Per service access to DB

REVIEW REQUIRED

Jira ticket

TECHGEN-1694 - [DevOps] Design Database Access Using multiple users with secrets in secret store to allow for ECS containers to have specific table access for rw, ro. A single table should only be written by a single container. **DEVELOPMENT COMPLETE**

Description

Design the way to organize per service access to DB as described in 12 factor app recommendations.

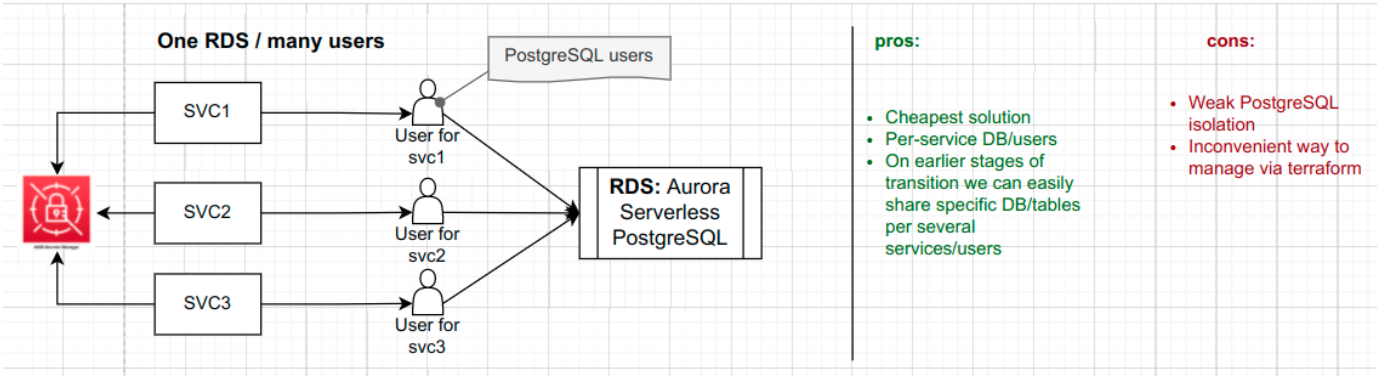
Each service should use dedicated user to auth to DB instance and have access only to DB/tables it need for work.

Since we are trying to use AWS services as most and manage infrastructure via terraform, the perfect way for us would be implement IAM based access for databases. But **current DB solution (Autora PostgreSQL Serverless) doesn't support IAM auth.**

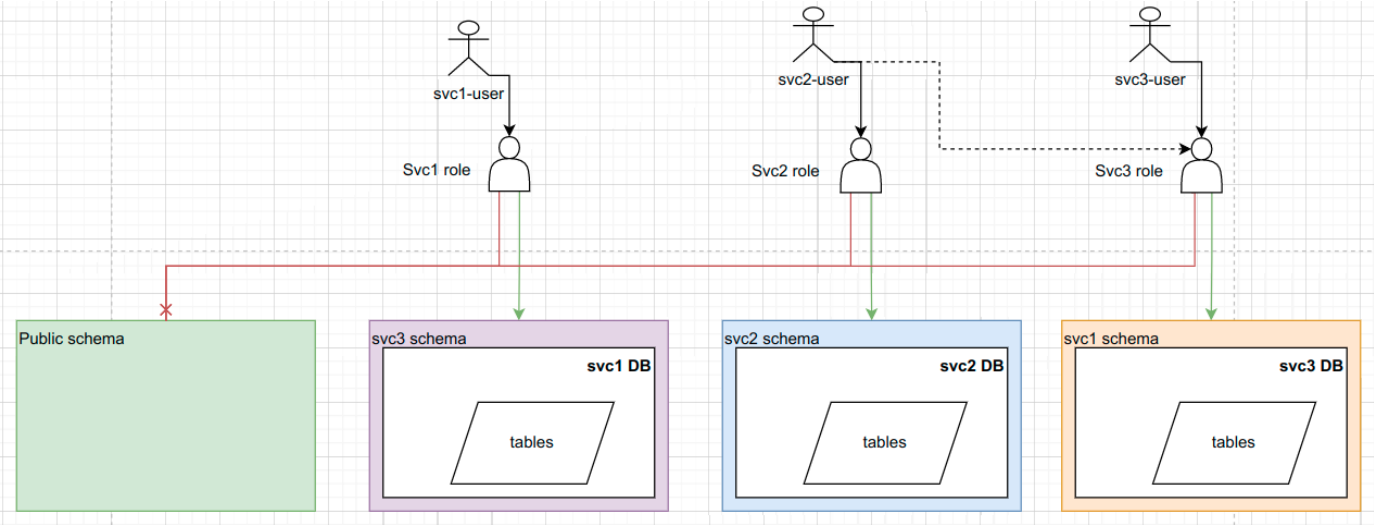
We agreed on following option:

- **Manage access only by PostgreSQL auth/permissions system.**
  - It compatible with current Aurora serverless solution
  - To implement strict permissions isolation we should add a lot of PostgreSQL configuration (E.g. create per app DB, create per app schema, revoke access from public schema, create DML and DDL roles, grant permissions to DB and schema, etc.). Do this might be inconvenient from terraform and it is easy to miss something that break isolation (See https://aws.amazon.com/blogs/database/managing-postgresql-users-and-roles/)
  - We can generate password by terraform prior PostgreSQL user creation and save it to AWS Secret Manager, where it can by consumed by ECS service

Schema

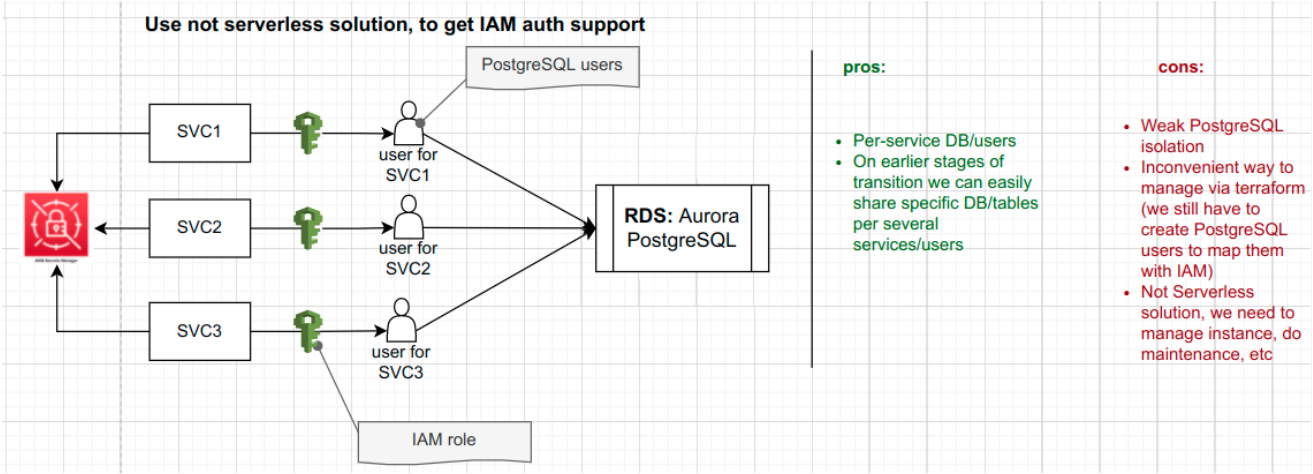


PostgreSQL level permissions:



Rejected options

- **Data API** https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/data-api.html **.**
  - Since require completely change services and has a lot of limitations.
- **Create separate Aurora Serverless for each service**
  - It is easy to manage by terraform, but too expensive
- **Move to Managed Aurora instances instead of Aurora serverless and manage access by IAM auth + PostgreSQL permissions system**
  - We can use IAM auth
    - We still have to create PostgreSQL with the same name as IAM role, so:
      - To implement strict permissions isolation we should add a lot of PostgreSQL configuration (E.g. create per app DB, create per app schema, revoke access frompublic schema, create DML and DDL roles, grant permissions to DB and schema, etc.). Do this might be inconvenient from terraform and it is ease to miss something that break isolation (See https://aws.amazon.com/blogs/database/managing-postgresql-users-and-roles/)
      - We don't set password for PostgreSQL users and use role for authentication and PostgreSQL role permissions for authorization.
  - We have to manage RDS updates, scaling, etc by our own
  - We can generate password by terraform prior Aurora Serverless creation and save it to AWS Secret Manager, where it can by consumed by ECS service
  - We really like this option, but will wait when we have more time for implementation
  -



