SRE – POSTGRESQL WITH CONNECTION POOLING

OVERVIEW

Our goal is to automate the installation and setup of PostgreSQL server on container with connection pooler ideally pgBouncer.

- pgBouncer should authenticate user login without depending on auth file.
- Incoming connections should go through pgBouncer layer
- A database should be created in PostgreSQL by loading a SQL script from GitHub.
- Additionally monitoring the pgBouncer if possible

TOOLS & TECHNOLOGIES

- Ansible
- Docker
- PostgreSQL
- pgBouncer
- Flask

ANSIBLE

Ansible is a versatile and open-source automation tool that simplifies the management and configuration of computer systems. It uses a declarative approach, allowing users to define the desired state of their systems through human readable YAML files called playbooks. Ansible's push-based model enables the execution of tasks remotely over SSH, making it easy to manage both Linux and Windows systems without the need for a central server or agent installation. With its extensive collection of built-in modules and ability to scale and orchestrate tasks, Ansible empowers organizations to automate software installations, configuration changes, and application deployments, streamlining system administration and enhancing operational efficiency.

DOCKER

Docker is an open-source platform that enables the creation, deployment, and running of applications using containerization. Containers are lightweight, isolated environments that package an application and its dependencies, allowing them to run consistently across different environments. With Docker, developers can easily build Docker images, which contain all the necessary components of an application, such as the code, runtime, libraries, and system tools. These images can be versioned, shared, and stored in a registry. Docker containers can then be created from these images, providing a portable and consistent runtime environment for the application. Docker simplifies application deployment by abstracting away the differences between development, testing, and production environments, ensuring that applications run reliably and consistently regardless of the underlying infrastructure.

POSTGRESQL

PostgreSQL is a powerful and open-source relational database management system (RDBMS) known for its robustness, scalability, and extensibility. It provides a reliable and efficient way to store, retrieve, and manipulate structured data, making it suitable for a wide range of applications. PostgreSQL offers a rich set of features, including support for advanced SQL queries, data integrity constraints, transactional processing, and multi-version concurrency control. It also provides various data types, indexing options, and extensibility through user-defined functions, stored procedures, and custom extensions. With its community-driven development and active user community, PostgreSQL continues to evolve, making it a popular choice for both small-scale and enterprise-level database needs.

PGBOUNCER

PgBouncer is a lightweight and open-source connection pooler for PostgreSQL databases. It sits between client applications and the PostgreSQL server, acting as an intermediary and efficiently managing database connections. PgBouncer helps improve the performance and scalability of PostgreSQL by reusing database connections, reducing overhead and resource consumption. It supports features such as connection pooling, transaction pooling, and statement pooling, allowing multiple client connections to share a smaller set of PostgreSQL connections. Additionally, PgBouncer provides advanced connection management options, load balancing capabilities, and configuration flexibility, making it a valuable tool for optimizing and efficiently managing database connections in PostgreSQL deployments.

FLASK

Flask is a lightweight and flexible web framework for Python that simplifies the development of web applications. It provides a minimalistic yet powerful set of tools and libraries, allowing developers to quickly build scalable and robust web services. Flask follows a "micro" design philosophy, focusing on simplicity and extensibility, while still providing essential features such as URL routing, request handling, and template rendering. With its modular architecture, Flask allows developers to choose and integrate additional libraries based on their specific needs, making it highly customizable. Whether building simple APIs or complex web applications, Flask offers a straightforward and elegant solution for Python developers to create efficient and maintainable web services.

DEVELOPMENT PROCEDURE

Everything developed and configured in Ubuntu 22.04.2 LTS

STEP-1

Install Ansible & Docker in host machine.

Ansible Installation:

```
sudo apt-add-repository ppa:ansible/ansible
sudo apt update
sudo apt install ansible
```

Docker Installation:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
apt-cache policy docker-ce
sudo apt install docker-ce
```

Most IDE works better. I preferred Visual Studio Code.

STEP-2

Writing Docker File to load and install required images

Filename: *Dockerfile*

```
# Use the latest version of Ubuntu Server as the base image
FROM ubuntu:latest
# we need this :-)
ARG DEBIAN_FRONTEND=noninteractive

# Update the package lists and install any desired dependencies
RUN apt-get update
RUN apt-get install -y postgresql-14
RUN apt-get install -y openssh-server
RUN apt-get install -y python3-psycopg2
RUN apt-get install -y ufw
RUN apt-get install -y nano

# changing some postgres configs
RUN sed -i 's/#password_encryption = scram-sha-256/password_encryption = md5/'
/etc/postgresql/14/main/postgresql.conf
RUN sed -i 's/host all all 127.0.0.1\/32 scram-sha-256/host all all 127.0.0.1\/32 md5/' /etc/postgresql/14/main/pg_hba.conf

