# **Migración de Postgres a EC2 y AWS RDS**

Primero vamos a probar a instalar Postgres en EC2 y realizar su configuración junto con el volcado de datos

# **Instalar PostgreSQL en EC2**

### **1. Instalar PostgreSQL de forma manual**

Para instalar PostgreSQL de forma manual, abre una terminal y ejecuta los siguientes comandos para asegurarte de tener la última versión del sistema:

sudo apt update -y

sudo apt install -y postgresql postgresql-contrib

Esto instalará PostgreSQL junto con algunas herramientas adicionales útiles.

### **2. Iniciar y habilitar PostgreSQL**

Una vez instalado, necesitas iniciar el servicio de PostgreSQL y habilitarlo para que se inicie automáticamente al arrancar el sistema. Ejecuta los siguientes comandos:

sudo systemctl enable postgresql

sudo systemctl start postgresql

### **3. Crear un nuevo usuario y base de datos en PostgreSQL**

Ahora, vas a crear un usuario y una base de datos en PostgreSQL. Primero, accede como el usuario postgres:

sudo -i -u postgres

Luego, abre la consola de PostgreSQL:

psql

Dentro de la consola de PostgreSQL, ejecuta los siguientes comandos para crear un usuario y una base de datos:

CREATE USER nombre\_usuario WITH PASSWORD 'password123';

CREATE DATABASE nombre\_base\_datos OWNER nombre\_usuario;

GRANT ALL PRIVILEGES ON DATABASE nombre\_base\_datos TO nombre\_usuario;

Recuerda que debes sustituir nombre\_usuario y nombre\_base\_datos por el nombre que desees para tu usuario y base de datos. También, recuerda cambiar la contraseña si es necesario.

### **4. Configurar autenticación para el nuevo usuario**

La configuración de autenticación de PostgreSQL se encuentra en el archivo **pg\_hba.conf**. Necesitamos modificarlo para permitir la conexión con el nuevo usuario. Abre el archivo de configuración:

sudo nano /etc/postgresql/$(ls /etc/postgresql)/main/pg\_hba.conf

Agrega las siguientes líneas al final de este archivo para permitir conexiones con el nuevo usuario:

host nombre\_base\_datos nombre\_usuario 127.0.0.1/32 md5

host nombre\_base\_datos nombre\_usuario ::1/128 md5

Guarda y cierra el archivo.

### **5. Reiniciar PostgreSQL para aplicar los cambios**

Para que los cambios se apliquen, reinicia el servicio de PostgreSQL con el siguiente comando:

sudo systemctl restart postgresql

### **Conectar a PostgreSQL**

### **6. Conectar a PostgreSQL localmente**

Para conectarte a PostgreSQL como el usuario postgres, ejecuta el siguiente comando:

sudo -i -u postgres

psql -d nombre\_base\_datos -U nombre\_usuario

Recuerda sustituir nombre\_base\_datos y nombre\_usuario por los valores que elegiste al crear la base de datos.

### **7. Habilitar conexiones remotas a PostgreSQL**

Para permitir conexiones remotas a PostgreSQL, necesitas modificar algunos archivos de configuración.

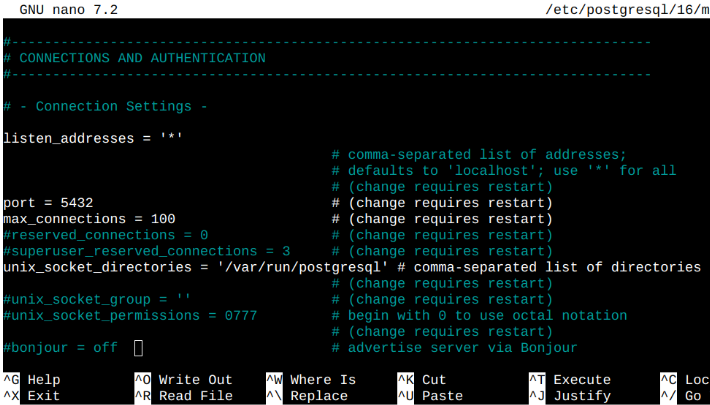
### **Editar postgresql.conf**

Abre el archivo postgresql.conf con el siguiente comando:

sudo nano /etc/postgresql/$(ls /etc/postgresql)/main/postgresql.conf

Busca la línea que dice listen\_addresses y cambia su valor para permitir conexiones desde cualquier dirección IP (o una IP específica si lo prefieres):

listen\_addresses = '\*'



