и скорректировать конфигурацию.
- Запустить СУБД, проверить работу и доступность данных, проанализировать результаты.

Выполнение:

Попробуем продемонстрировать сбой. Напишем скрипт, который удалит директорию с кластером, а потом запустит восстановление из дампа кластер.

Скрипт recovery.sh:

```
#!/bin/bash
set -e

# === Настройки ===
export PGDATA=$HOME/zeb22
export TBDATA=$HOME/rez82
export PGDATA_NEW=$HOME/zeb22_new
export TBDATA_NEW=$HOME/rez82_new
export PGPORT=9745
export REMOTE_DUMP_DIR=/var/db/postgres1/dumps/postgres
```

```
export LOCAL_DUMP_DIR=$HOME/dumps/postgres

# Симуляция сбоя
echo "Удаляем старый кластер..."
rm -rf $PGDATA
rm -rf $TBDATA

echo "Кластер и табличное пространство удалены."

# Копируем последнюю резервную копию с резервного узла
echo "Копируем последнюю резервную копию с резервного узла..."
LATEST_REMOTE_DUMP=$(ssh postgres1@pg112 "ls -t
$REMOTE_DUMP_DIR/dump_*.sql.gz | head -n 1")
scp postgres1@pg112:"$LATEST_REMOTE_DUMP" $LOCAL_DUMP_DIR/

echo "Резервная копия скопирована: $(basename $LATEST_REMOTE_DUMP)"

# Восстанавливаем кластер
echo "Создаём директории для нового кластера..."
mkdir -p $PGDATA
mkdir -p $TBDATA

echo "Инициализируем новый кластер..."
initdb -D $PGDATA

echo "Запускаем PostgreSQL..."
pg_ctl -D $PGDATA -o "-p $PGPORT" start
sleep 2

# Создаём табличное пространство
psql -U postgres0 -p $PGPORT -d postgres -c "CREATE TABLESPACE rez82 LOCATION
'$TBDATA';"

# Восстанавливаем из дампа
DUMP_FILE=$(ls -t $LOCAL_DUMP_DIR/dump_*.sql.gz | head -n1)
echo "Восстанавливаем данные из $DUMP_FILE..."
gunzip -c "$DUMP_FILE" | psql -U postgres0 -p $PGPORT -d postgres

# Проверка
echo "Проверка статуса кластера..."
pg_ctl -D $PGDATA status

echo "Список баз данных:"
psql -U postgres0 -p $PGPORT -d postgres -c "\l"

echo "Список табличных пространств:"
```

```
psql -U postgres0 -p $PGPORT -d postgres -c "\db+"

echo "Список ролей:"
psql -U postgres0 -p $PGPORT -d postgres -c "\du+"

echo "Восстановление завершено!"
```

Этап 4. Логическое повреждение данных

Задание:

Этот сценарий подразумевает частичную потерю данных (в результате нежелательной или ошибочной операции) при сохранении доступности основного узла. Необходимо выполнить восстановление данных на ОСНОВНОМ узле следующим способом:

- Восстановление с использованием архивных WAL файлов. (СУБД должна работать в режиме архивирования WAL, потребуется задать параметры восстановления).

Ход работы:

- В каждую таблицу базы добавить 2-3 новые строки, зафиксировать результат.
- Зафиксировать время и симулировать ошибку:
  - в любой таблице с внешними ключами подменить значения ключей на случайные (INSERT, UPDATE)
- Продемонстрировать результат.
- Выполнить восстановление данных указанным способом.
- Продемонстрировать и проанализировать результат.

Выполнение:

Для начала перед выполнением Point-in-Time Recovery настроим в файле конфигурации postgres.conf архивирование WAL-файлов.

```
archive_mode = on
archive_command = 'scp %p postgres1@pg112:/var/db/postgres1/wal_archive/%f'
wal_level = replica
archive_timeout = 60
```

Создадим копию кластера с помощью команды, так как восстановление при помощи wal-архивирования работает только с физической копией:
pg_basebackup -p 9745 -D $HOME/backup -Ft -Xs -P