# just to make container run even exits from terminal
CMD service ssh start && tail -f /dev/null
```

Summary:

1. Building a container with ubuntu latest image from docker

2. We need some tools to administer the applications inside the docker.
    a. openssh-server for providing ssh logins
    b. psycopg2 for running some ansible module inside docker
    c. ufw for managing firewall and rules
    d. nano for editing files if required

3. Some configuration changes done to postgresql
    a. md5 password encrypted
    b. Md5 password authentication

STEP-3

Writing hosts file for Ansible

Filename: *hosts*

```
[all]

[docker_hosts]
postgresv14 docker_service_name=postgresv14
```

Summary:

Since we are using docker containers I gave a host variable **docker_service_name**

STEP-4

Writing ansible playbook for Setting up Docker

Filename: *playbook_docker.yml*

```
- name: Setup Docker
  hosts: localhost
  become: yes
  gather_facts: no

  tasks:
    - name: Build PostgreSQL Image
      community.docker.docker_image:
        build:
          path: /home/chanukyasds/ansible/dev
        name: pgimage
        source: build
      tags:
        - build_image

    - name: Run PostgreSQL Image
      docker_container:
        name: postgresv14
        image: pgimage
        state: started
        hostname: pghost
        capabilities: NET_ADMIN
        privileged: yes
        exposed_ports:
          - 22
        published_ports:
          - "5432:6432"
          - "5000:5000"
      tags:
        - run_image
```

Summary:

1. Running this playbook on localhost because it is our host machine to run docker container
2. No need of facts for this playbook
3. Building image:
    a. Using ansible docker collection we can use **community.docker.docker_image**
    b. Providing details like
        i. path: docker path
        ii. name: pgimage
        iii. Source: build

Here we can give tags, but it is completely optional.

4. Running Container:
    a. Using ansible docker collection we can use **docker_container**
    b. Providing details like
        i. name: postgresv14
        ii. image: pgimage
        iii. hostname: pghost
        iv. capabilities: NET_ADMIN (we can use for iptables etc)
        v. privileged: yes
        vi. exposed_ports: 22 (use them to access the container services)
        vii. published_ports:
            1. 5432:6432 (used to expose pgbouncer to host machine)
            2. 5000:5000 (used for flask app)

Playbook execution steps:

1. First Task **Build PostgreSQL Image,** build the docker images with the dockerfile
2. Second Task **Run PostgreSQL Image,** runs the container build by first task

Note:

- Building image will take some time depends on network speed
- We have not started postgresql server
- We can inspect the docker images and containers
- Some useful commands:
    - docker ps
    - docker images
    - docker exec –it postgresv14 /bin/bash
    - docker exec postgresv14 hostname -I

More information about ansible docker collection is here.

STEP-4

Writing playbook for Setting up PostgreSQL

Filename: *playbook_postgresql.yml*

```
- name: Setup PostgreSQL
  hosts: docker_hosts
  become: yes
  become_method: su
  gather_facts: no
  connection: docker
  handlers:
    - name: Restart PostgreSQL
      ansible.builtin.command: service postgresql restart
    - name: Reload PostgreSQL
      ansible.builtin.command: service postgresql reload

  tasks:
    - name: Start PostgreSQL
      ansible.builtin.command: service postgresql start

    - name: Ping Server
      community.postgresql.postgresql_ping:
        db: "postgres"
      become_user: postgres

      # in production we set this differently
    - name: Reset postgres Password
      become_user: postgres
      ansible.builtin.command: psql -c "ALTER ROLE postgres WITH PASSWORD 'postgres';"

    - name: Download Database Script File
      ansible.builtin.get_url:
        url: https://raw.githubusercontent.com/harryho/db-samples/master/pgsql/northwind.sql
        dest: /var/lib/postgresql/
        mode: '0644'
        owner: postgres

    - name: Load Database From Script File
      become_user: postgres
      ansible.builtin.command: "{{ item }}"
      with_items:
        - psql -c "DROP DATABASE IF EXISTS northwind;"
        - psql -c "CREATE DATABASE northwind;"
        - psql -d northwind -f /var/lib/postgresql/northwind.sql

    - name: Clean Files
      become: yes
      ansible.builtin.command: rm /var/lib/postgresql/northwind.sql
```

Summary:

1. Running this playbook on docker_hosts to configure and load database northwind
2. Handlers are used to manage postgresql service for future extensibility
3. Using the ansible postgresql collection, we can use **community.postgresql.postgresql_ping** to ping the postgresql server.
4. Using the **ansible_builtin_get_url** we can download a file from web into container.
5. Using the **ansible_builtin_command** we can run several shell commands inside container

Playbook execution steps:

1. First task **Start PostgreSQL**, will start the postgresql server inside container
2. Second task **Ping Server**, will ping the started postgresql server
3. Third task **Reset postgres Password**, Changing the postgres password (this is just for demo ideally; we change it from host machine manually)
4. Fourth task **Download Database Script File**, will download the database script into specified directory inside container
5. Fifth task **Load Database From Script File**, will load the script and executes against the created northwind database
6. Sixth task **Clean Files**, will remove the used database file

Note:

- This playbook will execute quickly
- PostgreSQL will be started
- Use docker exec postgresv14 pg_lsclusters to see postgres cluster status
- We can login into container and connect to database using 5432 port and localhost

More information on ansible postgresql collection can be found here.

More information on ansible builtin collection can be found here.

STEP-5

Writing playbook for setting up pgBouncer

I have written a group_vars file for future extensibility.

Filename: *group_vars/docker_hosts.yml*