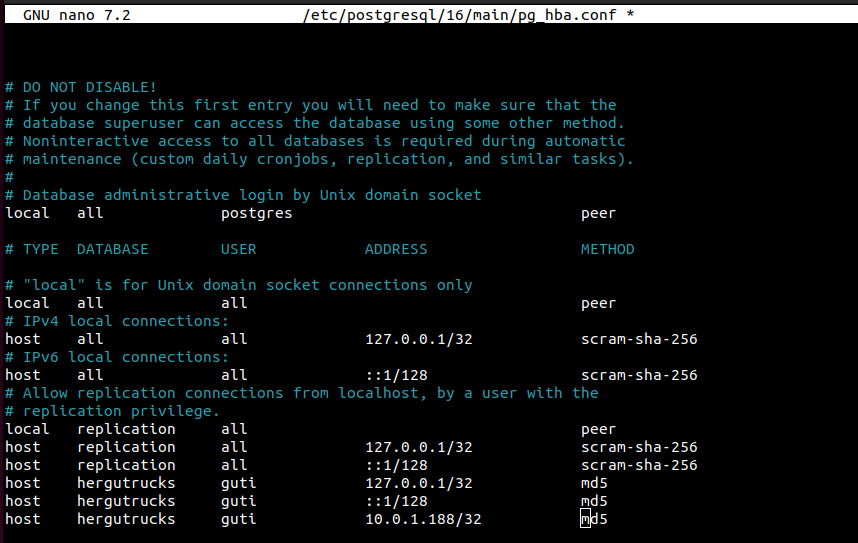
### **Configurar el archivo pg\_hba.conf**

Luego, abre el archivo de autenticación **pg\_hba.conf** nuevamente:

sudo nano /etc/postgresql/$(ls /etc/postgresql)/main/pg\_hba.conf

Asegúrate de que tenga una línea que permita conexiones desde las direcciones IP que necesites. Si deseas permitir acceso remoto desde una dirección IP específica, agrega lo siguiente:

host nombre\_base\_datos nombre\_usuario IP\_aplicacion/32 md5



No olvides reemplazar IP\_aplicacion con la IP de la máquina desde donde te conectarás.

### **8. Abrir el puerto 5432 en el firewall**

Si tienes un firewall activo, debes abrir el puerto 5432 (el puerto predeterminado de PostgreSQL). Ejecuta este comando:

sudo ufw allow 5432/tcp

Ahora podrás conectarte a PostgreSQL desde una máquina remota.

### **Transferir la Base de Datos de Django a PostgreSQL**

### **9. Instalar los paquetes necesarios**

Primero, instala las bibliotecas necesarias para que Django pueda interactuar con PostgreSQL. Ejecuta los siguientes comandos:

sudo apt update

sudo apt install libpq-dev

pip install psycopg2

### **10. Exportar los datos de Django a un archivo JSON**

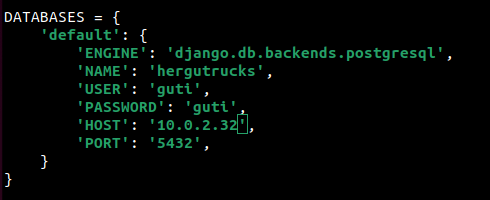
Antes de hacer la migración a PostgreSQL, debes exportar los datos actuales de Django a un archivo JSON. Para hacer esto, ejecuta el siguiente comando desde el directorio donde se encuentra tu proyecto Django:

python3 manage.py dumpdata > data.json

Este comando exporta todos los datos de las tablas de Django en un formato JSON que luego podrás importar en PostgreSQL.

### **11. Configurar Django para usar PostgreSQL**

Abre el archivo settings.py de tu proyecto Django. En la sección **DATABASES**, configura la conexión a tu base de datos PostgreSQL como sigue:



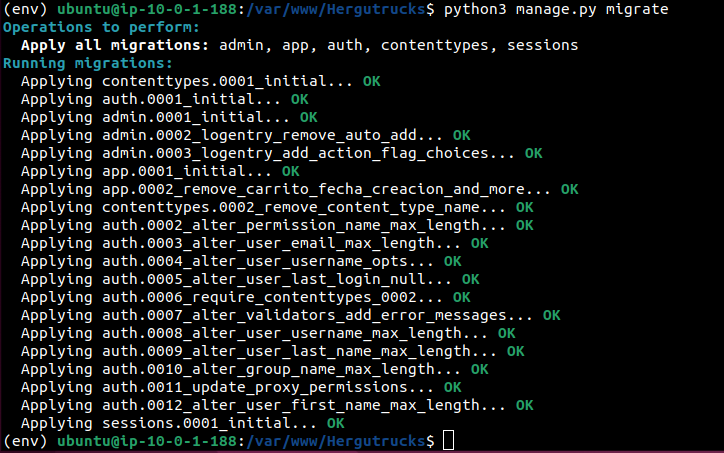
### 

### **12. Aplicar migraciones en PostgreSQL**

Ahora, aplica las migraciones necesarias para configurar las tablas en PostgreSQL. Ejecuta estos comandos:

python3 manage.py makemigrations

python3 manage.py migrate



Esto crea las tablas en tu base de datos PostgreSQL.

