МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Интеллектуальные системы информационной безопасности»

Политики безопасности баз данных

Практическая работа № 5

ОТЧЁТ

Выполнил студент группы
БСБО-07-20
Любовский С.В.

Москва 2023

Выполнение задания 1.

1. Создадим конфигурацию на два сервера

```yaml
docker-compose.yml - The Compose specification establishes a standard for the definition of mult
version: '3.9'

services:
    postgres1:
        image: postgres:15.2-alpine
        environment:
            - POSTGRES_USER=postgres
            - POSTGRES_PASSWORD=postgres
            - POSTGRES_DATABASE=postgres
    postgres2:
        image: postgres:15.2-alpine
        environment:
            - POSTGRES_USER=postgres
            - POSTGRES_PASSWORD=postgres
            - POSTGRES_DATABASE=postgres
```

2. Создадим тестовые таблиц на первом сервере

```
postgres=# create database test1;
CREATE DATABASE
postgres=# create database test2;
CREATE DATABASE
```

Создадим таблицу и данные в ней (для базы test1)

```
test1=# create table test_t1(
id serial primary key,
name varchar(100)
);
CREATE TABLE
test1=# insert into test_t1(name) values ('volk'), ('auf');
INSERT 0 2
```

Тоже самое для базы test2

3. При помощи pg_dumpall создадим дамп глобальных объектов

```
➜ practice5 (main) X docker compose exec -it postgres1 sh
/ # pg_dumpall -g -U postgres
--
-- PostgreSQL database cluster dump
--

SET default_transaction_read_only = off;

SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;

--
-- Roles
--

CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRLS PASSWORD 'SCRAM-SHA-256$4096:g2NxekxiYBhhzzmdnElPVg==$CAp0mE2LMXXorH0SoGRIxdvhNKystHfwIkrFCJzY
YRM=:/ZL2fKcrO9MfrliCCZce81swj5lWEipf/ODUOVYYn3c=';

--
-- User Configurations
--

--
-- PostgreSQL database cluster dump complete
--
```

4. При помощи pg_dump создадим дамп только выбранных таблиц

```
/ # pg_dump -U postgres -d test2 -t test_t1 -f /dump.sql
/ # cat dump.sql
--
-- PostgreSQL database dump
--

-- Dumped from database version 15.2
-- Dumped by pg_dump version 15.2

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: test_t1; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.test_t1 (
    id integer NOT NULL,
    name character varying(100)
);


ALTER TABLE public.test_t1 OWNER TO postgres;

--
-- Name: test_t1_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--
```

5. Используя эту резервную копию, восстановим данные на другом кластере

```
→ practice5 (main) X docker compose exec -it postgres2 sh
/ # pg_res
pg_resetwal  pg_restore
/ # psql -U postgres < ./dump.sql
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 2
 setval
--------
      2
(1 row)

ALTER TABLE
/ # _
```

Проверим таблицу

```
postgres=# select * from
information_schema.  public.            test_t1            test_t1_id_seq
postgres=# select * from test_t1
postgres-# ;
 id | name
----+-------
  1 | volk1
  2 | auf1
(2 rows)

postgres=#
```

Выполнение задания 2.

1. Параметры журналирования
   - log_destination - В log_destination указывается один или несколько методов протоколирования, разделённых запятыми. Выбран syslog как стандарт удобный для дальнейшей обработки.

- log_directory - При включённом logging_collector, определяет каталог, в котором создаются журнальные файлы. Использую дефолтное значение – log
- log_filename - Значение трактуется как строка формата в функции strftime, поэтому в ней можно использовать спецификаторы % для включения в имена файлов информации о дате и времени. Использовано стандартное значение
- log_rotation_age - Определяет максимальное время жизни отдельного журнального файла, при включённом logging_collector. После того как прошло заданное количество минут, создаётся новый журнальный файл. Использовано значение 1d.
- log_rotation_size - Определяет максимальный размер отдельного журнального файла, при включённом logging_collector. Выбрано 100 MB.