```
template_file_location:  ../templates/pgbouncer.ini
pgbouncer_config_location: /etc/pgbouncer/pgbouncer.ini

pgbouncer_start: service pgbouncer start
pgbouncer_stop: service pgbouncer stop
pgbouncer_restart: service pgbouncer restart
```

Summary:

1. These options are used in below *playbook_pgbouncer.yml*
2. These options can be used for future playbooks to deploy pgbouncer

SQL Files to configure pgbouncer*:*

Filename: *sql/auth_function.sql*

```
CREATE SCHEMA pgbouncer AUTHORIZATION pgbouncer;
CREATE OR REPLACE FUNCTION pgbouncer.auth_function(p_usename TEXT)
RETURNS TABLE(username TEXT, password TEXT) AS
$$
BEGIN
   RAISE WARNING 'PgBouncer user authentication request: %', p_usename;
   RETURN QUERY
   SELECT  usename::TEXT,
          passwd::TEXT
   FROM pg_catalog.pg_shadow
   WHERE usename = p_usename;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

Filename: *sql/pg_bouncer_config.sql*

```
CREATE USER pgbouncer WITH PASSWORD 'pgbouncer';
COPY(SELECT usename,
          passwd
    FROM pg_shadow
    WHERE usename='pgbouncer')
     TO '/etc/pgbouncer/userlist.txt'
       WITH (FORMAT CSV, DELIMITER ' ', FORCE_QUOTE *);
```

Filename: *templates/pgbouncer.ini*

```
[databases]
* = host=127.0.0.1
[pgbouncer]
logfile = /var/log/postgresql/pgbouncer.log
pidfile = /var/run/postgresql/pgbouncer.pid
listen_addr = *
listen_port = 6432
unix_socket_dir = /var/run/postgresql
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt
auth_user = pgbouncer
auth_query = SELECT * FROM pgbouncer.auth_function($1)
ignore_startup_parameters = extra_float_digits
```

Note: pgbouncer.ini is a huge file , so for compactness I have provided the required options only in the above file.

Filename: *playbook_pgbouncer.yml*

```yaml
- name: Setup pgBouncer
  hosts: docker_hosts
  become: yes
  become_method: su
  gather_facts: no
  connection: docker
  handlers:
    - name: Start pgBouncer
      ansible.builtin.command: "{{ pgbouncer_start }}"
    - name: Restart pgBouncer
      ansible.builtin.command: "{{ pgbouncer_restart }}"
    - name: Stop pgBouncer
      ansible.builtin.command: "{{ pgbouncer_stop }}"
  tasks:
    - name: Install pgBouncer
      ansible.builtin.command: "{{ item }}"
      become: yes
      with_items:
        - apt update
        - apt install pgbouncer -y
    - name: Deploy pgbouncer.ini File
      template:
        src: "{{ template_file_location }}"
        dest: "{{ pgbouncer_config_location }}"
        owner: postgres
        group: postgres
        mode: 0644
      when: (pgbouncer_config_location is defined) and (template_file_location is defined)
    - name: Load pgBouncer Script files
      become: true
      copy:
        src: ./sql/
        dest: /var/lib/postgresql
        owner: postgres
        group: postgres
        mode: 0644
    - name: Run pgBouncer Script Files
      become_user: postgres
      ansible.builtin.command: "{{ item }}"
      with_items:
        - psql -d postgres  -f /var/lib/postgresql/pg_bouncer_config.sql
        - psql -d postgres  -f /var/lib/postgresql/auth_function.sql
        - psql -d template1 -f /var/lib/postgresql/auth_function.sql
        - psql -d northwind -f /var/lib/postgresql/auth_function.sql
      notify: Restart pgBouncer
    - name: Clean Files
      become: yes
      ansible.builtin.command: "{{ item }}"
      with_items:
        - rm /var/lib/postgresql/pg_bouncer_config.sql
        - rm /var/lib/postgresql/auth_function.sql
```

Summary:

1. Running this playbook will install and configure the pgbouncer in docker container
2. **Template** module used to copy the pgbouncer.ini file from host machine to docker container
3. **Copy** module used to load the pgbouncer authentication files into docker container
4. Using the **ansible_builtin_command** we can run several shell commands inside container

Playbook execution steps:

1. First Task **Install pgBouncer**, will install pgbouncer inside docker container
2. Second Task **Deploy pgbouncer.ini File**, will load pgbouncer.ini into pgbouncer config directory
3. Third Task **Run pgBouncer Script Files**, will execute required scripts to setup user authentication for existing and new users. Ideally this authenticates logins to future databases because we run against the template1 database too.
4. Fourth Task **Clean Files**, will remove the used sql files in the docker container

Note:

- This playbook take couple of minutes to complete
- We can connect to postgres via pgbouncer
- Pgbouncer will authenticate existing and future users
- Databases can be connected with published ports
- New Database connections also handled by pgbouncer
- Incoming coming connections will have only access to host machine ip 5432 then it will redirect to 6432 in docker

We can divert incoming connections in multiple ways.

