Documentação e instrução para configuração do RDS Terraform.

Fred Chardson Bezerra Lopes

1. Descrição da atividade.

Inicialmente fora colocado dentro do arquivo main.tf as chaves para poder acessar a conta na AWS no entanto, como isso caracteriza quebra de segurança as chaves encontram-se nesse arquivo na sessão de passo a passo, posteriormente fora criado os blocos de códigos para a criação da VPC, grupo de segurança, bem como as subnets, contudo essas, estas e aquelas estão comentadas a fim de deixar a cargo de quem for utilizar o código para criação do RDS com subnet's e VPC'S descomentar ou seguir os passos descritos abaixo para a criação do RDS apenas.

Após as configurações destes pré-requisitos tem-se a criação do RDS propriamente dito com todos as configurações necessárias para o seu funcionamento assim como as especificidades que foram solicitas, a exemplo, senha, nome do banco, nome do usuário, monitoramento, snapshots, janela de backup, multi_az, proteção contra exclusão e demais. No que diz respeito ao cloudwatch o mesmo encontra-se comentado, pois está como variável a ser digitada as configurações ao rodar o comando de aplicação do código.

No arquivo de saída existem as saídas de configuração do RDS tais como: endpoint, porta, grupo de segurança, estas servem para obter e lançar as configurações supra mencionadas após a efetivação da criação do banco de dados.

Por fim, no arquivo de variáveis estão todas as variáveis a serem digitadas na criação do RDS: nome do banco, nome do usuário, senha, id da VPC, subnet, lista de Arns para notificação de CPU, armazenamento, memória, disco.

As métricas do pgbench estão dispostas entre os dois bancos criados, stone-cadastro e stone-faturamento, a fim de atender uma das exigências de comunicação entre duas contas e respectivamente o débito e o crédito entre as mesmas, resultante da operação de transferência. Fora utilizado para métrica um cliente realizando 1.000.000 de transações em ambos os bancos, com exceção do banco stone-cadastro o qual fora utilizado um parâmetro de 1.300.000 transações para coleta de dados somente de leitura.

2. Passo a passo para a criação do RDS.

Passo a Passo para a criação do RDS

Considerando que exista uma VPC e subnet já configuradas

1. Copiar a url do git.
2. Executar o git clone por meio do gibash no diretório o qual deseja extrair.
3. Inserir a secret key no arquivo main.
4. Abrir o terminal no vscode ou outro.
5. Executar o comando terraform init.
6. Executar o comando terraform plan.
7. Executar o comando terraform apply.
8. Fornecer um nome para a instância do RDS.
9. Fornecer nome para o banco de dados.
10. Fornecer uma senha para o banco de dados.
11. Fornecer um nome de usuário para o banco de dados.
12. Fornecer o id da VPC, no caso em tela já existe, adicionar a VPC existente ou se preferir criar nova.

VPC = vpc-00a2ececc433ef7c4

access_key = "AKIAZYGQ2EFVPXE5VJCP"

secret_key = "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"

3. DUMP'S e configurações de restauração dos bancos de dados criados.

Por fim, como a Amazon cobra por tempo em que o RDS fica ligado, como também um valor extra pela disponibilidade de zona de AZ e este custo é considerável fora disponibilizado os DUMP's referentes aos bancos criados localmente atendendo à requisição de modelagem presente no desafio. Por meio do comando, psql -U -h -p -d -f onde U = usuário, h = host p = porta d = banco alvo, f = arquivo a ser carregado .sql. No terminal, prompt de comando, é possível a restauração do mesmo desde de haja um banco criado e com um esquema público os DUMP's foram criados utilizando a porta 5433. O usuário com permissão apenas de select fora criado com a denominação de db_user usando o comando grant do PostgresSQL. A tabela a qual contém 1.000.000 de linhas, contendo nome e e-mail, é a tabela tb_pessoa_fisica dentro do banco stone-cadastro no esquema public. Dentro do arquivo main.tf existem recursos para a criação tanto das subnets quanto de vpc's se por ventura seja do desejo criar.