2. Параметры резервного копирования:
   - backup_mode - Выбирает режим резервного копирования. Используется значение FULL для создания полной резервной копии, содержащей все файлы данных кластера, необходимых для его восстановления.
   - archive_mode - Когда параметр archive_mode включён, полные сегменты WAL передаются в хранилище архива командой archive_command. Помимо значения off (выключающего архивацию) есть ещё два: on (вкл.) и always (всегда). В обычном состоянии эти два режима не различаются, но в режиме always архивация WAL активна и во время восстановления архива, и при использовании ведомого сервера. В этом режиме все файлы, восстановленные из архива или полученные при потоковой репликации, будут архивироваться (снова). Выставлено значение on для дальнейшего использования параметра archive_command.
   - archive_command - Команда локальной оболочки, которая будет выполняться для архивации завершённого сегмента WAL. Любое вхождение %p в этой строке заменяется путём архивируемого файла, а вхождение %f заменяется только его именем. Используется значение “tar -czvf /var/lib/postgres/bak/%f.tar.gz %p”.
   - restore_command - Команда оболочки ОС, которая выполняется для извлечения архивного сегмента файлов WAL. Этот параметр требуется для восстановления из архива, но необязателен для потоковой репликации. Используется значение “tar -xzvf /var/lib/postgres/bak/%f.tar.gz -C %p”
   - recovery_target_timeline - Указывает линию времени для восстановления. По умолчанию производится восстановление той же линии времени, которая была текущей в момент создания базовой резервной копии. Выбрано значение current для восстановления той же линии времени, которая была текущей в момент создания базовой резервной копии.

3. Параметры безопасности паролей
    - password_encryption - Доступность различных методов аутентификации по паролю зависит от того, как пароли пользователей хешируются. Использовано значение scram-sha-256 т.к. Это наиболее безопасный из существующих на данный момент методов.
    - password_strength_check - Модуль проверки пароля проверяет пароли пользователей всякий раз, когда они устанавливаются с помощью CREATE ROLE или ALTER ROLE. Если пароль будет сочтен слишком слабым, он будет отклонен, и выполнение команды завершится с ошибкой.
4. Параметры безопасности аутентификации
    - authentication_timeout - Устанавливает максимально допустимое время для завершения аутентификации клиента. Оставлено значение по умолчанию.
    - ssl_cert_file - отправляется клиенту для идентификации сервера. (aka Публичный ключ)
    - ssl_key_file – используется для дешифровки на стороне сервера (aka приватный ключ)
5. Параметры безопасности доступа к файлам
    - data_directory – Каталог с файлами данных. Можно установить только при первом запуске.
    - data_file_mode – Права на файлы данных.
    - data_directory_mode – Права на директории с данными.
6. Параметры безопасности запросов
    - max_statement_time - Если установлено ненулевое значение, любые запросы, которые занимают больше времени в секундах, будут прерваны. Значение по умолчанию равно нулю, и в этом случае ограничения не применяются.
    - statement_timeout - Задаёт максимальное время выполнения оператора (в миллисекундах), начиная с момента получения сервером команды от клиента, по истечении которого оператор прерывается. Установлен дефолт – 0.
7. Параметры безопасности сеанса
    - idle_in_transaction_session_timeout - Завершать любые сеансы, в которых открытая транзакция простаивает дольше заданного (в миллисекундах) времени. Это позволяет освободить все блокировки сеанса и вновь задействовать слот подключения; также это позволяет очистить кортежи, видимые только для этой транзакции. Установлено значение 0, с целью отключения данного функционала
    - lock_timeout - Задаёт максимальную длительность ожидания (в миллисекундах) любым оператором получения блокировки таблицы, индекса, строки или другого объекта базы данных. Если ожидание не закончилось за указанное время, оператор прерывается. Это ограничение действует на каждую попытку получения блокировки по

отдельности и применяется как к явным запросам блокировки (например, LOCK TABLE или SELECT FOR UPDATE без NOWAIT) так и к неявным. При значении, равном нулю (по умолчанию), этот контроль длительности отключается.

Выполнение задания 3.