1. Using iptables rules
2. Exposing pgbouncer port only to world
3. Using ufw before rules

STEP-6

Writing playbook for monitoring pgbouncer admin console

This is an additional feature I have developed to monitor some pgbouncer admin console commands.

Why and what made me develop this?

I feel whenever we want to monitor the pools and clients, we login to pgbouncer database as pgbouncer user and run some commands like SHOW POOLS, SHOW CLIENTS etc.

Several other ways to monitor pgbouncer some are below

1. Grafana with pgbouncer dashboard
2. Grafana with PMM client

But I feel setting Grafana and configuring needs some manual work because dashboard automation is not really working well. And our need is to view the pgbouncer admin stats only.

So I feel I can develop a simple flask app to fetch admin console commands from pgbouncer database and render those command outputs to webpage.

This flask app refreshes automatically every 5sec and fetches new stats because Grafana provides 5sec refresh to fetch fresh stats and load into dashboards.

To run this monitoring app, we need to install flask package.

Filename: *flask/app.py*

```
from flask import Flask, render_template
import subprocess
import logging

log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)


app = Flask(__name__, template_folder='/root/monitor/')

@app.route('/')
def index():
    command = "su - postgres -c \"psql -U pgbouncer -p 6432 -c '\pset footer off' -c 'show clients;' -c 'show pools' -c 'show stats' \""
    output = subprocess.check_output(command, shell=True).decode()
    return render_template('output.html', output=output)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Filename*: flask/output.html*

```
<!DOCTYPE html>
<html>
<head>
    <title>pgbouncer</title>
    <div style="text-align:center">
    <h3 text-align: center;>pgBouncer Monitor</h3>
    <h4 text-align: center;>This page will auto refresh every 5 sec.</h5>
    </div>
</head>
<script type="text/javascript">
setTimeout(function(){
   window.location.reload(1);
}, 5000);
</script>
<body onload="load()">
    <br>
    <br>
    <pre>{{ output }}</pre>
</body>
</html>
```

Both files will be copied to docker container and run by root to render the results on webpage.

Filename: *playbook_monitor.yml*

```
- name: Setup Monitor
  hosts: docker_hosts
  become: yes
  become_method: su
  gather_facts: no
  connection: docker

  tasks:
    - name: Install Flask
      ansible.builtin.command: "{{ item }}"
      become: yes
      with_items:
        - apt-get install -y pip
        - pip install flask
        - mkdir /root/monitor

    - name: Load Flask Files
      become: true
      copy:
        src: ./flask/
        dest: /root/monitor
        owner: root
        group: root
        mode: 0644

    - name: Run Flask App
      become: yes
      shell: nohup python3 /root/monitor/app.py > /tmp/log1.txt 2>&1 &

    - name: Display pgBouncer stats
      delegate_to: localhost
      become: yes
      become_method: su
      become_user: chanukyasds
      shell: nohup xdg-open http://localhost:5000 > /tmp/log2.txt 2>&1 &
```

Summary:

1. Running this playbook will build a flask app and configure monitoring for pgbouncer
2. **Copy** module used to load the flask app files into docker container
3. Using the **ansible_builtin_command** we can run several shell commands inside container

Playbook execution steps:

1. First task **Install Flask**, will install flask module in docker container
2. Second task **Load Flask Files**, will copy flask app files into docker container
3. Third task **Run Flask App**, will execute the flask app in background with nohup option
4. Fourth task **Display pgBouncer stats**, will opens the webpage containing pgbouncer stats in default browser of host machine. We can access using hostip:5000.

5. Webpage will get auto refreshed every 5 sec and fetches new client connections and pools information

Note:

- This is quite easy to implement and takes less time
- Flask app will run in background even we close and re-open browser several times
- Published port is 5000 from docker container

STEP-7

Writing playbook to execute all playbooks in a particular order

Filename: *main.yml*

```
---

 - name: Setup Docker
   ansible.builtin.import_playbook: playbook_docker.yml
   tags:
     - setup_docker

 - name: Setup PostgreSQL
   ansible.builtin.import_playbook: playbook_postgresql.yml
   tags:
     - setup_postgresql

 - name: Setup pgBouncer
   ansible.builtin.import_playbook: playbook_pgbouncer.yml
   tags:
     - setup_pgbouncer

