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»

Администрирование баз данных

Практическая работа № 6

ОТЧЁТ

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

Москва 2023

Выполнение задания 1.

1-2. Создадим конфигурацию кластера используя docker-compose

```yaml
services:
    mongo1:
        container_name: mongo1
        image: mongo:6-jammy
        volumes:
            - ./scripts/rs-init.sh:/scripts/rs-init.sh
            - ./scripts/init.js:/scripts/init.js
        networks:
            - mongo-network
        ports:
            - 27017:27017
        depends_on:
            - mongo2
            - mongo3
        links:
            - mongo2
            - mongo3
        restart: always
        entrypoint:
            [
                "/usr/bin/mongod",
                "--bind_ip_all",
                "--replSet",
                "dbrs"
            ]

    mongo2:
        container_name: mongo2
        image: 6-jammy
        networks:
            - mongo-network
        ports:
            - 27018:27017
        restart: always
        entrypoint:
            [
                "/usr/bin/mongod",
                "--bind_ip_all",
                "--replSet",
                "dbrs"
            ]

    mongo3:
        container_name: mongo3
```

Используя скрипт для подготовки базы включим репликацию на всех узлах

```bash
#!/bin/bash

DELAY=25

mongo <<EOF
var config = {
    "_id": "dbrs",
    "version": 1,
    "members": [
        {
            "_id": 1,
            "host": "mongo1:27017",
            "priority": 2
        },
        {
            "_id": 2,
            "host": "mongo2:27017",
            "priority": 1
        },
        {
            "_id": 3,
            "host": "mongo3:27017",
            "priority": 1
        }
    ]
};
rs.initiate(config, { force: true });
EOF

echo "****** Waiting for ${DELAY} seconds for replicaset configuration to be applied ******"

sleep $DELAY

mongo < /scripts/init.js
```

После успешной установки и запуска контейнеров зайдем в консоль primary ноды и увидим результат

```
dbrs [direct: primary] test> rs.status()
{
  set: 'dbrs',
  date: ISODate("2023-04-05T11:31:56.689Z"),
  myState: 1,
  term: Long("1"),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long("2000"),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1680694311, i: 1 }), t: Long("1") },
    lastCommittedWallTime: ISODate("2023-04-05T11:31:51.832Z"),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1680694311, i: 1 }), t: Long("1") },
    appliedOpTime: { ts: Timestamp({ t: 1680694311, i: 1 }), t: Long("1") },
    durableOpTime: { ts: Timestamp({ t: 1680694311, i: 1 }), t: Long("1") },
    lastAppliedWallTime: ISODate("2023-04-05T11:31:51.832Z"),
    lastDurableWallTime: ISODate("2023-04-05T11:31:51.832Z")
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1680694271, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate("2023-04-05T11:27:31.625Z"),
    electionTerm: Long("1"),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1680694040, i: 1 }), t: Long("-1") },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1680694040, i: 1 }), t: Long("-1") },
    numVotesNeeded: 2,
    priorityAtElection: 2,
    electionTimeoutMillis: Long("10000"),
    numCatchUpOps: Long("0"),
    newTermStartDate: ISODate("2023-04-05T11:27:31.754Z"),
    wMajorityWriteAvailabilityDate: ISODate("2023-04-05T11:27:33.035Z")
  },
  members: [
    {
      _id: 1,
      name: 'mongo1:27017',
      health: 1,
```

3. Создадим коллекцию на primary узле и добавим туда данных

```
dbrs [direct: primary] test> db.a.insertOne({field_a: 2, field_b: 3})
{
  acknowledged: true,
  insertedId: ObjectId("642d5d31711e2bb29da35d89")
}
dbrs [direct: primary] test> db.a.find
db.a.find              db.a.findOne           db.a.findOneAndDelete   db.a.findOneAndReplace  db.a.findOneAndUpdate

dbrs [direct: primary] test> db.a.find()
[
  { _id: ObjectId("642d5d31711e2bb29da35d89"), field_a: 2, field_b: 3 }
]
dbrs [direct: primary] test>
```

Перейдем на другой узел и проверим состояние коллекции

```
dbrs [direct: secondary] test> db.a.find()
[
  { _id: ObjectId("642d5d31711e2bb29da35d89"), field_a: 2, field_b: 3 }
]
dbrs [direct: secondary] test> _
```

данные успешно реплицируются между нодами.

