Методическое пособие по выполнению домашнего задания по курсу «Администратор Linux. Professional»

Vagrant-стенд с Postgres

Цель домашнего задания

Научиться настраивать репликацию и создавать резервные копии в СУБД PostgreSQL

Описание домашнего задания

1) Настроить hot_standby репликацию с использованием слотов
2) Настроить правильное резервное копирование

пример плейбука:

```
---
- name: Установка postgres11
  hosts: master, slave
  become: yes
  roles:
   - postgres_install

- name: Настройка master
  hosts: master
  become: yes
  roles:
   - master-setup
```

Введение

PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД).

Основные термины в Postgres:

Кластер - объединение нескольких баз данных. В postgres это означает что на одном хосте создаётся несколько баз сразу.

База данных - физическое объединение объектов

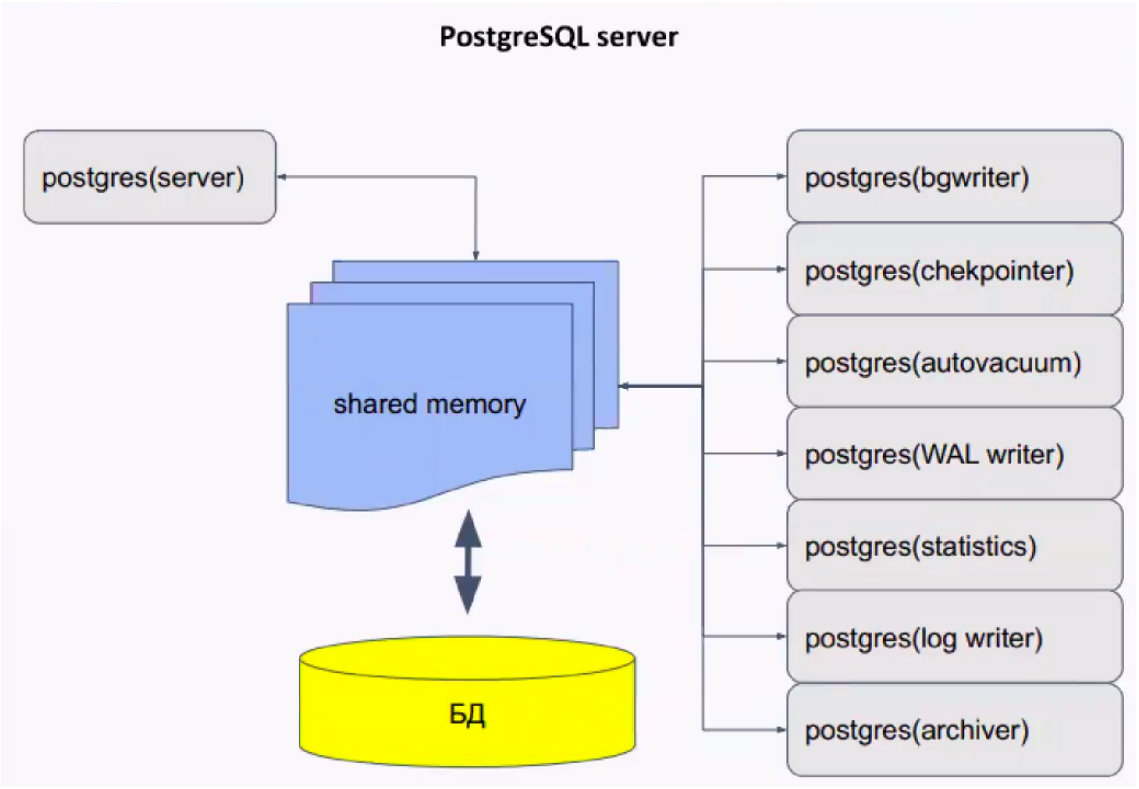
Схема - логическое объединение таблиц в базе данных. По умолчанию в postgres создаётся одна схема под названием Public

По умолчанию в кластере находятся:

- **template0** - read only БД, содержащая инициализационный набор данных
- **template1** - база-шаблон для создания новых баз
- **postgres** (при желании можно поменять название). В базе находятся служебные таблицы, можно также использовать данную базу для своих нужд, но это не рекомендуется.

Управлять базами, таблицами и данными можно не только с помощью консольной утилиты psql, но и с помощью GUI-утилит, например pgAdmin, Dbeaver и т. д.

Postgres - это мультироцессное приложение. Состоит из главного процесса (postgres), который отвечает за подключение клиентов, взаимодействие с кэшом и отвечает за остальные процессы (background processes).



Основные конфигурационные файлы в Postgres:

- pg_hba.conf - файл задаёт способ доступа к базам и репликации из различных источников.
- postgresql.conf - файл конфигурации, обычно находится в каталоге данных, может редактироваться вручную. Может быть несколько значений одного и того же параметра, тогда вступает в силу последнее значение.
- postgresql.auto.conf - предназначен для автоматического изменения параметров postgres

WAL (Write Ahead Log) - журнал упреждающей записи
В WAL записывается информация, достаточная для повторного выполнения всех действий с БД. Записи этого журнала обязаны попасть на диск раньше, чем изменения в соответствующей странице. Журнал состоит из нескольких файлов (обычно по 16МБ), которые циклически перезаписываются.