Solution specifics

https://aws.amazon.com/blogs/database/overview-of-security-best-practices-for-amazon-rds-for-postgresql-and-amazon-aurora-postgresql-compatible-edition/

Implementation steps

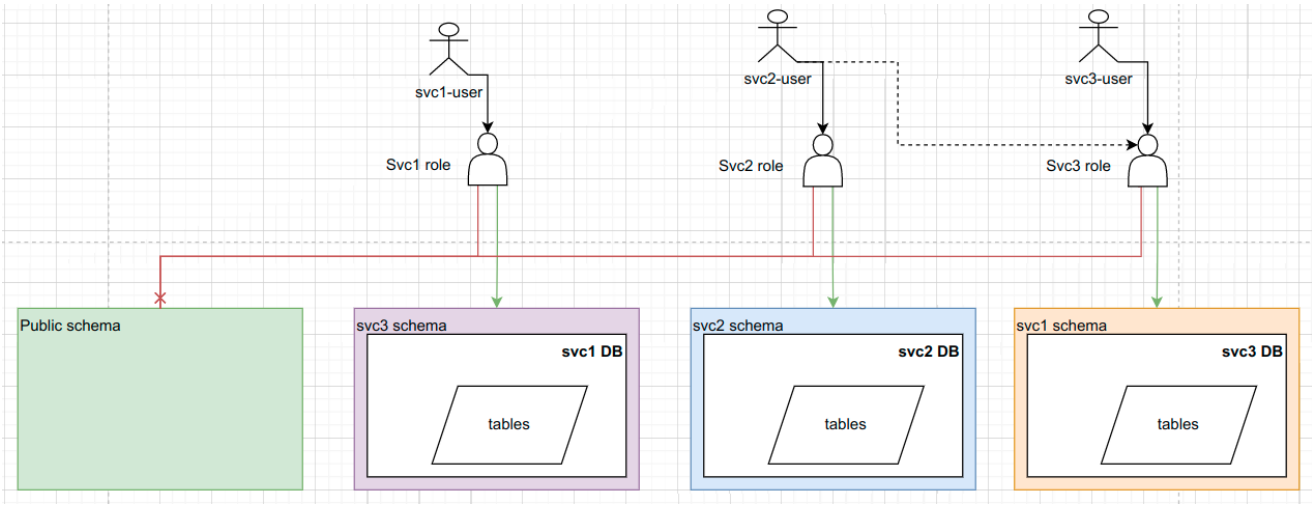
1. **Developers should make sure each service can work with its own database. Only then we can proceed**
2. Terraform should generate passwords that will be set to PostgreSQL users and saves them to Secret manager
3. For each service we should create separate DB, schema, role and user. And revoke access to public schema.
   We are going to manage this using PostgreSQL Terraform provider (https://registry.terraform.io/providers/cyrilgdn/postgresql/latest/docs) but for this we should have direct connection from Cloud9 instance where we run Terraform to RDS VPC. As solution we can setup one common VPC for all our Cloud9 instances with pre-configured VPC peering connections

```
# Part 1:: Run as admin on postgres DB
CREATE DATABASE <db name>;
REVOKE ALL ON DATABASE <db name> FROM PUBLIC;
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
CREATE ROLE <role name> WITH ENCRYPTED PASSWORD '<role password>';
GRANT CONNECT, TEMPORARY ON DATABASE <db name> TO <role name>;

# Part 2:: Run as admin on <db name> DB
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
CREATE SCHEMA IF NOT EXISTS <schema name>;
CREATE USER <user name> WITH ENCRYPTED PASSWORD '<user password>';
GRANT USAGE, CREATE ON SCHEMA <schema name> TO <role name>;
GRANT ALL ON ALL SEQUENCES <schema name> TO <role name>;
GRANT <role name> TO <user name>;
```

4. To organize cross DB permissions (not recommended, but probably we would need it anyway) we should do

```
GRANT <other service role name> TO <user for service who need
access>;
# And set required permissions for user to specific tables in schema
```



5. Update connection string for each ECS service

Deployment notes

1. Create per service Schemas/DBs/Users/Roles in parallel with existing common DB
2. With developers help split and move tables to per service DBs
3. If needed configure cross DB access for PostgreSQL users

4. Fix migrations
5. Repoint services to use new connection strings

Depends on

no