4. Остановим наш primary узел

```
→ mongo (main) X docker compose stop mongo1
[+] Running 1/1
 ⠿ Container mongo1  Stopped
```

Зайдем на один из узлов и посмотрим кто теперь главный

```
members: [
  {
    _id: 1,
    name: 'mongo1:27017',
    health: 0,
    state: 8,
    stateStr: '(not reachable/healthy)',
    uptime: 0,
    optime: { ts: Timestamp({ t: 0, i: 0 }), t: Long("-1") },
    optimeDurable: { ts: Timestamp({ t: 0, i: 0 }), t: Long("-1") },
    optimeDate: ISODate("1970-01-01T00:00:00.000Z"),
    optimeDurableDate: ISODate("1970-01-01T00:00:00.000Z"),
    lastAppliedWallTime: ISODate("2023-04-05T11:42:33.523Z"),
    lastDurableWallTime: ISODate("2023-04-05T11:42:33.523Z"),
    lastHeartbeat: ISODate("2023-04-05T11:43:24.299Z"),
    lastHeartbeatRecv: ISODate("2023-04-05T11:42:32.538Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: 'Error connecting to mongo1:27017 :: caused by :: Could not find address for
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 1,
    configTerm: 2
  },
  {
    _id: 2,
    name: 'mongo2:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 1193,
    optime: { ts: Timestamp({ t: 1680695003, i: 1 }), t: Long("2") },
    optimeDate: ISODate("2023-04-05T11:43:23.000Z"),
    lastAppliedWallTime: ISODate("2023-04-05T11:43:23.526Z"),
    lastDurableWallTime: ISODate("2023-04-05T11:43:23.526Z"),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1680694943, i: 1 }),
    electionDate: ISODate("2023-04-05T11:42:23.000Z"),
    configVersion: 1,
    configTerm: 2,
    self: true,
    lastHeartbeatMessage: ''
  }
```

Вторая нода стала главной.

5. Снова добавим коллекцию и проверим репликацию

```
dbrs [direct: primary] test> use test
already on db test
dbrs [direct: primary] test> db.b.insertOne({lol_kek: true})
{
  acknowledged: true,
  insertedId: ObjectId("642d5f3b07b078239aa8deb8")
}
dbrs [direct: primary] test> _
```

```
dbrs [direct: secondary] test> use test
already on db test
dbrs [direct: secondary] test> b.find()
ReferenceError: b is not defined
dbrs [direct: secondary] test> db.b.find()
[ { _id: ObjectId("642d5f3b07b078239aa8deb8"), lol_kek: true } ]
dbrs [direct: secondary] test> _
```

успешно реплицируется.

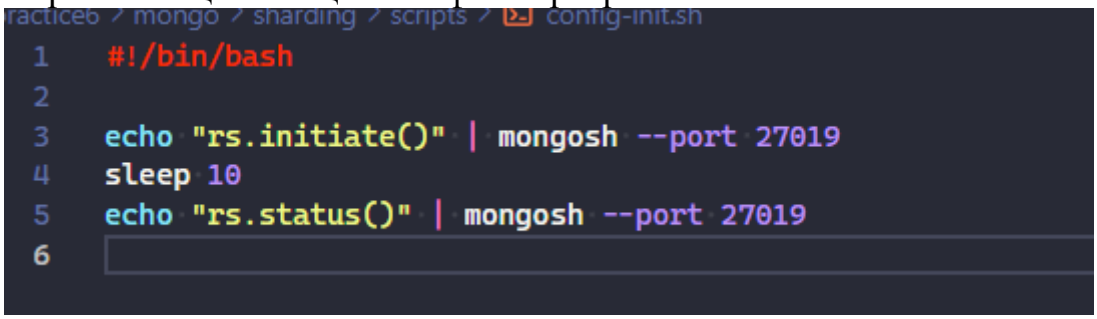
6. Теперь создадим кластер с шардированием. Для этого создадим новую конфигурацию docker-compose. Сконфигурируем 4 сервиса – конфиг-сервер,

2 шарда и mongos (роутер в шардированном кластере).