### **13. Importar los datos exportados**

Una vez que hayas migrado las tablas, importa los datos que exportaste en el paso 10 con el siguiente comando:

python3 manage.py loaddata data.json



Esto restaura los datos de tu base de datos Django a PostgreSQL.

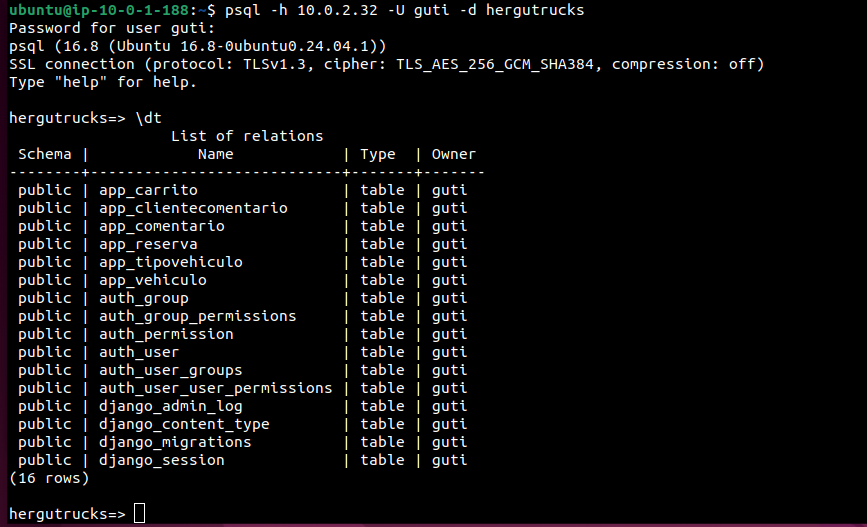
### **14. Verificar que los datos fueron importados correctamente**

Para asegurarte de que los datos se han importado correctamente, abre una conexión a PostgreSQL y ejecuta el siguiente comando para ver las tablas:

psql -d nombre\_base\_datos -U nombre\_usuario

Luego, dentro de la consola de PostgreSQL, ejecuta:

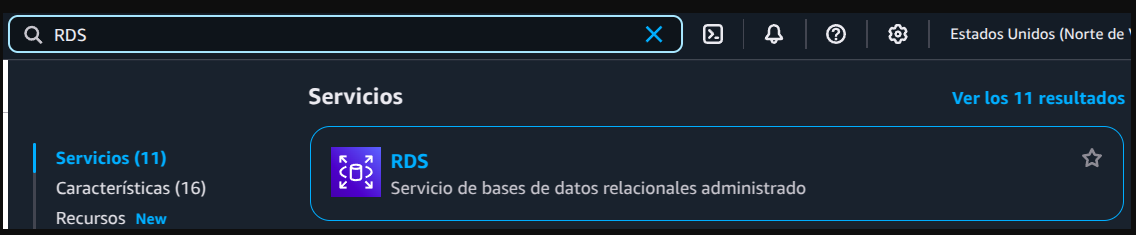
\dt



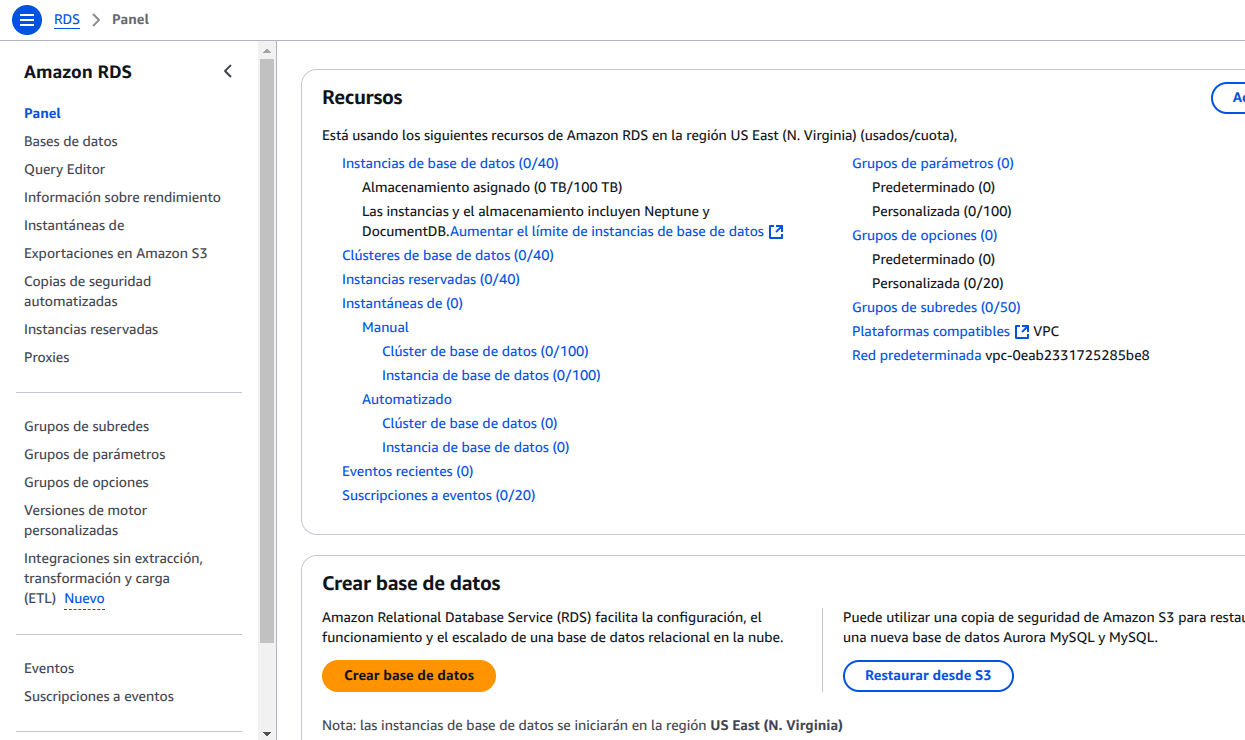
Esto te mostrará las tablas en la base de datos. Si ves que las tablas de tu aplicación Django están presentes, significa que la migración fue exitosa.