 - name: Setup Monitor
   ansible.builtin.import_playbook: playbook_monitor.yml
   tags:
     - setup_monitor
```

Playbook execution steps:

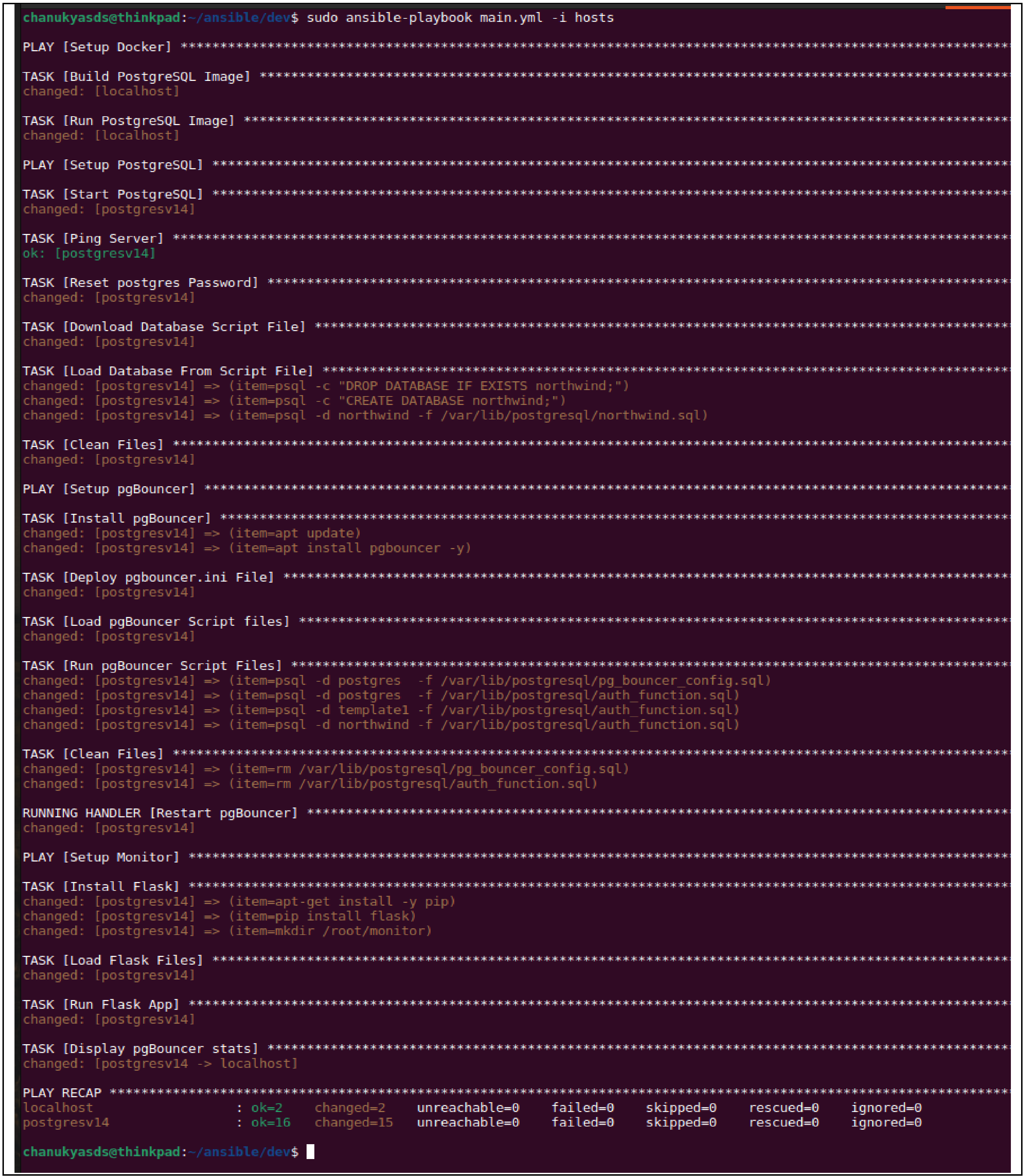
1. Running this playbook do the following:
    a. Build Image and Run Container
    b. Install and configure PostgreSQL in docker
    c. Install and configure pgBouncer in docker
    d. Install and run Flask app in docker

Note:

- Run this playbook with -i hosts option
- This playbook executes all the playbooks defined in it

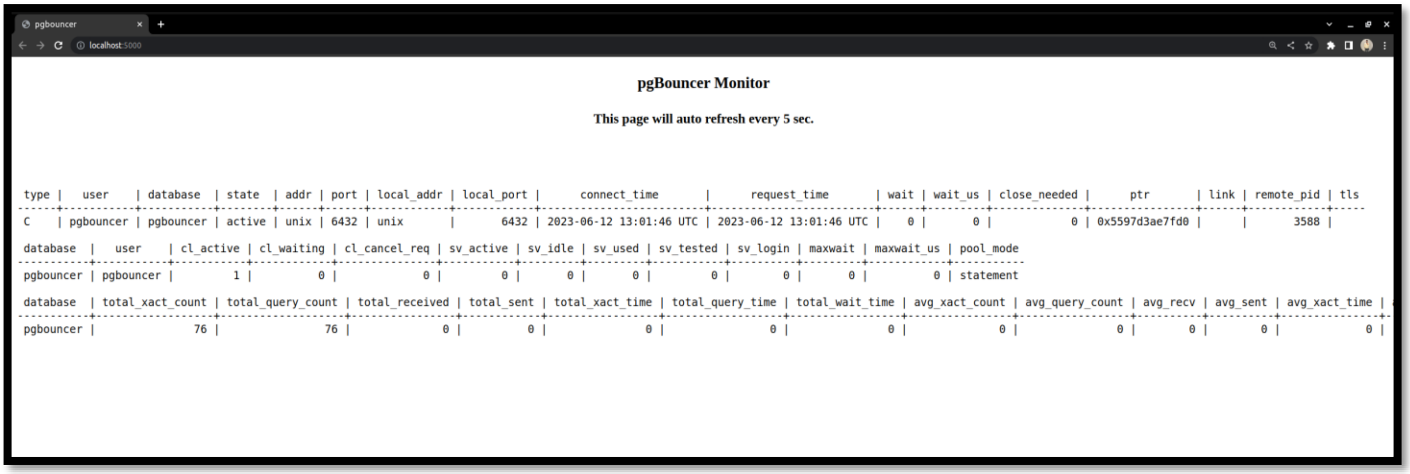
EXECUTION PROCEDURE