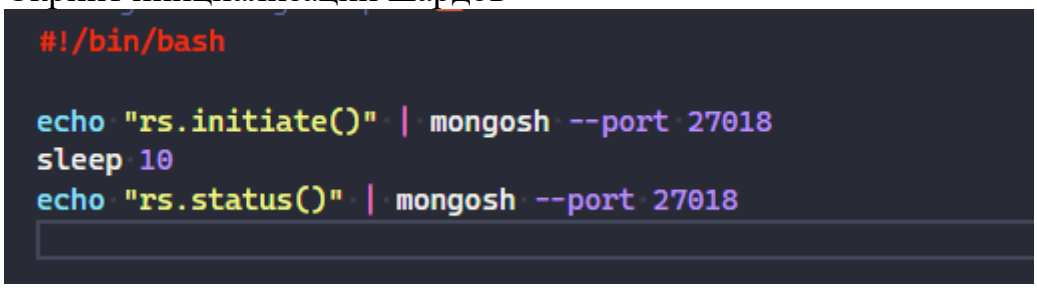
```yaml
# docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container pla
version: '3.9'

services:
  config-server:
    image: mongo:6-jammy
    container_name: mongo__config-server
    networks:
      - mongo-network
    volumes:
      - ./scripts/:/scripts/:ro
    entrypoint:
      [
        "/usr/bin/mongod",
        "--configsvr",
        "--bind_ip_all",
        "--replSet",
        "config"
      ]

  shard1:
    image: mongo:6-jammy
    container_name: mongo__shard-1
    networks:
      - mongo-network
    volumes:
      - ./scripts/:/scripts/:ro
    entrypoint:
      [
        "/usr/bin/mongod",
        "--shardsvr",
        "--bind_ip_all",
        "--replSet",
        "shard1"
      ]

  shard2:
    image: mongo:6-jammy
    container_name: mongo__shard-2
    networks:
      - mongo-network
    volumes:
      - ./scripts/:/scripts/:ro
    entrypoint:
      [
        "/usr/bin/mongod",
        "--shardsvr",
        "--bind_ip_all",
        "--replSet",
        "shard2"
      ]

  mongo-router:
    image: mongo:6-jammy
    container_name: mongo__mongos
    networks:
      - mongo-network
    volumes:
      - ./scripts/:/scripts/:ro
    entrypoint:
      [
        "/usr/bin/mongos",
        "--bind_ip_all",
        "--configdb",
        "config/config-server:27019"
      ]
```