Репликация - процесс синхронизации нескольких копий одного объекта. Решает задачу отказоустойчивости и масштабируемости.

Задачи репликации:

- балансировка нагрузки
- резервирование (НЕ БЭКАП, бэкап можно делать с реплики)
- обновление без остановки системы
- горизонтальное масштабирование
- георасределение нагрузки

Виды репликации:

- Физическая репликация - описание изменений на уровне файлов. Побайтовая копия данных.
- Логическая репликация - изменения данных в терминах строк таблиц. Более высокий уровень, чем файлы

Помимо репликации, рекомендуется создавать резервные копии. Они могут потребоваться, если вдруг сервера СУБД выйдут из строя.

Функциональные и нефункциональные требования

- ПК на Unix с 8ГБ ОЗУ или виртуальная машина с включенной Nested Virtualization.
- Созданный аккаунт на GitHub - https://github.com/
- Если Вы находитесь в России, для корректной работы Вам может потребоваться VPN.

Предварительно установленное и настроенное следующее ПО:

- Hashicorp Vagrant (https://www.vagrantup.com/downloads)
- Oracle VirtualBox (https://www.virtualbox.org/wiki/Linux_Downloads).
- Любой редактор кода, например Visual Studio Code, Atom и т.д.

Инструкция по выполнению домашнего задания

Все дальнейшие действия были проверены при использовании Vagrant 2.3.1, VirtualBox v6.1.32. В лабораторной работе используются Vagrant boxes с CentOS 8 Stream. Серьёзные отступления от этой конфигурации могут потребовать адаптации с вашей стороны.

Создадим Vagrantfile, в котором будут указаны параметры наших ВМ:

```
# Описание параметров ВМ
MACHINES = {
  # Имя DV "pam"
  :node1 => {
      # VM box
      :box_name => "centos/stream8",
      # Имя VM
      :vm_name => "node1",
      # Количество ядер CPU
      :cpus => 2,
      # Указываем количество ОЗУ (В Мегабайтах)
      :memory => 1024,
      # Указываем IP-адрес для ВМ
      :ip => "192.168.57.11",
  },
  :node2 => {
      :box_name => "centos/stream8",
      :vm_name => "node2",
      :cpus => 2,
      :memory => 1024,
      :ip => "192.168.57.12",

  },
  :barman => {

      :box_name => "centos/stream8",

      :vm_name => "barman",
      :cpus => 1,
      :memory => 1024,
      :ip => "192.168.57.13",
```

```
  },

}

Vagrant.configure("2") do |config|

  MACHINES.each do |boxname, boxconfig|

    config.vm.define boxname do |box|

      box.vm.box = boxconfig[:box_name]
      box.vm.host_name = boxconfig[:vm_name]
      box.vm.network "private_network", ip: boxconfig[:ip]
      box.vm.provider "virtualbox" do |v|
        v.memory = boxconfig[:memory]
        v.cpus = boxconfig[:cpus]
      end

      # Запуск ansible-playbook
      if boxconfig[:vm_name] == "barman"
       box.vm.provision "ansible" do |ansible|
        ansible.playbook = "ansible/provision.yml"
        ansible.inventory_path = "ansible/hosts"
        ansible.host_key_checking = "false"
        ansible.limit = "all"
       end
      end
    end
  end
end
```

После создания Vagrantfile запустим наши ВМ командой *vagrant up*. Будет создано три виртуальных машины.

На всех хостах предварительно должны быть выключены firewalld и SElinux:

- Отключаем службу firewalld: systemctl stop firewalld
- Удаляем службу из автозагрузки: systemctl disable firewalld
- Отключаем SElinux: setenforce 0
- Правим параметр SELINUX=disabled в файле /etc/selinux/config

Для удобства на все хосты можно установить текстовый редактор vim и утилиту telnet:
yum install -y vim telnet

Команды должны выполняться от root-пользователя
Для перехода в root-пользователя вводим sudo -i

Настройка hot_standby репликации с использованием слотов

Перед настройкой репликации необходимо установить postgres-server на хосты node1 и node2:
1) Добавим postgres репозиторий: sudo dnf install -y
https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm

2) Исключаем старый postgresql модуль: yum -qy module disable postgresql
3) Устанавливаем postgresql-server 14: yum install -y postgresql14-server
4) Выполняем инициализацию кластера: sudo /usr/pgsql-14/bin/postgresql-14-setup initdb
5) Запускаем postgresql-server: systemctl start postgresql-14
6) Добавляем postgresql-server в автозагрузку: systemctl enable postgresql-14