Apêndice A

Script para atribuição de permissões a determinado grupo e restrições a outro (s). Usuário postgres possui permissão de super usuário, e, é membro do grupo all_group o qual possui privilégio de insert, delete, update, select, criar e excluir regras. Por outro lado, o usuário user_db possui apenas a permissão de select e, pertence ao grupo r_group, grupo o qual possui apenas permissão de leitura.

```
/*
   Name: abort_command(); Type: FUNCTION; Schema: public; Owner: postgres
   Função a qual bloqueia permissões não permitidas para o usuário

*/

create function public.abort_command() returns event_trigger
    language plpgsql
    as $$
begin --fora usado o postgres, mas poderia ser usado algum outro padrão de nome de
--usuário.
      if current_user not similar to 'postgres%' then
      raise notice '[TIME] %',
current_timestamp;

raise notice '[EVENT-TRIGGERED] abort_command';

raise notice '[DESCRIPTION] COMANDO BLOQUEADO POR GATILHO - USUARIO ATUAL NAO
POSSUI AS PERMISSOES NECESSARIAS !';

raise notice '[EVENT] %',
tg_event;

raise notice '[COMMAND] %',
tg_tag;

raise exception '[!] % NAO POSSUI PERMISSOES PARA USAR O COMANDO %',
current_user,
tg_tag;
end if;
end;

$$;

alter function public.abort_command() owner to postgres;

/*
    Name: trg_create_set_owner(); Type: FUNCTION; Schema: public; Owner: postgres
   Função a qual ativa o gatilho de bloqueio
*/


create function public.trg_create_set_owner() returns event_trigger
    language plpgsql
    as $$

declare

  obj record;
```

```
begin

	if current_user = 'user_db' then

		execute execute 'REASSIGN OWNED BY user_db TO all_group';

raise notice '[TIME] %',
current_timestamp;

raise notice '[EVENT-TRIGGERED] trg_create_set_owner';

raise notice '[DESCRIPTION] DONO DO OBJETO ALTERADO PELO GATILHO';

raise notice '[EVENT] %',
tg_event;

raise notice '[COMMAND] %',
tg_tag;
end if;
end;

$$;

alter function public.trg_create_set_owner() owner to postgres;

/*

 Name: usp_alter_privileges(); Type: FUNCTION; Schema: public; Owner: all_group

 */

create function public.usp_alter_privileges() returns void
    language plpgsql
    as $$
declare
	dbname text;

begin

select
	CURRENT_DATABASE()
into
	dbname;

raise INFO '============================================================';

raise INFO '[...] ALTERANDO PRIVILÉGIOS NO BANCO DE DADOS: %',
dbname;

raise INFO '============================================================';

execute 'ALTER DATABASE ' || dbname || ' OWNER TO all_group;';

raise INFO '[OK] DONO DO DATABASE % ALTERADO PARA all_group',
dbname;

alter schema public owner to all_group;

raise INFO '[OK] DONO DO SCHEMA public ALTERADO PARA all_group';

revoke all privileges on
all tables in schema public
from
public;
```

```
raise INFO '[OK] TODOS PRIVILEGIOS REVOGADOS DE TODAS TABELAS DO SCHEMA public DO
PUBLICO';

revoke all privileges on
all sequences in schema public
from
public;

raise INFO '[OK] TODOS PRIVILEGIOS REVOGADOS DE SEQUENCIAS NO SCHEMA public DO
PUBLICO';