1. Создадим тестовую базу данных

```
postgres=# CREATE DATABASE test;
CREATE DATABASE
postgres=# _
```

2. Добавим pgbouncer в конфигурацию

```
      - POSTGRES_DATABASE=postgres
  pgbouncer:
    image: edoburu/pgbouncer
    environment:
      - DB_HOST=postgres1
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - DB_NAME=test
    ports:
      - 5432:5432
```

3. Запустим и проверим логи

```
practice5-pgbouncer-1  | Wrote authentication credentials to /etc/pgbouncer/userlist.txt
practice5-pgbouncer-1  | Create pgbouncer config in /etc/pgbouncer
practice5-pgbouncer-1  | ################## Auto generated ##################
practice5-pgbouncer-1  | [databases]
practice5-pgbouncer-1  | test = host=postgres1 port=5432 user=postgres
practice5-pgbouncer-1  |
practice5-pgbouncer-1  | [pgbouncer]
practice5-pgbouncer-1  | listen_addr = 0.0.0.0
practice5-pgbouncer-1  | listen_port = 5432
practice5-pgbouncer-1  | unix_socket_dir =
practice5-pgbouncer-1  | user = postgres
practice5-pgbouncer-1  | auth_file = /etc/pgbouncer/userlist.txt
practice5-pgbouncer-1  | auth_type = md5
practice5-pgbouncer-1  | ignore_startup_parameters = extra_float_digits
practice5-pgbouncer-1  |
practice5-pgbouncer-1  | # Log settings
practice5-pgbouncer-1  | admin_users = postgres
practice5-pgbouncer-1  |
practice5-pgbouncer-1  | # Connection sanity checks, timeouts
practice5-pgbouncer-1  |
practice5-pgbouncer-1  | # TLS settings
practice5-pgbouncer-1  |
practice5-pgbouncer-1  | # Dangerous timeouts
practice5-pgbouncer-1  | ################## end file ##################
practice5-pgbouncer-1  | Starting /usr/bin/pgbouncer /etc/pgbouncer/pgbouncer.ini...
practice5-pgbouncer-1  | 2023-04-30 12:40:36.410 UTC [1] LOG kernel file descriptor limit: 1048576 (hard: 1048576); max_client_conn: 100, max expected fd use: 132
practice5-pgbouncer-1  | 2023-04-30 12:40:36.416 UTC [1] LOG listening on 0.0.0.0:5432
practice5-pgbouncer-1  | 2023-04-30 12:40:36.416 UTC [1] LOG process up: PgBouncer 1.18.0, libevent 2.1.12-stable (epoll), adns: udns 0.4, tls: OpenSSL 3.0.7 1 Nov 2022
practice5-pgbouncer-1  | 2023-04-30 12:41:36.416 UTC [1] LOG stats: 0 xacts/s, 0 queries/s, in 0 B/s, out 0 B/s, xact 0 us, query 0 us, wait 0 us
_
```

4. Подключитесь к базе данных через PgBouncer с помощью утилиты psql и выполните несколько простых SQL-запросов.

```
→practice5 (main) X docker compose exec -it pgbouncer psql -U postgres -h localhost
Password for user postgres:
psql (15.1, server 15.2)
Type "help" for help.

postgres=# \c postgres
psql (15.1, server 15.2)
You are now connected to database "postgres" as user "postgres".
postgres=# select name from pg_settings;
                    name
-----------------------------------------
 allow_in_place_tablespaces
 allow_system_table_mods
 application_name
 archive_cleanup_command
 archive_command
 archive_library
 archive_mode
 archive_timeout
 array_nulls
```

5. Измените настройки PgBouncer для повышения производительности. Было увеличено кол-во подключений до 5000
6. Проверим бенчмарк в прямом подключении к базе и через pgbouncer Сначала напрямую

```
/ $ pgbench -c 10 -T 30
pgbench (15.2)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
maximum number of tries: 1
duration: 30 s
number of transactions actually processed: 8778
number of failed transactions: 0 (0.000%)
latency average = 34.160 ms
initial connection time = 47.388 ms
tps = 292.738104 (without initial connection time)
```

Теперь через pgbouncer

```
→ practice5 (main) ✗ docker compose exec -it pgbouncer sh
/ $ pgbench -h localhost -c 10 -T 30 test
Password:
pgbench (15.1, server 15.2)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
maximum number of tries: 1
duration: 30 s
number of transactions actually processed: 13599
number of failed transactions: 0 (0.000%)
latency average = 22.072 ms
initial connection time = 4.805 ms
tps = 453.052618 (without initial connection time)
/ $ _
```