Далее приступаем к настройке репликации:
На хосте node1:
1) Заходим в psql:
[vagrant@node1 ~]$ sudo -u postgres psql
could not change directory to "/home/vagrant": Permission denied
psql (14.5)
Type "help" for help.

postgres=#

2) В psql создаём пользователя replicator с правами репликации и паролем «Otus2022!»
CREATE USER replicator WITH REPLICATION Encrypted PASSWORD 'Otus2022!';

3) В файле /var/lib/pgsql/14/data/postgresql.conf указываем следующие параметры:
#Указываем ip-адреса, на которых postgres будет слушать трафик на порту 5432 (параметр port)
listen_addresses = 'localhost, 192.168.57.11'
#Указываем порт порт postgres
port = 5432
#Устанавливаем максимально 100 одновременных подключений
max_connections = 100
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_rotation_age = 1d
log_rotation_size = 0
log_truncate_on_rotation = on
max_wal_size = 1GB
min_wal_size = 80MB
log_line_prefix = '%m [%p] '
#Указываем часовой пояс для Москвы
log_timezone = 'UTC+3'
timezone = 'UTC+3'
datestyle = 'iso, mdy'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
#можно или нет подключаться к postgresql для выполнения запросов в процессе восстановления;
hot_standby = on
#Включаем репликацию
wal_level = replica
#Количество планируемых слейвов
max_wal_senders = 3
#Максимальное количество слотов репликации
max_replication_slots = 3
#будет ли сервер slave сообщать мастеру о запросах, которые он выполняет.
hot_standby_feedback = on

```
#Включаем использование зашифрованных паролей
password_encryption = scram-sha-256
```

4) Настраиваем параметры подключения в файле /var/lib/pgsql/14/data/pg_hba.conf:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
host    replication replication    192.168.57.11/32        scram-sha-256
host    replication replication    192.168.57.12/32        scram-sha-256
```

Две последние строки в файле разрешают репликацию пользователю replication.

5) Перезапускаем postgresql-server: systemctl restart postgresql-14.service

На хосте node2:
1) Останавливаем postgresql-server: systemctl stop postgresql-14.service
2) С помощью утилиты pg_basebackup копируем данные с node1:
pg_basebackup -h 192.168.57.11 -U replication -p 5432 -D /var/lib/pgsql/14/data/ -R -P
3) В файле var/lib/pgsql/14/data/postgresql.conf меняем параметр:
listen_addresses = 'localhost, 192.168.57.12'
4) Запускаем службу postgresql-server: systemctl start postgresql-14.service

Проверка репликации:
На хосте node1 в psql создадим базу otus_test и выведем список БД:

```
postgres=# CREATE DATABASE otus_test;
CREATE DATABASE
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 otus_test | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
(4 rows)
postgres=#
```

На хосте node2 также в psql также проверим список БД (команда \l), в списке БД должна появится БД otus_test.

Также можно проверить репликацию другим способом:
На хосте node1 в psql вводим команду: select * from pg_stat_replication;
На хосте node2 в psql вводим команду: select * from pg_stat_wal_receiver;
Вывод обеих команд должен быть не пустым.

На этом настройка репликации завершена.

В случае выхода из строя master-хоста (node1), на slave-сервере (node2) в psql необхоимо выполнить команду select pg_promote();
Также можно создать триггер-файл. Если в дальнейшем хост node1 заработает корректно, то для восстановления его работы (как master-сервера) необходимо:

- Настроить сервер node1 как slave-сервер
- Также с помощью команды select pg_promote(); перевести режим его работы в master

Настройка hot_standby репликации с использованием слотов с помощью ansible

В каталоге с нашей лабораторной работой создадим каталог Ansible: *mkdir ansible*
В каталоге ansible создадим файл **hosts** со следующими параметрами:

```
[servers]
node1 ansible_host=192.168.57.11 ansible_user=vagrant
ansible_ssh_private_key_file=./.vagrant/machines/node1/virtualbox/private_key
node2 ansible_host=192.168.57.12 ansible_user=vagrant
ansible_ssh_private_key_file=./.vagrant/machines/node2/virtualbox/private_key
barman ansible_host=192.168.57.13 ansible_user=vagrant
ansible_ssh_private_key_file=./.vagrant/machines/barman/virtualbox/private_key
```

Файл содержит группу servers в которой прописаны 3 хоста:

- node1
- node2
- barman

Также указаны и ip-адреса, имя пользователя от которого будет логин и ssh-ключ.

Далее создадим файл **provision.yml** в котором непосредственно будет выполняться настройка клиентов:

```
- name: Postgres
  hosts: all
  become: yes
  tasks:
  #Устанавливаем vim и telnet (для более удобной работы с хостами)
  - name: install base tools
    dnf:
      name:
        - vim
        - telnet
      state: present
      update_cache: true


#Запуск ролей install_postgres и postgres_replication на хостах node1 и node2
- name: install postgres 14 and set up replication
  hosts: node1,node2
  become: yes
  roles:
   - install_postgres
```