Теперь поднимем нашу конфигурацию начиная с сервера и инициализируем

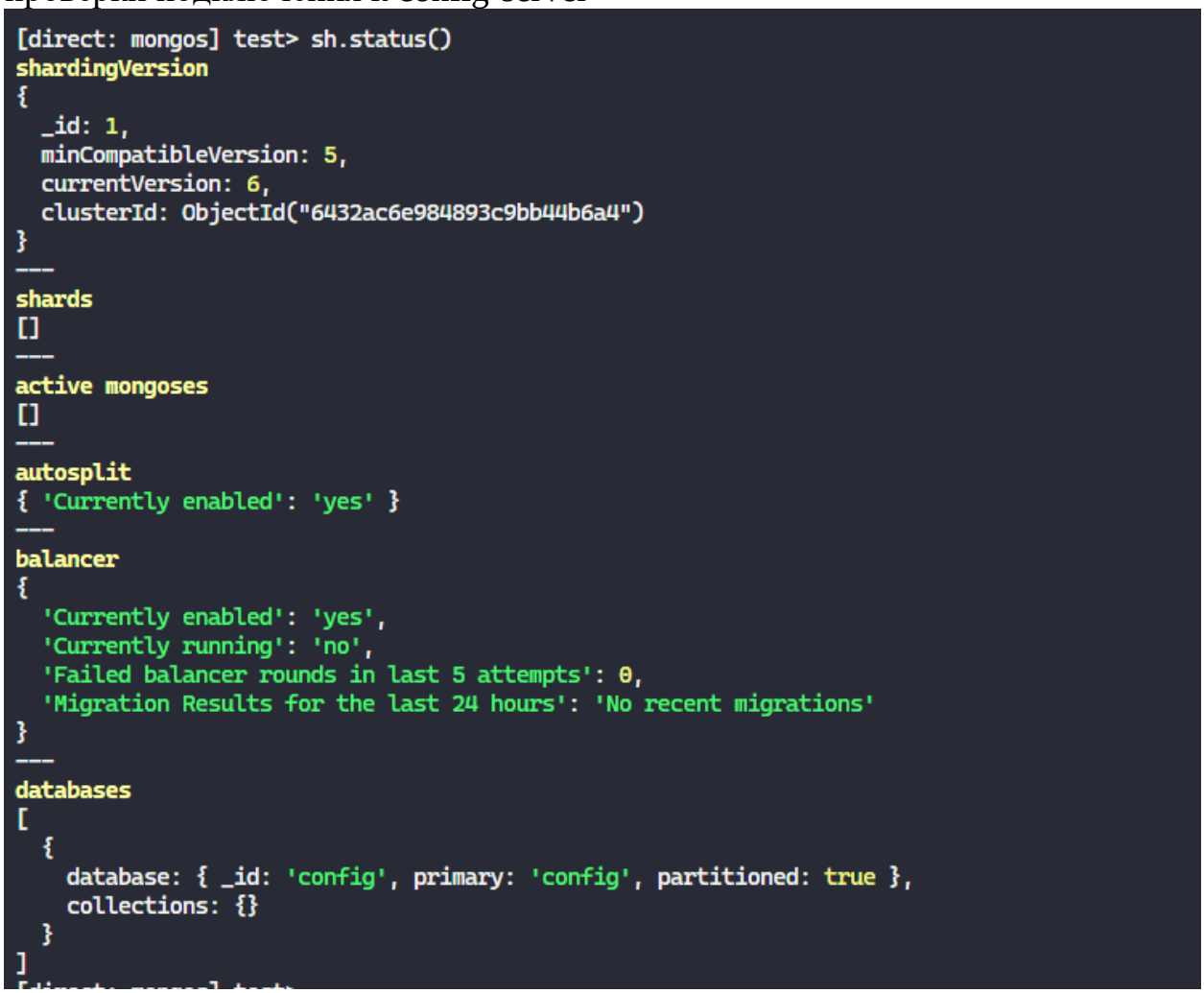
репликасеты и необходимые параметры репликации.
Скрипт инициализации конфиг-сервера



Скрипт инициализации шардов



Теперь зайдем в наш mongos сервис и пропишем следующую команду для проверки подключения к config-server



видно, что конфиг сервер успешно подключен. Теперь подключим наши

шарды.

```
[direct: mongos] test> sh.addShard("sh1/shard1:27018")
{
  shardAdded: 'sh1',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1681045079, i: 6 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1681045079, i: 6 })
}
```

```
[direct: mongos] test> sh.addShard("sh2/shard2:27018")
{
  shardAdded: 'sh2',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1681045207, i: 4 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1681045207, i: 4 })
}
[direct: mongos] test> _
```

7. Активируем шардирование на базе test

```
[direct: mongos] test> sh.enableSharding("test")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1681045439, i: 5 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1681045439, i: 3 })
}
```