# **Haciendo uso de AWS RDS**

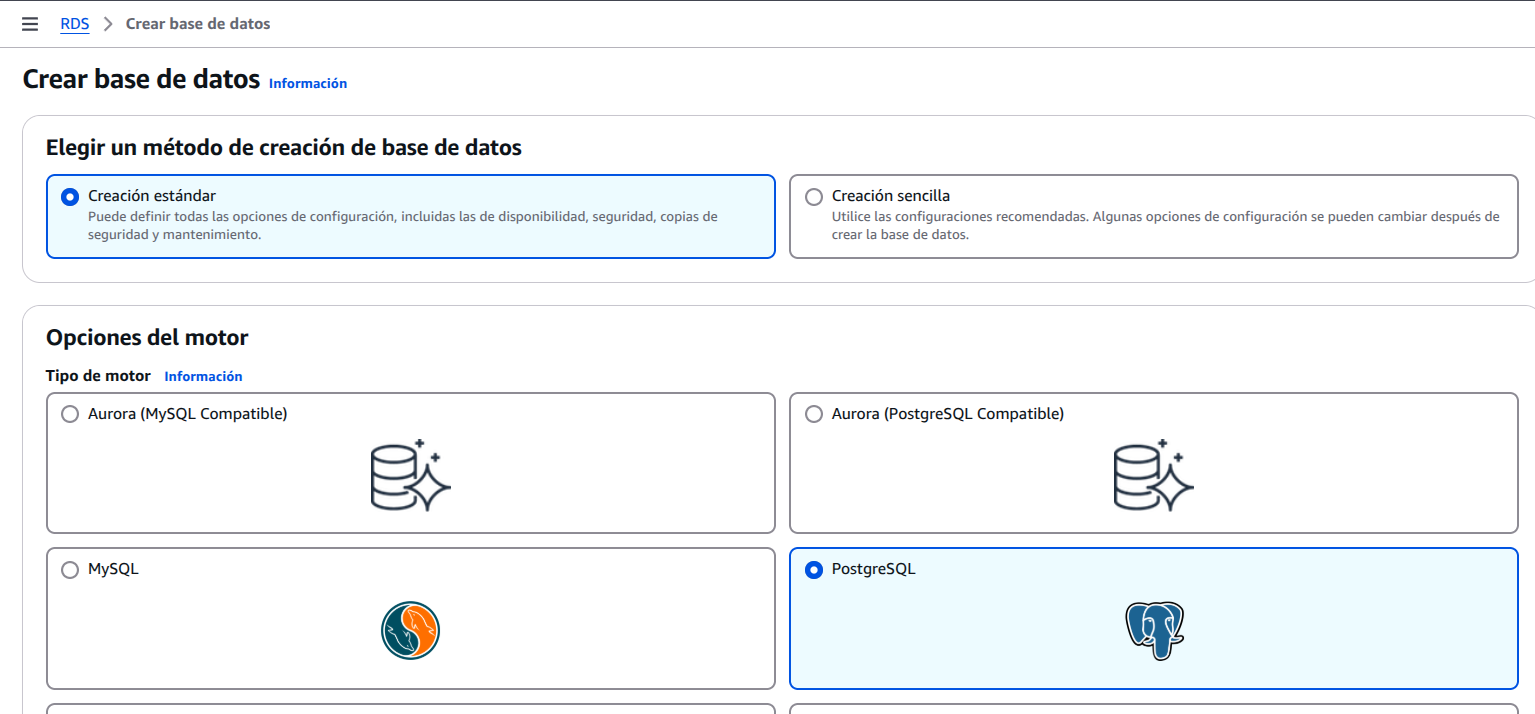
Escribimos “RDS” en el buscador de AWS



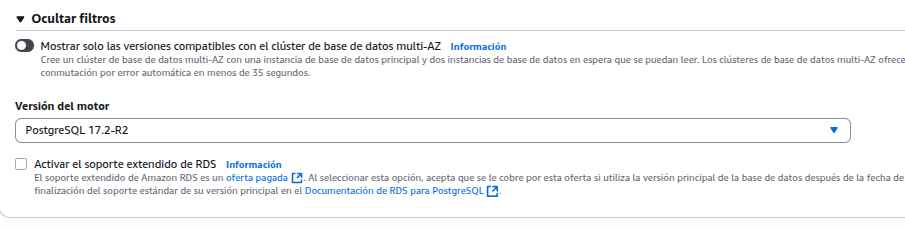
Una vez cargue la página clico sobre el botón de crear base de datos



Marco la opción de “Creación estándar” y seleccione la base de datos