```
  - postgres_replication


#Запуск роли install_barman на всех хостах
- name: set up backup
  hosts: all
  become: yes
  roles:
   - install_barman
```

В файле **provision.yml** мы видим модули запуска ролей. Для того, чтобы создать роль, находясь в каталоге ansible, нужно ввести команду: ansible-galaxy init <имя_роли>
После выполнения команды в каталоге ansible будет создан каталог с именем роли, в нём автоматически будут созданны каталоги. Рассмотрим каталоги, которые будут использоваться в нашей методичке:

- **defaults** — каталог, в котором могут находится дефолтные значения
- **tasks** — каталог, в котором содержатся ansible плейбуки, первым всегда запускается плейбук с именем main.yml
- **templates** — каталог в котором содержатся черновики, которые будут использоваться в настройке конфигурации
- README.md — файл-руководство, в нём обычно указана важная информация о роли и/или пример запуска
  Более подробно изучить информацию о ролях можно в документации Ansible.

Для настройки репликации у нас используются 2 роли:

- install_postgres
- postgres_replication

Создадим роль install_postgres: ansible-galaxy init install_postgres

В каталоге tasks создаём файл main.yml и добавляем в него следующее содержимое:

```
---
# Отключаем firewalld и удаляем его из автозагрузки
  - name: disable firewalld service
    service:
      name: firewalld
      state: stopped
      enabled: false

  # Отключаем SElinux
  - name: Disable SELinux
    selinux:
      state: disabled

  # Отключаем SElinux после перезагрузки
  - name: Ensure SELinux is set to disable mode
    lineinfile:
      path: /etc/selinux/config
      regexp: '^SELINUX='
      line: SELINUX=disabled

  # Добавляем репозиторий postgres
  - name: install repo
    dnf:
      name:
'https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm'
```

```
    state: present

  # Отключаем старый модуль
  - name: disable old postgresql module
    shell: dnf -qy module disable postgresql

  # Устанавливаем postgresql14-server
  - name: install postgresql-server 14
    dnf:
      name: postgresql14-server
      state: present
      update_cache: true

  # Проверяем, что postgres на хосте ещё не инициализирован
  - name: check init
    stat:
      path: /var/lib/pgsql/14/data/pg_stat
    register: stat_result

  # Выполняем инициализацию postgres
  - name: initialization setup
    shell: sudo /usr/pgsql-14/bin/postgresql-14-setup initdb
    when: not stat_result.stat.exists

  # Запускаем postgresql-14
  - name: enable and start service
    service:
      name: postgresql-14
      state: started
      enabled: true
```

Далее, создаём роль **postgres_replication:** ansible-galaxy init postgres_replication

В каталоге defaults создаём файл main.yml со следующими переменными:

```
---
# defaults file for postgres_replication
replicator_password: 'Otus2022!'
master_ip: '192.168.57.11'
slave_ip: '192.168.57.12'
```

Данные переменные требуются для темплейтов и выполнения некоторых модулей.

В каталоге templates создадим файлы:

- pg_hba.conf.j2
- postgresql.conf.j2

Содержимое файлов было представлено ранее в методичке, так как мы используем темплейты, вместо ip-адресов будут указанные переменные из файла **defaults/main.yml**

В каталоге tasks создаём файл main.yml со следующим содержимым:

```
---
# Установка python-пакетов для модулей psql
  - name: install base tools
    dnf:
      name:
        - python3-pexpect.noarch
        - python3-psycopg2
```

```
    state: present
    update_cache: true

#CREATE USER replicator WITH REPLICATION Encrypted PASSWORD 'Otus2022!';
  - name: Create replicator user
    become_user: postgres
    postgresql_user:
      name: replication
      password: '{{ replicator_password }}'
      role_attr_flags: REPLICATION
    ignore_errors: true
    when: (ansible_hostname == "node1")


  #Остановливаем postgresql-14 на хосте node2
  - name: stop postgresql-server on node2
    service:
      name: postgresql-14
      state: stopped
    when: (ansible_hostname == "node2")

  #Копиуем конфигурационный файл postgresql.conf
  - name: copy postgresql.conf
    template:
      src: postgresql.conf.j2
      dest: /var/lib/pgsql/14/data/postgresql.conf
      owner: postgres
      group: postgres
      mode: '0600'
    when: (ansible_hostname == "node1")

  #Копиуем конфигурационный файл pg_hba.conf
  - name: copy pg_hba.conf
    template:
      src: pg_hba.conf.j2
      dest: /var/lib/pgsql/14/data/pg_hba.conf
      owner: postgres
      group: postgres
      mode: '0600'
    when: (ansible_hostname == "node1")

   #Перезапускаем службу  postgresql-14
  - name: restart postgresql-server on node1
    service:
      name: postgresql-14
      state: restarted
    when: (ansible_hostname == "node1")

  #Удаляем содержимое каталога /var/lib/pgsql/14/data/
  - name: Remove files from data catalog
    file:
      path: /var/lib/pgsql/14/data/
      state: absent
    when: (ansible_hostname == "node2")