execute 'REVOKE ALL PRIVILEGES ON DATABASE ' || dbname || ' FROM public';

raise INFO '[OK] TODOS PRIVILEGIOS REVOGADOS DO DATABASE % DO PUBLICO',
dbname;

revoke all privileges on
schema public
from
public;

raise INFO '[OK] TODOS PRIVILEGIOS REVOGADOS DO SCHEMA public DO PUBLICO';

execute 'GRANT CONNECT ON DATABASE ' || dbname || ' TO r_group WITH GRANT OPTION';

raise INFO '[OK] CONEXAO GARANTIDA PARA r_group';

execute 'GRANT ALL PRIVILEGES ON DATABASE ' || dbname || ' TO all_group WITH GRANT
OPTION';

raise INFO '[OK] PRIVILEGIOS GARANTIDOS NO DATABASE PARA all_group';

grant usage on
schema public to r_group with grant option;

raise INFO '[OK] PRIVILEGIO USAGE GARANTIDO NO SCHEMA public PARA r_group';

grant all privileges on
schema public to all_group;

raise INFO '[OK] PRIVILEGIOS GARANTIDOS NO SCHEMA public PARA all_group';

grant
select
      on
      all tables in schema public to r_group with grant option;

raise INFO '[OK] PRIVILEGIO SELECT GARANTIDO EM TODAS TABELAS DO SCHEMA public
PARA r_group';

grant
select
      ,
      insert
      ,
      update
      ,
      delete
      on
      all tables in schema public to rw_group with grant option;

raise INFO '[OK] PRIVILEGIOS SELECT, INSERT, UPDATE, DELETE GARANTIDOS EM TODAS
TABELAS DO SCHEMA public PARA r_group';

grant all privileges on
all tables in schema public to all_group with grant option;
```

```
raise INFO '[OK] PRIVILEGIOS GARANTIDOS EM TODAS TABELAS DO SCHEMA public PARA
all_group';

grant usage,
select
	on
	all sequences in schema public to r_group with grant option;

grant all privileges on
all sequences in schema public to all_group with grant option;

raise INFO '============================================================';

raise INFO '[OK] PRIVILEGIOS ALTERADOS NO BANCO DE DADOS: %',
dbname;

raise INFO '============================================================';
end;

$$;

alter function public.usp_alter_privileges() owner to all_group;

revoke all on
schema public
from
user_db;

revoke all on
schema public
from
PUBLIC;

grant all on
schema public to PUBLIC;

/*
 Name: DEFAULT PRIVILEGES FOR TABLES; Type: DEFAULT ACL; Schema: -; Owner:
postgres
*/


alter default privileges for role user_db grant
select
	on
	tables to r_group;

alter default privileges for role postgres grant
select
	,
	insert
	,
	delete
	,
	update
	on
	tables to all_group;

/*

 Name: abort_command; Type: EVENT TRIGGER; Schema: -; Owner: all_group

*/


create event trigger abort_command on
ddl_command_start
```

```
when TAG in ('GRANT', 'REVOKE')
   execute function public.abort_command();

alter event trigger abort_command owner to all_group;

/*
 Name: trg_create_set_owner; Type: EVENT TRIGGER; Schema: -; Owner: postgres
*/

create event trigger trg_create_set_owner on
ddl_command_end
when TAG in ('CREATE TABLE', 'CREATE TYPE', 'CREATE SEQUENCE', 'CREATE DOMAIN')
   execute function public.trg_create_set_owner();

      alter event trigger trg_create_set_owner owner to postgres;
```