que más me gusta, en este caso usare PostgreSQL

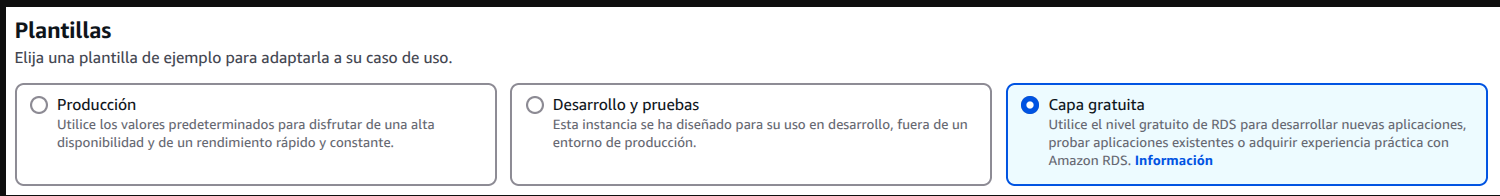


Si nos movemos hacia abajo en la página, veremos que nos permite modificar la versión del motor de la base de datos:

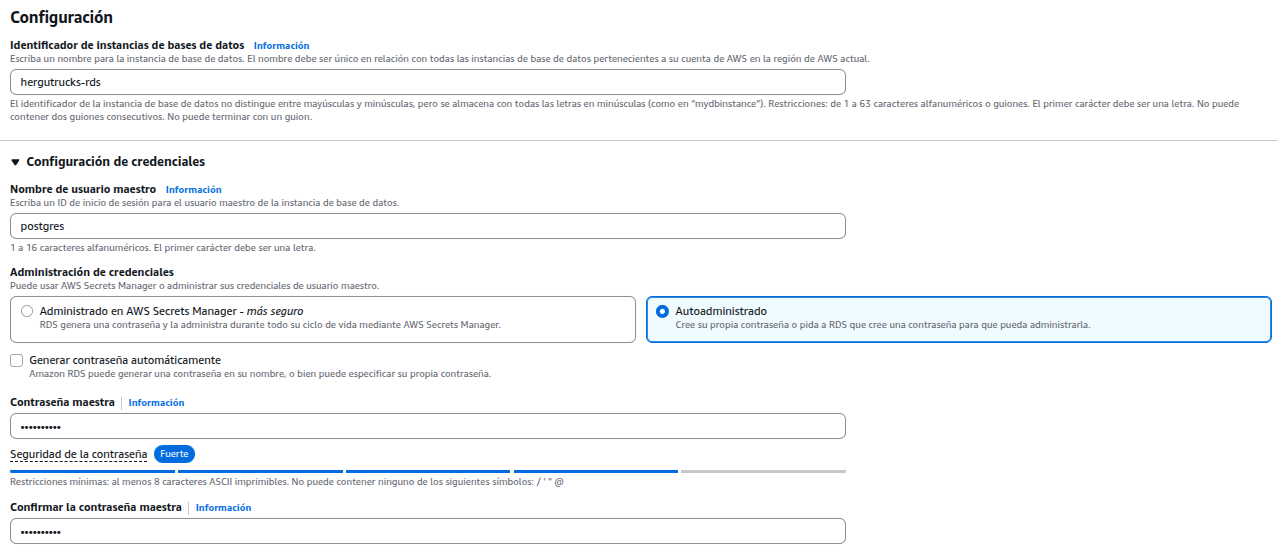


Yo lo dejaré con sus valores por defecto.