  #Копируем данные с node1 на node2
  - name: copy files from master to slave
    become_user: postgres
    expect:
```

```
    command: 'pg_basebackup -h {{ master_ip }} -U  replication -p 5432 -D /var/lib/pgsql/14/data/ -R -P'
    responses:
      '.*Password*': "{{ replicator_password }}"
  when: (ansible_hostname == "node2")


 #Копируем конфигурационный файл postgresql.conf
 - name: copy postgresql.conf
   template:
     src: postgresql.conf.j2
     dest: /var/lib/pgsql/14/data/postgresql.conf
     owner: postgres
     group: postgres
     mode: '0600'
   when: (ansible_hostname == "node2")


 #Копируем конфигурационный файл pg_hba.conf
 - name: copy pg_hba.conf
   template:
     src: pg_hba.conf.j2
     dest: /var/lib/pgsql/14/data/pg_hba.conf
     owner: postgres
     group: postgres
     mode: '0600'
   when: (ansible_hostname == "node2")


 #Запускаем службу postgresql-14 на хосте node2
 - name: start postgresql-server on node2
   service:
     name: postgresql-14
     state: started
   when: (ansible_hostname == "node2")
```

Настройка резервного копирования

Настраивать резервное копирование мы будем с помощью утилиты Barman. В документации Barman рекомендуется разворачивать Barman на отдельном сервере. В этом случае потребуется настроить доступы между серверами по SSH-ключам. В данном руководстве мы будем разворачивать Barman на отдельном хосте, если Вам удобнее, для теста можно будет развернуть Barman на хосте node1.

На хостах node1 и node2 необходимо установить утилиту barman-cli, для этого:

1) Устанавливаем epel-release: dnf install epel-release -y
2) Устанавливаем barman-cli: dnf install barman-cli

На хосте barman выполняем следующие настройки:

1) *предаварительно отключаем firewalld и SElinux
2) Устанавливаем epel-release: dnf install epel-release -y
3) Добавим postgres репозиторий: sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
4) Исключаем старый postgresql модуль: yum -qy module disable postgresql
5) Устанавливаем пакеты barman и postgresql-client: dnf install barman-cli barman postgresql14
6) Переходим в пользователя barman и генерируем ssh-ключ:

```
su barman
cd
ssh-keygen -t rsa -b 4096
```

На хосте node1:

7) Переходим в пользователя postgres и генерируем ssh-ключ:
```
su postgres
cd
ssh-keygen -t rsa -b 4096
```

8) После генерации ключа, выводим содержимое файла ~/.ssh/id_rsa.pub:
```
cat ~/.ssh/id_rsa.pub
```
Копируем содержимое файла на сервер **barman** в файл **/var/lib/barman/.ssh/authorized_keys**

9) В psql создаём пользователя barman с правами суперпользователя:
```
CREATE USER barman WITH REPLICATION Encrypted PASSWORD 'Otus2022!';
```

10) В файл /var/lib/pgsql/14/data/pg_hba.conf добавляем разрешения для пользователя barman:
```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
host    replication replication   192.168.57.11/32       scram-sha-256
host    replication replication   192.168.57.12/32       scram-sha-256
host    all             barman      192.168.57.13/32        scram-sha-256
host    replication   barman      192.168.57.13/32      scram-sha-256
```

11) Перезапускаем службу postgresql-14: `systemctl restart postgresql-14`
12) В psql создадим тестовую базу otus: `CREATE DATABASE otus;`
13) В базе создаём таблицу test в базе otus:
```
\c otus;
CREATE TABLE test (id int, name varchar(30));
INSERT INTO test VALUES (1, alex);
```

На хосте barman:

14) После генерации ключа, выводим содержимое файла ~/.ssh/id_rsa.pub:
```
cat ~/.ssh/id_rsa.pub
```
Копируем содержимое файла на сервер **postgres** в файл **/var/lib/pgsql/.ssh/authorized_keys**

15) Находясь в пользователе barman создаём файл ~/.pgpass со следующим содержимым:

```
192.168.57.11:5432:*:barman:Otus2022!
```

В данном файле указываются реквизиты доступа для postgres. Через знак двоеточия пишутся следующие параметры:

- ip-адрес
- порт postgres
- имя БД (* означает подключение к любой БД)
- имя пользователя
- пароль пользователя

Файл должен быть с правами 600, владелец файла barman.

16) После создания postgres-пользователя barman необходимо проверить, что права для пользователя настроены корректно:

Проверяем возможность подключения к postgres-серверу:

```
bash-4.4$ psql -h 192.168.57.11 -U barman -d postgres
psql (14.5)
Type "help" for help.

postgres=# \q
bash-4.4$
```

Проверяем репликацию:

```
bash-4.4$ psql -h 192.168.57.11 -U barman -c "IDENTIFY_SYSTEM" replication=1
      systemid       | timeline |  xlogpos  | dbname
      --------------------+----------+-----------+--------
       7151863316617733050 |        1 | 0/4000E78 |
      (1 row)
bash-4.4$
```

17) Создаём файл /etc/barman.conf со следующим содержимым
Владельцем файла должен быть пользователь barman.