Видно что он лучше справляется с множеством подключений

7. В логах можно найти подключения и отключения клиентов

```
8448)
practice5-pgbouncer-1  | 2023-04-30 13:17:42.403 UTC [1] LOG C-0x7fd06784c910: test/postgres@127.0.0.1:53252 closing because: client close request (age=
0s)
practice5-pgbouncer-1  | 2023-04-30 13:17:42.403 UTC [1] LOG C-0x7fd06784c910: test/postgres@127.0.0.1:53260 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.404 UTC [1] LOG C-0x7fd06784cb70: test/postgres@127.0.0.1:53268 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.404 UTC [1] LOG C-0x7fd06784cdd0: test/postgres@127.0.0.1:53272 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.405 UTC [1] LOG C-0x7fd06784d030: test/postgres@127.0.0.1:53282 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.405 UTC [1] LOG C-0x7fd06784d290: test/postgres@127.0.0.1:53286 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.406 UTC [1] LOG C-0x7fd06784d4f0: test/postgres@127.0.0.1:53294 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.406 UTC [1] LOG C-0x7fd06784d750: test/postgres@127.0.0.1:53310 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.407 UTC [1] LOG C-0x7fd06784d9b0: test/postgres@127.0.0.1:53314 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.407 UTC [1] LOG C-0x7fd06784dc10: test/postgres@127.0.0.1:53322 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.408 UTC [1] LOG C-0x7fd06784de70: test/postgres@127.0.0.1:53336 login attempt: db=test user=postgres tls=no
practice5-pgbouncer-1  | 2023-04-30 13:17:42.408 UTC [1] LOG S-0x7fd067811b90: test/postgres@172.25.0.3:5432 new connection to server (from 172.25.0.2:5
8452)
practice5-pgbouncer-1  | 2023-04-30 13:17:42.410 UTC [1] LOG S-0x7fd067811df0: test/postgres@172.25.0.3:5432 new connection to server (from 172.25.0.2:5
8466)
practice5-pgbouncer-1  | 2023-04-30 13:17:42.412 UTC [1] LOG S-0x7fd067812050: test/postgres@172.25.0.3:5432 new connection to server (from 172.25.0.2:5
8478)
practice5-pgbouncer-1  | 2023-04-30 13:17:42.415 UTC [1] LOG S-0x7fd0678122b0: test/postgres@172.25.0.3:5432 new connection to server (from 172.25.0.2:5
8492)
practice5-pgbouncer-1  | 2023-04-30 13:17:42.417 UTC [1] LOG S-0x7fd067812510: test/postgres@172.25.0.3:5432 new connection to server (from 172.25.0.2:5
8508)
```

Выполнение задания 4.

1. Создадим две БД

```
postgres=# create database source_db;
CREATE DATABASE
postgres=# create database taget_db;
CREATE DATABASE
postgres=# _
```

2. В source_db создадим таблицу employees

```
postgres=# \c source_db;
You are now connected to database "source_db" as user "postgres".
source_db=# create table employees(id serial, name varchar(255), salary numeric(18,2));
CREATE TABLE
```

3. Добавим пару записей

```
source_db=# insert into employees(name, salary) values ('ivan', 1000.50), ('kate', 2590.25)
source_db-# ;
INSERT 0 2
source_db=# select * from employees;
 id | name | salary
----+------+---------
  1 | ivan | 1000.50
  2 | kate | 2590.25
(2 rows)

source_db=# _
```

4. Установим расширение postgres_fdw

```
source_db=# create extension postgres_fdw;
CREATE EXTENSION
source_db=# _
```

5. Создадим сервер

```
source_db=# create server target_server foreign data wrapper postgres_fdw options (host 'localhost', dbname 'target_db', port '5432');
CREATE SERVER
source_db=#
```

6. Создадим маппинг на внешнем сервере

```
source_db=# create user mapping for postgres server target_server options (user 'postgres', password 'postgres');
CREATE USER MAPPING
source_db=# create foreign table employees_mapping (id serial primary key, name varchar(255), salary numeric(18,2)) server target_serve
ERROR:  primary key constraints are not supported on foreign tables
LINE 1: create foreign table employees_mapping (id serial primary ke...
                                                ^
source_db=# create foreign table employees_mapping (id serial, name varchar(255), salary numeric(18,2)) server target_server;
CREATE FOREIGN TABLE
source_db=#
```

7. Выполним запрос ко внешней таблице

```
source_db=# select * from employees_mapping;
 id | name | salary
----+------+---------
  1 | ivan | 1000.50
  2 | kate | 2590.25
(2 rows)

source_db=# _
```