Посмотрим на нашу таблицу customers из БД nicebluemom, с ней мы и будем работать:

```
nicebluemom=# SELECT * FROM customers;
 customer_id | first_name | last_name |       email         |    phone     |         address
-------------+------------+-----------+---------------------+--------------+-------------------------
           1 | John       | Doe       | john.doe@example.com | 123-456-7890 | 1234 Elm St, Springfield
           2 | Alice      | Smith     | alice@example.com    | 111-222-3333 | Sunset Blvd
           3 | Bob        | Brown     | bob@example.com      | 222-333-4444 | Oak Street
(3 строки)
```

```
# Добавим новые данные
INSERT INTO customers (first_name, last_name, email, phone, address)
VALUES ('Julia', 'Oseledko', 'juuul.osel@example.com', '123-456-7890',
'Krasnodar');

# Зафиксируем время
select pg_switch_wal();
select now();
```

Получим вот такую таблицу:

```
niceбluemom=# SELECT * FROM customers;
 customer_id | first_name | last_name |        email        |    phone     |         address
-------------+------------+-----------+---------------------+--------------+-------------------------
           1 | John       | Doe       | john.doe@example.com | 123-456-7890 | 1234 Elm St, Springfield
           2 | Alice      | Smith     | alice@example.com    | 111-222-3333 | Sunset Blvd
           3 | Bob        | Brown     | bob@example.com      | 222-333-4444 | Oak Street
           5 | Ravil      | Bayanov   | jejfje               | 123-456-7890 | 1234 Elm St, Springfield
(4 строки)
```

С помощью этой команды симулируем ошибку в БД:

```
delete from customers where customer_id % 2 = 0;
```

Получаем такую БД:

```
niceбluemom=# SELECT * FROM customers;
 customer_id | first_name | last_name |        email        |    phone     |         address
-------------+------------+-----------+---------------------+--------------+-------------------------
           1 | John       | Doe       | john.doe@example.com | 123-456-7890 | 1234 Elm St, Springfield
           3 | Bob        | Brown     | bob@example.com      | 222-333-4444 | Oak Street
           5 | Ravil      | Bayanov   | jejfje               | 123-456-7890 | 1234 Elm St, Springfield
(3 строки)
```

Начинаем восстановление:

```
# Начнём восстановление

pg_ctl -D $HOME/zeb22 stop
cp $HOME/zeb22/pg_wal/* $HOME/wal_dir/
scp wal_dir/* postgres1@pg112:~/wal_archive/

rm-rf zeb22/
rm-rf rez82/

mkdir pg_data_recovery
mkdir rez82
chmod 750 pg_data_recovery/

tar -xvf $HOME/backup/base.tar -C $HOME/pg_data_recovery
tar -xvf $HOME/backup/pg_wal.tar -C $HOME/pg_data_recovery/pg_wal
TABLESPACE_ARCHIVE=$(ls $HOME/backup | grep -E '^[0-9]+\.tar$' | head -n 1)
tar -xvf $HOME/backup/$TABLESPACE_ARCHIVE -C $HOME/rez82
cp wal_dir/* pg_data_recovery/pg_wal/

echo "
restore_command = 'scp postgres1@pg112:~/wal_archive/%f %p'
recovery_target_time = ''
recovery_target_action = promote " >> $HOME/pg_data_recovery/postgres.conf

touch recovery.signal
pg_ctl -D $HOME/pg_data_recovery start
```

После запуска сервера, получаем БД, которая была до фиксации времени:

```
nicebluemom=# select * from customers;
 customer_id | first_name | last_name |        email         |    phone     |         address
-------------+------------+-----------+----------------------+--------------+--------------------------
           1 | John       | Doe       | john.doe@example.com | 123-456-7890 | 1234 Elm St, Springfield
           2 | Alice      | Smith     | alice@example.com    | 111-222-3333 | Sunset Blvd
           3 | Bob        | Brown     | bob@example.com      | 222-333-4444 | Oak Street
           5 | Ravil      | Bayanov   | jejfje               | 123-456-7890 | 1234 Elm St, Springfield
(4 строки)
```

Таким образом нам удалось осуществить Point-in-Time Recovery.

Сложности

При выполнении данной лабораторной работа, сложно было выполнить 4 этап, так как не сразу было понятно, как можно с помощью wal-архивов откатывать систему назад с помощью Point-In-Time Recovery.

Вывод

Выполнив данную лабораторную работу, я научился копировать свой кластер баз данных на основе логического резервирования. Также с помощью wal-архивирования у меня получилось откатить кластер баз данных к последним важным изменениям.