```
[barman]
#Указываем каталог, в котором будут храниться бекапы
barman_home = /var/lib/barman
#Указываем каталог, в котором будут храниться файлы конфигурации бекапов
configuration_files_directory = /etc/barman.d
#пользователь, от которого будет запускаться barman
barman_user = barman
#расположение файла с логами
log_file = /var/log/barman/barman.log
#Используемый тип сжатия
compression = gzip
#Используемый метод бекапа
backup_method = rsync
archiver = on
retention_policy = REDUNDANCY 3
immediate_checkpoint = true
#Глубина архива
last_backup_maximum_age = 4 DAYS
minimum_redundancy = 1
```

18) Создаём файл /etc/barman.d/node1.conf со следующим содержимым
Владельцем файла должен быть пользователь barman.

```
[node1]
#Описание задания
description = "backup node1"
#Команда подключения к хосту node1
ssh_command = ssh postgres@192.168.57.11
#Команда для подключения к postgres-серверу
conninfo = host=192.168.57.11 user=barman port=5432 dbname=postgres
retention_policy_mode = auto
retention_policy = RECOVERY WINDOW OF 7 days
wal_retention_policy = main
streaming_archiver=on
#Указание префикса, который будет использоваться как $PATH на хосте node1
path_prefix = /usr/pgsql-14/bin/
#настройки слота
create_slot = auto
slot_name = node1
#Команда для потоковой передачи от postgres-сервера
streaming_conninfo = host=192.168.57.11 user=barman
#Тип выполняемого бекапа
backup_method = postgres
archiver = off
```

19) На этом настройка бекапа завершена. Теперь проверим работу barman:

```
bash-4.4$ barman switch-wal node1
The WAL file 000000010000000000000005 has been closed on server 'node1'
bash-4.4$ barman cron
Starting WAL archiving for server node1
bash-4.4$ barman check node1
Server node1:
        PostgreSQL: OK
        superuser or standard user with backup privileges: OK
        PostgreSQL streaming: OK
        wal_level: OK
        replication slot: OK
        directories: OK
        retention policy settings: OK
        backup maximum age: FAILED (interval provided: 4 days, latest backup age: No available
backups)
        backup minimum size: OK (0 B)
        wal maximum age: OK (no last_wal_maximum_age provided)
        wal size: OK (0 B)
        compression settings: OK
        failed backups: OK (there are 0 failed backups)
        minimum redundancy requirements: FAILED (have 0 backups, expected at least 1)
        pg_basebackup: OK
        pg_basebackup compatible: OK
        pg_basebackup supports tablespaces mapping: OK
        systemid coherence: OK (no system Id stored on disk)
        pg_receivexlog: OK
        pg_receivexlog compatible: OK
        receive-wal running: OK
        archiver errors: OK
bash-4.4$
```

Если во всех пунктах, кроме выделенных будет OK, значит бекап отработает корректно. Если в остальных пунктах вы видите FAILED, то бекап у вас не запустится. Требуется посмотреть в логах, в чём может быть проблема...

20) После этого запускаем резервную копию:

```
bash-4.4$ barman backup node1
Starting backup using postgres method for server node1 in
/var/lib/barman/node1/base/20221008T010731
Backup start at LSN: 0/6000148 (000000010000000000000006, 00000148)
Starting backup copy via pg_basebackup for 20221008T010731
Copy done (time: 1 second)
Finalising the backup.
This is the first backup for server node1
WAL segments preceding the current backup have been found:
        000000010000000000000004 from server node1 has been removed
        000000010000000000000005 from server node1 has been removed
Backup size: 41.8 MiB
Backup end at LSN: 0/8000000 (000000010000000000000007, 00000000)
Backup completed (start time: 2022-10-08 01:07:31.939958, elapsed time: 1 second)
Processing xlog segments from streaming for node1
        000000010000000000000006
        000000010000000000000007
bash-4.4$
```

На этом процесс настройки бекапа закончен, для удобства команду barman backup node1 требуется добавить в crontab.

Проверка восстановления из бекапов:

На хосте node1 в psql удаляем базы Otus:

```
bash-4.4$ psql
psql (14.5)
Type "help" for help.

postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 otus      | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 otus_test | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(5 rows)

postgres=# DROP DATABASE otus;
DROP DATABASE
postgres=#
postgres=# DROP DATABASE otus_test;
DROP DATABASE
postgres=# \l
                                 List of databases
```

```
   Name    | Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(3 rows)

postgres=#
```

Далее на хосте barman запустим восстановление:

```
bash-4.4$ barman list-backup node1
node1 20221008T010731 - Fri Oct  7 22:07:50 2022 - Size: 41.8 MiB - WAL Size: 0 B
bash-4.4$
bash-4.4$ barman recover node1 20221008T010731 /var/lib/pgsql/14/data/ --remote-ssh-comman
"ssh postgres@192.168.57.11"
The authenticity of host '192.168.57.11 (192.168.57.11)' can't be established.
ECDSA key fingerprint is SHA256:NDadubkUsCyw+X3o+WPVePaWJ+5Bl99wfYw5/JdrNYs.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Starting remote restore for server node1 using backup 20221008T010731
Destination directory: /var/lib/pgsql/14/data/
Remote command: ssh postgres@192.168.57.11
Copying the base backup.
Copying required WAL segments.
Generating archive status files
Identify dangerous settings in destination directory.