Активируем шардирование на коллекции workers по полю _id используя метод шардирования hashed

```
[direct: mongos] test> sh.shardCollection("test.workers", { "_id": "hashed" })
{
  collectionsharded: 'test.workers',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1681045597, i: 40 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1681045597, i: 36 })
}
```

Добавим данных в коллекцию

```
[direct: mongos] test> db.workers.insertOne({"name": "Ivan", "surname": "Ivanov"})
{
  acknowledged: true,
  insertedId: ObjectId("6432b8c74e8993ce7d4a4364")
}
[direct: mongos] test> db.workers.insertOne({"name": "Petr", "surname": "Ivanov"})
{
  acknowledged: true,
  insertedId: ObjectId("6432b8d14e8993ce7d4a4365")
}
[direct: mongos] test> db.workers.insertOne({"name": "Yuri", "surname": "Ivanov"})
{
  acknowledged: true,
  insertedId: ObjectId("6432b8db4e8993ce7d4a4366")
}
```

Проверим статус шардирования коллекции

```
[direct: mongos] test> db.workers.getShardDistribution()
Shard sh1 at sh1/shard1:27018
{
  data: '114B',
  docs: 2,
  chunks: 2,
  'estimated data per chunk': '57B',
  'estimated docs per chunk': 1
}
---
Shard sh2 at sh2/shard2:27018
{
  data: '57B',
  docs: 1,
  chunks: 2,
  'estimated data per chunk': '28B',
  'estimated docs per chunk': 0
}
---
Totals
{
  data: '171B',
  docs: 3,
  chunks: 4,
  'Shard sh1': [
    '66.66 % data',
    '66.66 % docs in cluster',
    '57B avg obj size on shard'
  ],
  'Shard sh2': [
    '33.33 % data',
    '33.33 % docs in cluster',
    '57B avg obj size on shard'
  ]
}
[direct: mongos] test> _
```

8. Попробуем получить данные распределенные по шардам

```
[direct: mongos] test> db.workers.find()
[
  {
    _id: ObjectId("6432b8db4e8993ce7d4a4366"),
    name: 'Yuri',
    surname: 'Ivanov'
  },
  {
    _id: ObjectId("6432b8c74e8993ce7d4a4364"),
    name: 'Ivan',
    surname: 'Ivanov'
  },
  {
    _id: ObjectId("6432b8d14e8993ce7d4a4365"),
    name: 'Petr',
    surname: 'Ivanov'
  }
]
```

Посмотрим explain запроса

Видно что в плане присутствует MERGE из двух шардов.

Выполнение задания 2.

1. Создадим два сервиса (мастер и реплика)

```yaml
version: '3.9'

services:
  primary:
    image: postgres:15.2-alpine3.17
    container_name: postgres__primary

    environment:
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_DATABASE=postgres

  secondary:
    image: postgres:15.2-alpine3.17
    container_name: postgres__secondary

    environment:
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_DATABASE=postgres
```

2. Настроим мастер

Создадим юзера для репликации

```
/ # su - postgres
3c06a11988b2:~$ create
createdb    createuser
3c06a11988b2:~$ createuser --replication -P repluser
Enter password for new role:
Enter it again:
```

Настроим след параметры

```
wal_level = replica
max_wal_senders = 2
max_replication_slots = 2
hot_standby = on
hot_standby_feedback = on
```

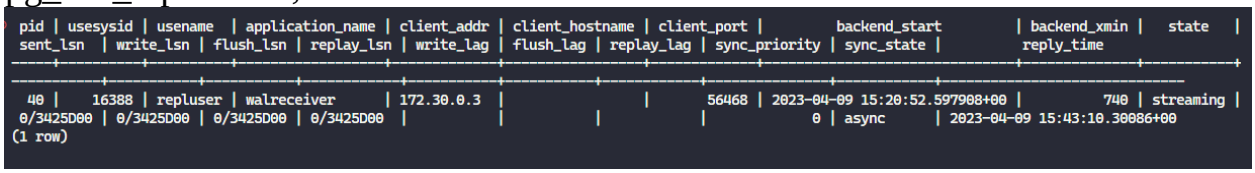
Настроим разрешенные для реплик адреса

```
# IPv6 local connections:
host    all             all             ::1/1
# Allow replication connections from localhos
# replication privilege.
local   replication     all
host    replication     all             127.0
host    replication     all             ::1/1

host all all all scram-sha-256
host replication all 172.30.0.3/16 md5
```