```
sudo ansible-playbook main.yml -i hosts
```



```
chanukyasds@thinkpad:~/ansible/dev$ sudo ansible-playbook main.yml -i hosts

PLAY [Setup Docker] **********************************************************************************************************

TASK [Build PostgreSQL Image] ************************************************************************************************
changed: [localhost]

TASK [Run PostgreSQL Image] **************************************************************************************************
changed: [localhost]

PLAY [Setup PostgreSQL] ******************************************************************************************************

TASK [Start PostgreSQL] ******************************************************************************************************
changed: [postgresv14]

TASK [Ping Server] ***********************************************************************************************************
ok: [postgresv14]

TASK [Reset postgres Password] ***********************************************************************************************
changed: [postgresv14]

TASK [Download Database Script File] *****************************************************************************************
changed: [postgresv14]

TASK [Load Database From Script File] ****************************************************************************************
changed: [postgresv14] => (item=psql -c "DROP DATABASE IF EXISTS northwind;")
changed: [postgresv14] => (item=psql -c "CREATE DATABASE northwind;")
changed: [postgresv14] => (item=psql -d northwind -f /var/lib/postgresql/northwind.sql)

TASK [Clean Files] ***********************************************************************************************************
changed: [postgresv14]

PLAY [Setup pgBouncer] *******************************************************************************************************

TASK [Install pgBouncer] *****************************************************************************************************
changed: [postgresv14] => (item=apt update)
changed: [postgresv14] => (item=apt install pgbouncer -y)

TASK [Deploy pgbouncer.ini File] *********************************************************************************************
changed: [postgresv14]

TASK [Load pgBouncer Script files] *******************************************************************************************
changed: [postgresv14]

TASK [Run pgBouncer Script Files] ********************************************************************************************
changed: [postgresv14] => (item=psql -d postgres  -f /var/lib/postgresql/pg_bouncer_config.sql)
changed: [postgresv14] => (item=psql -d postgres  -f /var/lib/postgresql/auth_function.sql)
changed: [postgresv14] => (item=psql -d template1 -f /var/lib/postgresql/auth_function.sql)
changed: [postgresv14] => (item=psql -d northwind -f /var/lib/postgresql/auth_function.sql)

TASK [Clean Files] ***********************************************************************************************************
changed: [postgresv14] => (item=rm /var/lib/postgresql/pg_bouncer_config.sql)
changed: [postgresv14] => (item=rm /var/lib/postgresql/auth_function.sql)

RUNNING HANDLER [Restart pgBouncer] ******************************************************************************************
changed: [postgresv14]

PLAY [Setup Monitor] *********************************************************************************************************

TASK [Install Flask] *********************************************************************************************************
changed: [postgresv14] => (item=apt-get install -y pip)
changed: [postgresv14] => (item=pip install flask)
changed: [postgresv14] => (item=mkdir /root/monitor)

TASK [Load Flask Files] ******************************************************************************************************
changed: [postgresv14]

TASK [Run Flask App] *********************************************************************************************************
changed: [postgresv14]

TASK [Display pgBouncer stats] ***********************************************************************************************
changed: [postgresv14 -> localhost]