Si seguimos bajando nos permite seleccionar la plantilla, en nuestro caso vamos a escoger la versión gratuita

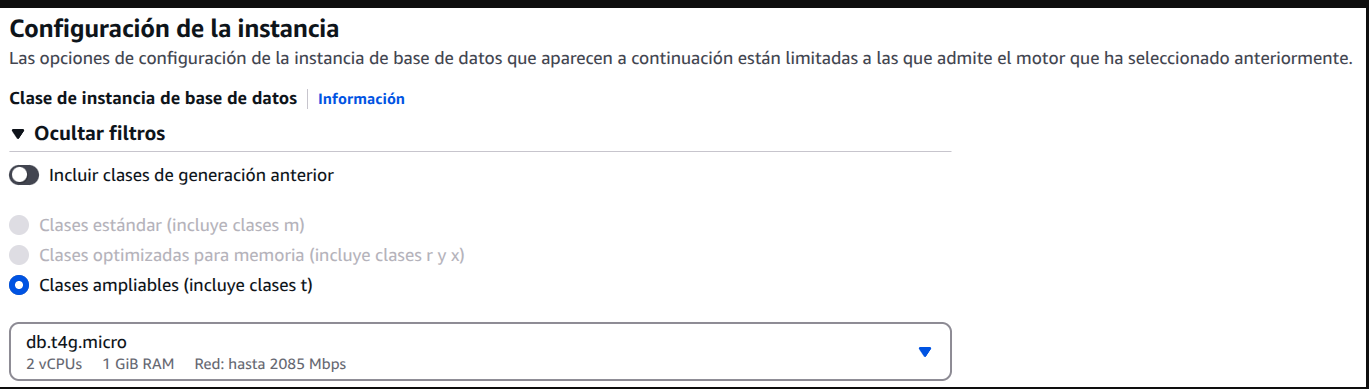


En el apartado de configuración simplemente le daré un nombre identificador de la base de datos y agregare una contraseña. Y aunque podemos optar por dos métodos para la gestión de credenciales: **AWS Secrets Manager** o **Self managed**. En este caso, seleccionaremos **Self managed** para reducir costos. Sin embargo, es importante considerar que la opción de **AWS Secrets Manager** ofrece un nivel de seguridad superior, ya que permite la rotación automática de las claves de acceso.

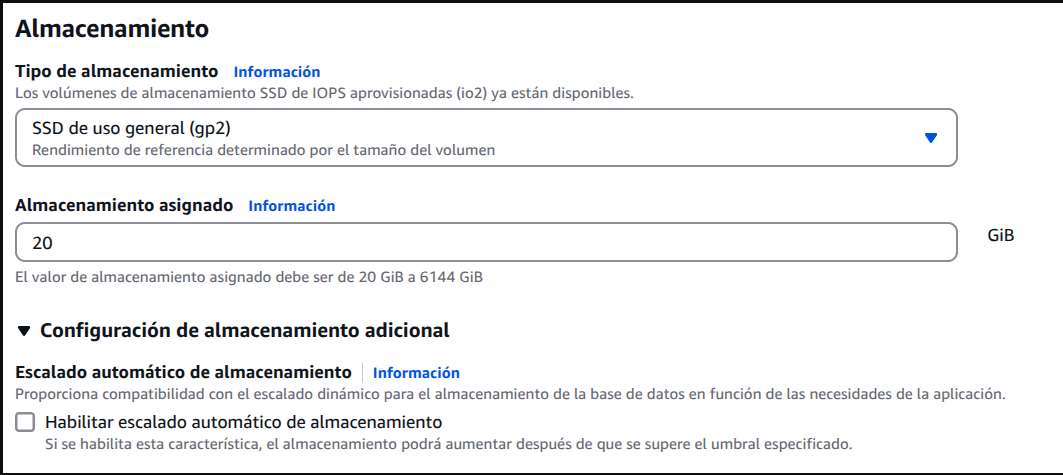


Hemos optado por migrar la base de datos PostgreSQL a una instancia EC2 en una subnet privada, eligiendo específicamente una instancia **"db.t4g.micro"**, que forma parte del **tier gratuito** de AWS. Aquí te cuento por qué creemos que es la mejor opción:

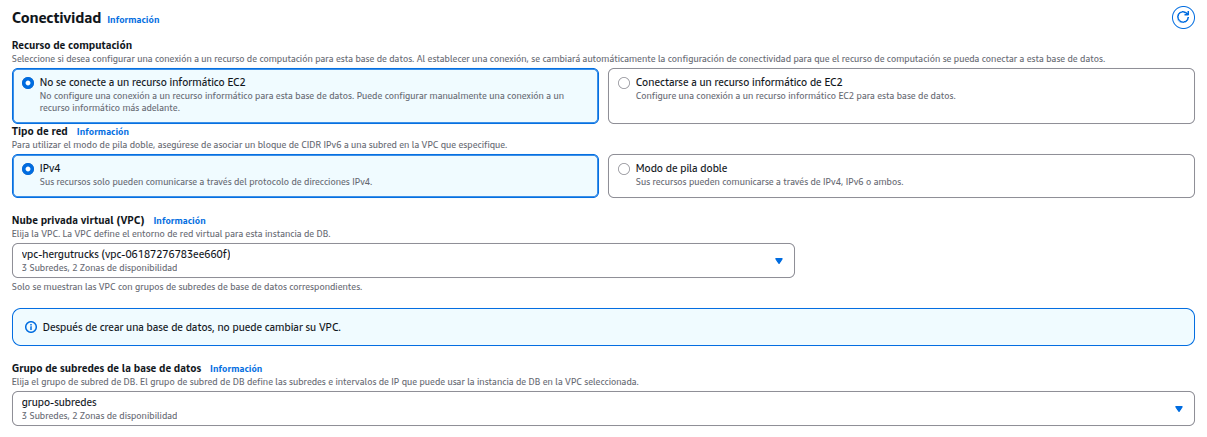
* **Mayor control y personalización:**Usar EC2 nos permite ajustar la configuración de la base de datos según nuestras necesidades. Podemos optimizar el sistema operativo, instalar herramientas adicionales y modificar parámetros avanzados sin las restricciones que tendríamos con RDS.
* **Seguridad reforzada:**Al alojar la base de datos en una subnet privada, evitamos que sea accesible desde internet, reduciendo riesgos y garantizando que solo los recursos internos puedan conectarse. Esto mejora la protección de los datos.
* **Flexibilidad y escalabilidad a nuestro ritmo:**Aunque RDS permite escalar automáticamente, con EC2 tenemos la posibilidad de decidir cuándo y cómo aumentar los recursos. Podemos ajustar el tamaño de la instancia o migrar a una más potente si en el futuro la demanda crece.
* **Ahorro de costos:**La instancia **db.t4g.micro** es gratuita, lo que nos permite reducir costos sin sacrificar funcionalidad. Además, al gestionar la infraestructura nosotros mismos, evitamos los costos adicionales de un servicio gestionado como RDS.



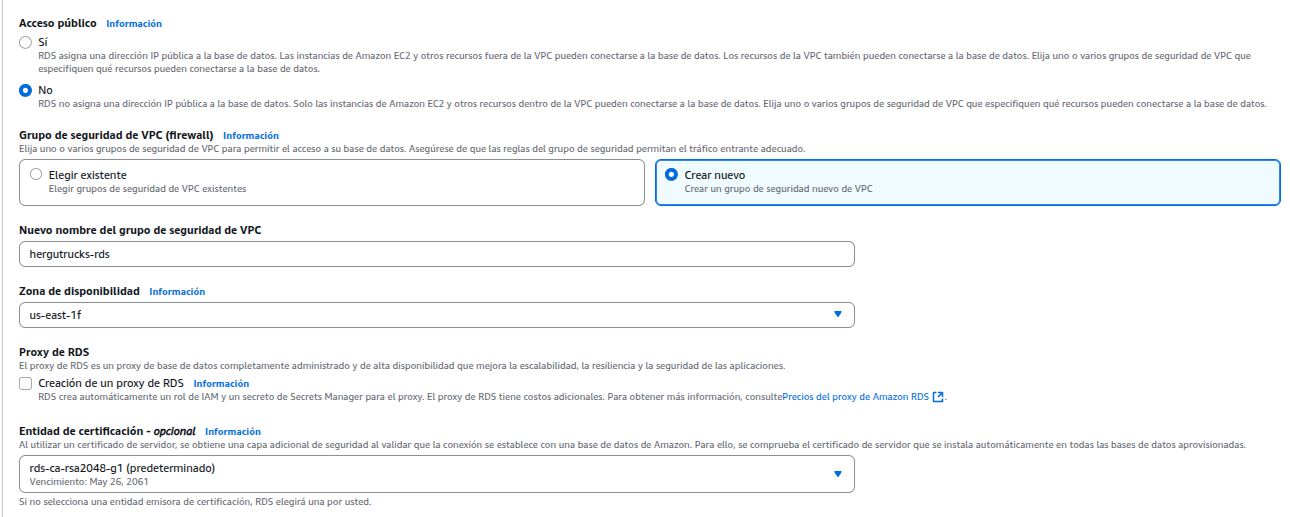
Vamos a mantener el almacenamiento en el mínimo permitido, que en este caso son **20GB**. Además, **deshabilitaremos el Storage Autoscaling** para reducir costos y evitar incrementos automáticos en el almacenamiento que no sean estrictamente necesarios. Si en el futuro vemos que necesitamos más espacio, siempre podemos ampliarlo manualmente según la demanda.



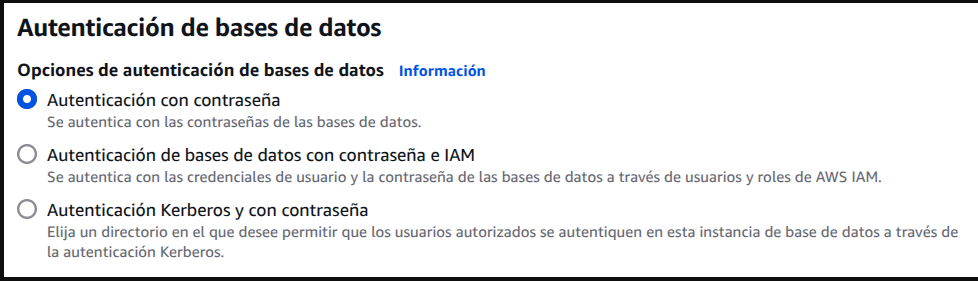
Ahora toca configurar la conectividad de la base de datos. En nuestro caso, ya contamos con un **VPC configurado con dos subredes**, una pública y otra privada. Vamos a utilizar este mismo **VPC**, asegurándonos de que en él se creen las subredes necesarias para que **RDS pueda conectarse correctamente** dentro de nuestra infraestructura.