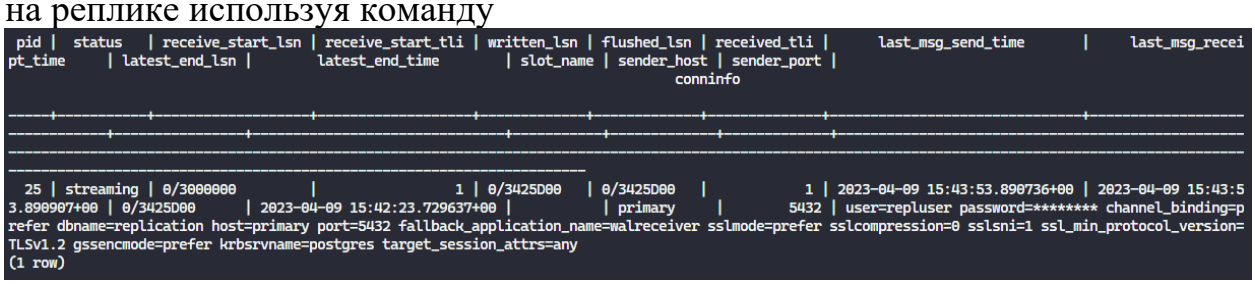
Перезапустим и перейдем к настройке сервера реплики.

Для начала удалим все данные из папки data, далее восстановим данные с нашего матер сервера используя команду
pg_basebackup --host=primary --username=repluser --pgdata=/var/lib/postgresql/data --wal-method=stream --write-recovery-conf
Теперь проверим статусы репликации:

- на мастере используя команду
  docker compose exec -it primary su - postgres -c "psql -c 'select * from pg_stat_replication;'"



- на реплике используя команду



Видим, что настройка прошла успешно.

3-4. Создадим базу и таблицу с данными на мастере и проверим, что данные реплицируются.

```
postgres=# CREATE DATABASE test
postgres-# ;
CREATE DATABASE
postgres=# \c test
You are now connected to database "test" as user "postgres".
test=# CREATE TABLE test_table(
id serial primary key,
name varchar(200)
);
CREATE TABLE
test=# insert into test_table (name) values ('111'), ('222');
INSERT 0 2
test=# _
```

```
➜ postgres (main) X docker compose exec -it secondary su - postgres
64e79aa01ed5:~$ psql
psql (15.2)
Type "help" for help.

postgres=# \c test;
You are now connected to database "test" as user "postgres".
test=# select * from test_table;
 id | name
----+------
  1 | 111
  2 | 222
(2 rows)

test=# _
```

Данные успешно реплицируются.

5. Попробуем создать запись на реплика-сервере

```
test=# insert into test_table (name) values ('333');
ERROR:  cannot execute INSERT in a read-only transaction
test=# _
```

Создать запись не получается, т.к. реплика-сервер находится в режиме read-only.

6-7.

```
test=# BEGIN;
BEGIN
test=*# select * from test_table;
 id | name
----+------
  1 | 111
  2 | 222
  3 | 333
(3 rows)

test=*# insert into test_table (name) values ('444'), ('555');
INSERT 0 2
test=*# delete from test_table where name = '333';
DELETE 1
test=*# commit;
COMMIT
test=# select * from test_table;
 id | name
----+------
  1 | 111
  2 | 222
  4 | 444
  5 | 555
(4 rows)
```

Чтение и запись на мастер сервере работают корректно.