PLAY RECAP *******************************************************************************************************************
localhost                  : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
postgresv14                : ok=16   changed=15   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

chanukyasds@thinkpad:~/ansible/dev$
```

When playbook execution completes, we will be redirected to a browser and can see the pgBouncer stats.

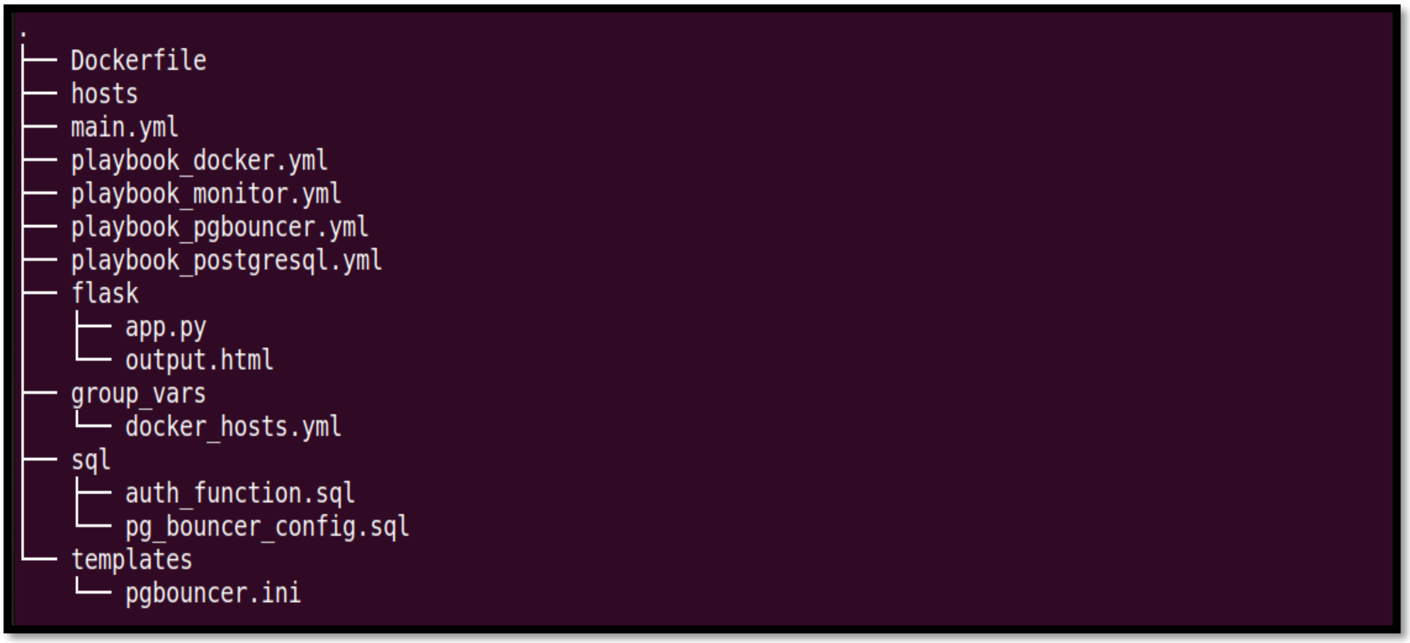


pgBouncer Monitor

This page will auto refresh every 5 sec.

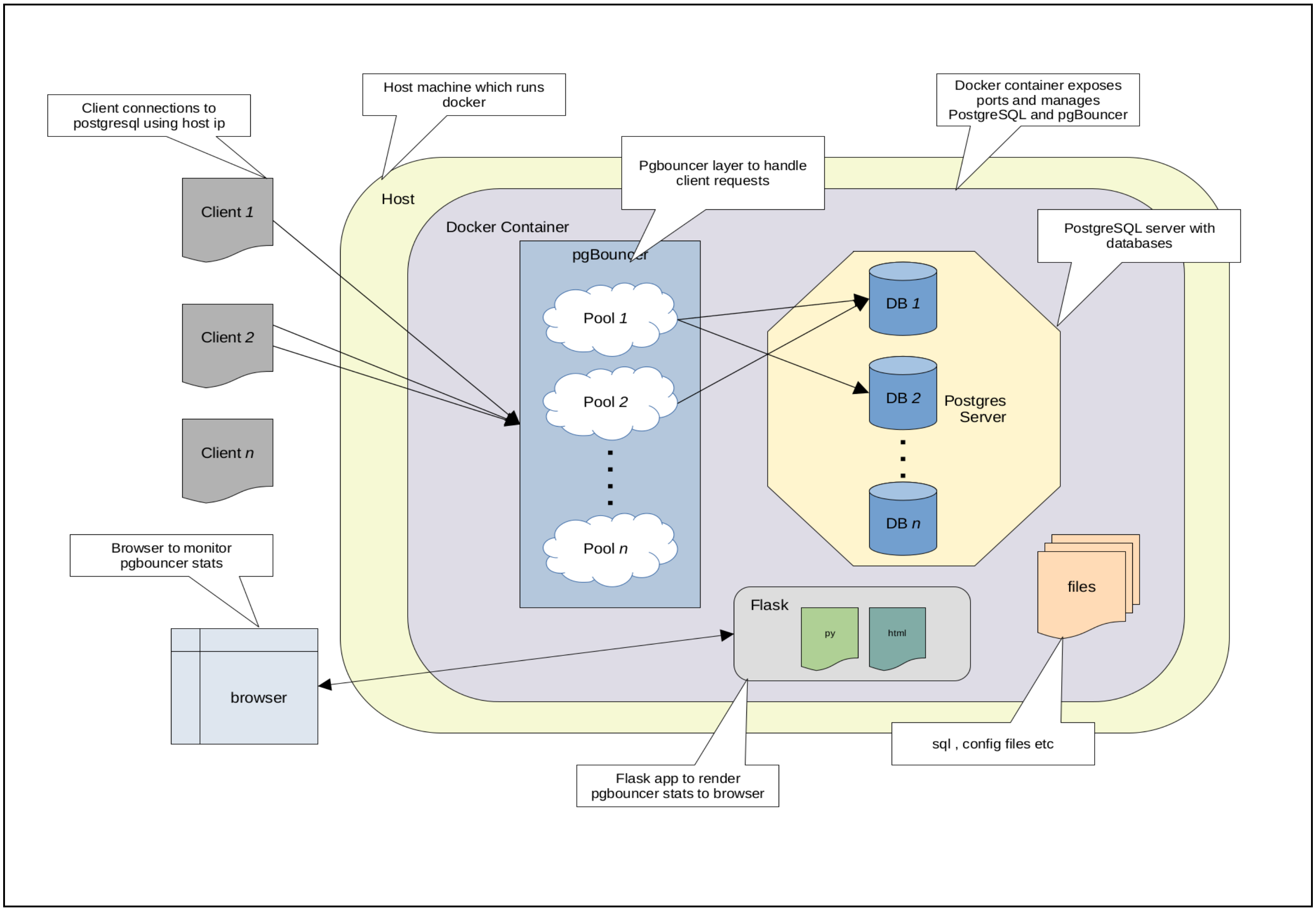
```
type |   user    | database  | state  | addr | port | local_addr | local_port |      connect_time       |      request_time       | wait | wait_us | close_needed |      ptr       | link | remote_pid | tls
-----+-----------+-----------+--------+------+------+------------+------------+-------------------------+-------------------------+------+---------+--------------+----------------+------+------------+-----
C    | pgbouncer | pgbouncer | active | unix | 6432 | unix       |       6432 | 2023-06-12 13:01:46 UTC | 2023-06-12 13:01:46 UTC |    0 |       0 |            0 | 0x5597d3ae7fd0 |      |       3588 |