Recovery completed (start time: 2022-10-08 01:20:24.864427+00:00, elapsed time: 4 seconds)
Your PostgreSQL server has been successfully prepared for recovery!
bash-4.4$
```

Далее на хосте node1 потребуется перезапустить postgresql-сервер и снова проверить список БД. Базы otus должны вернуться обратно…

Настройка резервного копирования с помощью Ansible

Создаём роль **install_barman:** ansible-galaxy init install_barman

В каталоге defaults создаём файл main.yml со следующими переменными:

```
---
# defaults file for install_barman
master_ip: '192.168.57.11'
master_user: 'postgres'
barman_ip: '192.168.57.13'
barman_user: 'barman'
barman_user_password: 'Otus2022!'
```

Данные переменные требуются для темплейтов и выполнения некоторых модулей.

В каталоге templates создадим файлы:

- .pgpass.j2
- barman.conf.j2
- node1.conf.j2

Содержимое файлов было представлено ранее в методичке, так как мы используем темплейты, вместо ip-адресов и имён пользователей будут указанные переменные из файла **defaults/main.yml**

В каталоге tasks создаём файл main.yml со следующим содержимым:

```
---
# Установка необходимых пакетов для работы с postgres и пользователями
 - name: install base tools
   dnf:
     name:
       - python3-pexpect.noarch
       - python3-psycopg2
       - bash-completion
       - wget
     state: present
     update_cache: true


 # Отключаем firewalld и удаляем его из автозагрузки
 - name: disable firewalld service
   service:
     name: firewalld
     state: stopped
     enabled: false
   when: (ansible_hostname == "barman")


  # Отключаем SElinux
 - name: Disable SELinux
   selinux:
     state: disabled
   when: (ansible_hostname == "barman")

 - name: Ensure SELinux is set to disable mode
   lineinfile:
     path: /etc/selinux/config
     regexp: '^SELINUX='
     line: SELINUX=disabled
   when: (ansible_hostname == "barman")


 # Добавляем postgres репозиторий на хост barman
 - name: install repo
   dnf:
     name:
'https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm'
     state: present
   when: (ansible_hostname == "barman")

 #  Отключение старого postgres-модуля
 - name: disable old postgresql module
   shell: dnf -qy module disable postgresql
   when: (ansible_hostname == "barman")
```

```
  #  Установка EPEL-release
  - name: install epel-release
    dnf:
      name:
        - epel-release
      state: present
      update_cache: true


  #  Установка пакетов barman и stgresql-client на сервер barman
  - name: install barman and postgresql packages on barman
    dnf:
      name:
        - barman
        - barman-cli
        - postgresql14
      state: present
      update_cache: true
    when: (ansible_hostname == "barman")


  #  Установка пакетов barman-cli на серверах node1 и node2
  - name: install barman-cli and postgresql packages on nodes
    dnf:
      name:
        - barman-cli
      state: present
      update_cache: true
    when: (ansible_hostname != "barman")

#  Генерируем SSH-ключ для пользователя postgres на хосте node1
  - name: generate SSH key for postgres
    user:
      name: postgres
      generate_ssh_key: yes
      ssh_key_type: rsa
      ssh_key_bits: 4096
      force: no
    when: (ansible_hostname == "node1")


#  Генерируем SSH-ключ для пользователя barman на хосте barman
  - name: generate SSH key for barman
    user:
      name: barman
      uid: 994
      shell: /bin/bash
      generate_ssh_key: yes
      ssh_key_type: rsa
      ssh_key_bits: 4096
      force: no
    when: (ansible_hostname == "barman")

  #  Забираем содержимое открытого ключа postgres с хоста node1
  - name: fetch all public ssh keys node1
    shell: cat /var/lib/pgsql/.ssh/id_rsa.pub
    register: ssh_keys
    when: (ansible_hostname == "node1")
```

```
#  Копируем ключ с barman на node1
- name: transfer public key to barman
  delegate_to: barman
  authorized_key:
    key: "{{ ssh_keys.stdout }}"
    comment: "{{ansible_hostname}}"
    user: barman
  when: (ansible_hostname == "node1")

#  Забираем содержимое открытого ключа barman с хоста barman
- name: fetch all public ssh keys barman
  shell: cat /var/lib/barman/.ssh/id_rsa.pub
  register: ssh_keys
  when: (ansible_hostname == "barman")