Comandos os quais concedem e revogam privilégios ao usuário (user_db), bem como garantem sua conexão com o banco de dados.

```
grant connect on database "stone-faturamento" to user_db;

alter default privileges in schema public grant all on tables to user_db;

grant usage on schema public to user_db;

grant select on all sequences in schema public to user_db;

grant select on all tables in schema public to user_db;

revoke all privileges on all tables in schema public from user_db;
```

Apêndice B

Pgbench do banco de dados stone-cadastro e stone-faturamento.

Identificação das siglas:

-c = número de clientes.

-j = número de threads.

-t = quantidade de transações.

Dados pgbench banco stone-cadastro.

Comando usado: C:\Users\USER>pgbench -U postgres -p 5433 -c 1 -j 2 -t 1000000 stone-cadastro

Dados da tabela 1 evidenciam que o banco de cadastro realizou um total de 76.43 de tps, ou seja, realizou um total de 76.43 transações por segundo. Sendo, para tanto, fornecido um cliente e utilizando uma thread em um total de 1.000.000 de transações para esse cliente. O que dentro da perspectiva do server cuja finalidade é de cadastro de clientes significa que o server - o banco- suporta uma carga de 76.43 cadastros por segundo de clientes ou consultas aos mesmos, dentro do que fora fornecido como entrada. Em um cenário acima de 1.500.000 transações passadas como parâmetro de entrada o comportamento tornou-se demasiadamente lento.

starting vacuum...end.

Tabela 1: dados pgbench banco stone-cadastro.

transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 100000
number of transactions actually processed: 100000/100000
latency average = 13.084 ms
tps = 76.428351 (including connections establishing)

tps = 76.435454 (excluding connections establishing)

Fonte: autor.

Dados pgbench somente de leitura banco stone-cadastro

Dados da tabela 2 mostram que em um cenário só de leitura o server – o banco – obteve um rendimento de 263.62 tps, o que implica em dizer que o banco tendo como parâmetro de entrada 1.000.000 de transações a serem realizadas forneceu um valor de 263.62 transações por segundo de leitura utilizando uma thread e um cliente. Assim como na tabela 1 referente ao pgbench dados de entrada acima de 1.500.000 transações e, com mais de um cliente tornaram-se demasiadamente lentos, dentro desta perspectiva, o banco de dados como está, sem trabalho de indexação e outros que melhoram a performance.

Comando usado: C:\Users\USER>pgbench -U postgres -p 5433 -c 1 -j 2 -t 1000000 -S stone-cadastro

Password:

starting vacuum...end.

Tabela 2: dados pgbench banco stone-cadastro.

transaction type: <builtin: select only>
scaling factor: 50
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000000
number of transactions actually processed: 1000000/1000000
latency average = 3.793 ms
tps = 263.613621 (including connections establishing)
tps = 263.620271 (excluding connections establishing)

Fonte: autor.

Dados pgbench do banco stone-faturamento somente de leitura.

Dados da tabela 3 mostram que esse banco conseguiu realizar um total de 1.300.000 transações somente de leitura o que dá um total de 325.16 transações por segundo. Valores acima disso, tanto para o número de transações quanto para o número de clientes, nesse cenário, passados como parâmetro de entrada tornam-se inviáveis, uma vez que, o banco rodou infinitamente a operação e não a concluiu.

C:\Users\USER>pgbench -U postgres -p 5433 -c 1 -j 2 -t 1300000 -S stone-faturamento

Password:

starting vacuum...end.

Tabela 3: dados pgbench banco stone-faturamento somente leitura.

transaction type: <builtin: select only>
scaling factor: 50
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1300000
number of transactions actually processed: 1300000/1300000
latency average = 3.075 ms
tps = 325.161757 (including connections establishing)
tps = 325.169968 (excluding connections establishing)

Fonte: autor.

A tabela 4 mostra os dados extraídos do comando pgbench o qual retrata a saída fornecida de acordo com os parâmetros de entrada os quais foram: 1.000.000 de transações a serem realizadas por um cliente utilizando duas threads, no entanto, efetivamente fora utilizada apenas uma. Dentro dessa perspectiva o banco de dados realizou esse quantitativo de transações com uma latência de 21,3 milissegundos e um total de 46,79 transações por segundo. Valores acima destes foram testados e o comportamento do bando tornou-se demasiadamente demorado.

C:\Users\ACER>pgbench -U postgres -p 5433 -c 1 -j 2 -t 100000 stone-faturamento

Password:

starting vacuum...end.

Tabela 4: dados pgbench banco stone-faturamento.

transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 100000
number of transactions actually processed: 100000/100000
latency average = 21.369 ms
tps = 46.797493 (including connections establishing)
tps = 46.799304 (excluding connections establishing)

Fonte: autor.