database  |   user    | cl_active | cl_waiting | cl_cancel_req | sv_active | sv_idle | sv_used | sv_tested | sv_login | maxwait | maxwait_us | pool_mode
----------+-----------+-----------+------------+---------------+-----------+---------+---------+-----------+----------+---------+------------+-----------
pgbouncer | pgbouncer |         1 |          0 |             0 |         0 |       0 |       0 |         0 |        0 |       0 |          0 | statement

database  | total_xact_count | total_query_count | total_received | total_sent | total_xact_time | total_query_time | total_wait_time | avg_xact_count | avg_query_count | avg_recv | avg_sent | avg_xact_time |
----------+------------------+-------------------+----------------+------------+-----------------+------------------+-----------------+----------------+-----------------+----------+----------+---------------+-
pgbouncer |               76 |                76 |              0 |          0 |               0 |                0 |               0 |              0 |               0 |        0 |        0 |             0 |
```

FILE STRUCTURE LAYOUT



```
.
├── Dockerfile
├── hosts
├── main.yml
├── playbook_docker.yml
├── playbook_monitor.yml
├── playbook_pgbouncer.yml
├── playbook_postgresql.yml
├── flask
│   ├── app.py
│   └── output.html
├── group_vars
│   └── docker_hosts.yml
├── sql
│   ├── auth_function.sql
│   └── pg_bouncer_config.sql
└── templates
    └── pgbouncer.ini
```

GRAPHICAL DEMONSTRATION



Client connections to postgresql using host ip
Host machine which runs docker
Docker container exposes ports and manages PostgreSQL and pgBouncer
Pgbouncer layer to handle client requests
Host
Docker Container
pgBouncer
PostgreSQL server with databases
Client *1*
Client *2*
Client *n*
Pool *1*
Pool *2*
Pool *n*
DB *1*
DB *2*
DB *n*
Postgres Server
files
Browser to monitor pgbouncer stats
browser
Flask
py
html
sql , config files etc
Flask app to render pgbouncer stats to browser

Summary:

Docker contains our postgresql databases and we expose the pgbouncer port only to the real world with host-ip.

The clients make connections to postgresql database using HOSTIP. In the background, pgbouncer checks the user authentication and handles their requests.

Flask app will run to monitor the pgbouncer stats and render those stats to host browser.

If host is not interactive, we can use hostip:5000 to open the pgbouncer stats.

CODE REPO

This entire development code can be found in the GitHub repo.

https://github.com/chanukyasds/ansible_docker_pg_dev

THE END