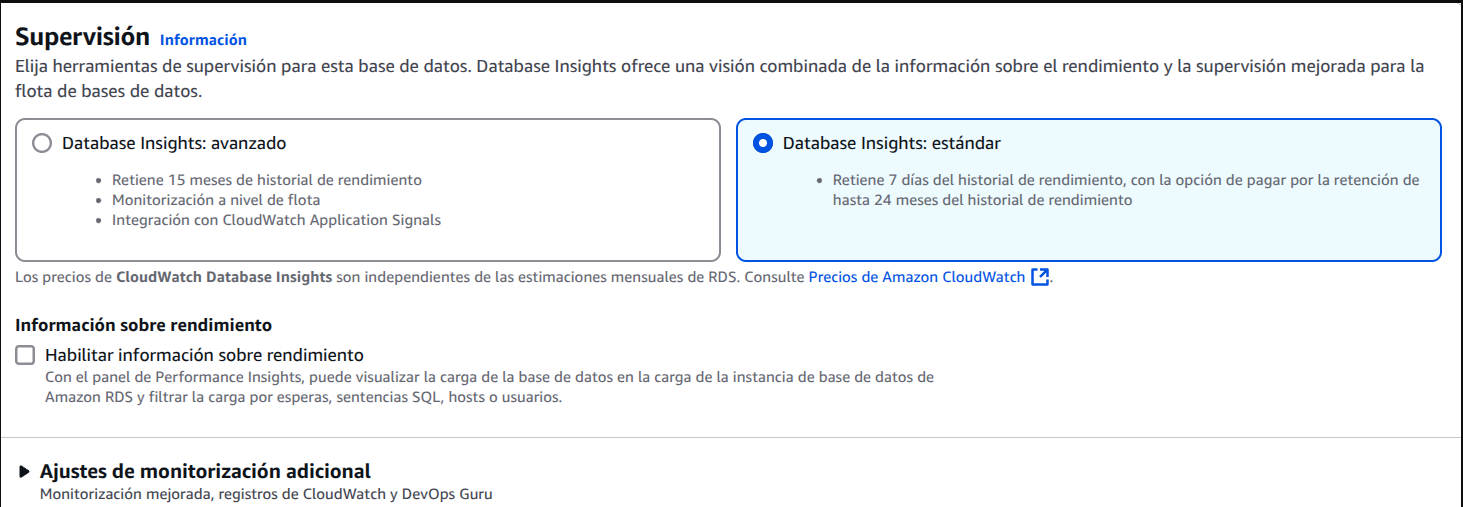
Verificamos que la base de datos **no tenga acceso público** y, además, creamos un **nuevo grupo de seguridad** específicamente para el RDS, asegurando un control adecuado del tráfico y los accesos.



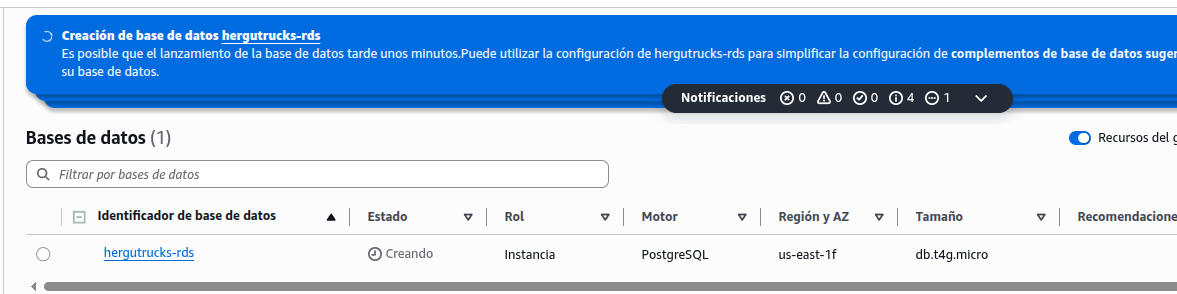
La base de datos utilizará autenticación mediante contraseña.



En la supervisión (monitorización) deshabilitamos la casilla de habilitar información sobre el rendimiento



Una vez hecho todo esto ya podemos ir al final y clicar sobre el botón que dice “Crear base de datos” para comprobar el resultado



Esperamos unos instantes para comprobar que se ha creado correctamente