#  Копируем ключ с node1 на barman
- name: transfer public key to barman
  delegate_to: node1
  authorized_key:
    key: "{{ ssh_keys.stdout }}"
    comment: "{{ansible_hostname}}"
    user: postgres
  when: (ansible_hostname == "barman")

#CREATE USER barman SUPERUSER;
- name: Create barman user
  become_user: postgres
  postgresql_user:
    name: barman
    password: '{{ barman_user_password }}'
    role_attr_flags: SUPERUSER
  ignore_errors: true
  when: (ansible_hostname == "node1")

 # Добавляем разрешения для поключения с хоста barman
- name: Add permission for barman
  lineinfile:
    path: /var/lib/pgsql/14/data/pg_hba.conf
    line: 'host    all   {{ barman_user }}    {{ barman_ip }}/32    scram-sha-256'
  when: (ansible_hostname == "node1") or
        (ansible_hostname == "node2")

# Добавляем разрешения для подключения с хоста barman
- name: Add permission for barman
  lineinfile:
    path: /var/lib/pgsql/14/data/pg_hba.conf
    line: 'host    replication   {{ barman_user }}    {{ barman_ip }}/32    scram-sha-256'
  when: (ansible_hostname == "node1") or
        (ansible_hostname == "node2")

# Перезагружаем службу postgresql-server
- name: restart postgresql-server on node1
  service:
    name: postgresql-14
    state: restarted
  when: (ansible_hostname == "node1")
```

```
# Создаём БД otus;
- name: Create DB for backup
  become_user: postgres
  postgresql_db:
    name: otus
    encoding: UTF-8
    template: template0
    state: present
  when: (ansible_hostname == "node1")

# Создаём таблицу test1 в БД otus;
- name: Add tables to otus_backup
  become_user: postgres
  postgresql_table:
    db: otus
    name: test1
    state: present
  when: (ansible_hostname == "node1")


# Копируем файл .pgpass
- name: copy .pgpass
  template:
    src: .pgpass.j2
    dest: /var/lib/barman/.pgpass
    owner: barman
    group: barman
    mode: '0600'
  when: (ansible_hostname == "barman")


# Копируем файл barman.conf
- name: copy barman.conf
  template:
    src: barman.conf.j2
    dest: /etc/barman.conf
    owner: barman
    group: barman
    mode: '0755'
  when: (ansible_hostname == "barman")


# Копируем файл node1.conf
- name: copy node1.conf
  template:
    src: node1.conf.j2
    dest: /etc/barman.d/node1.conf
    owner: barman
    group: barman
    mode: '0755'
  when: (ansible_hostname == "barman")


- name: barman switch-wal node1
  become_user: barman
  shell: barman switch-wal node1
  when: (ansible_hostname == "barman")

- name: barman cron
  become_user: barman
```

```
shell: barman cron
when: (ansible_hostname == "barman")
```

На этом настройка бекапа с помощью Ansible завершена. Проверить работу бекапа можно также, как описано в предыдущем пункте руководства.

Критерии оценивания

Статус «Принято» ставится при выполнении следующих условий:

1. Ссылка на репозиторий GitHub.
2. Vagrantfile, который будет разворачивать виртуальные машины
3. Плейбук Ansible
4. Конфигурационные файлы postgresql.conf, pg_hba.conf и recovery.conf,
5. Конфиг barman, либо скрипт резервного копирования.
6. Документация по каждому заданию:

Создайте файл README.md и снабдите его следующей информацией:
- название выполняемого задания;
- текст задания;
- описание команд и их вывод;
- особенности проектирования и реализации решения,
- заметки, если считаете, что имеет смысл их зафиксировать в репозитории.

Рекомендуемые источники

- Статья «Настройка потоковой репликации PostgreSQL» - https://www.dmosk.ru/miniinstruktions.php?mini=postgresql-replication
- Статья «Настройка streaming Master-Slave репликации в PosgtreSQL 12 и мониторинг» - https://it-lux.ru/streaming-replication-master-slave-posgtresql-12/
- Статья «Barman. менеджер бэкапов для серверов PostgreSQL» - https://sidmid.ru/barman-%D0%BC%D0%B5%D0%BD%D0%B5%D0%B4%D0%B6%D0%B5%D1%80-%D0%B1%D1%8D%D0%BA%D0%B0%D0%BF%D0%BE%D0%B2-%D0%B4%D0%BB%D1%8F-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BE%D0%B2-postgresql/
- Статья «Implement backup with Barman» - https://medium.com/@kiwiv/implement-backup-with-barman-bb0b44af71f9
- Статья о простых операциях с данными в Postgres - https://metanit.com/sql/postgresql/3.1.php
- Официальная документация Barman - https://docs.pgbarman.org/release/3.1.0/#introduction
- Статья «The password file» - https://www.postgresql.org/docs/current/libpq-pgpass.html
- Статья Roles in Ansible - https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html