Contents

Introduction

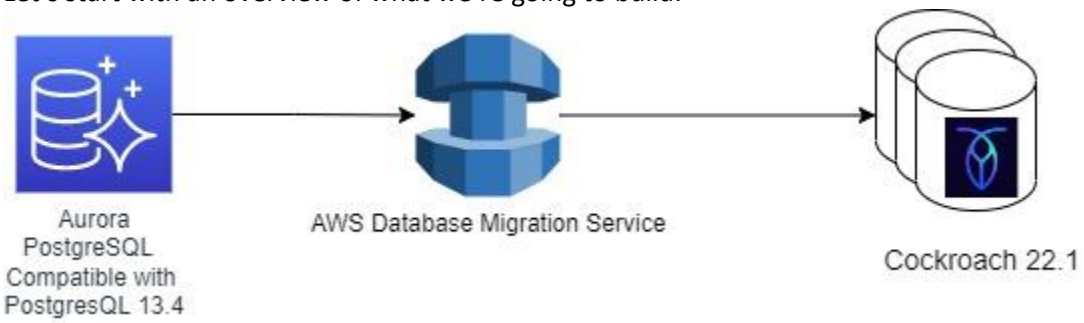
So many exciting things are always happening at Cockroach Labs, thanks in large part to our amazing engineering staff. Our release of 22.1 promises to continue to push our Distributed SQL forward with many new features and improvements. One of the most exciting things for me in 22.1 is the Preview Release of CockroachDB's integration with AWS Data Migration Service (DMS)! Stay tuned, many more exciting advancements with DMS are already in the works.

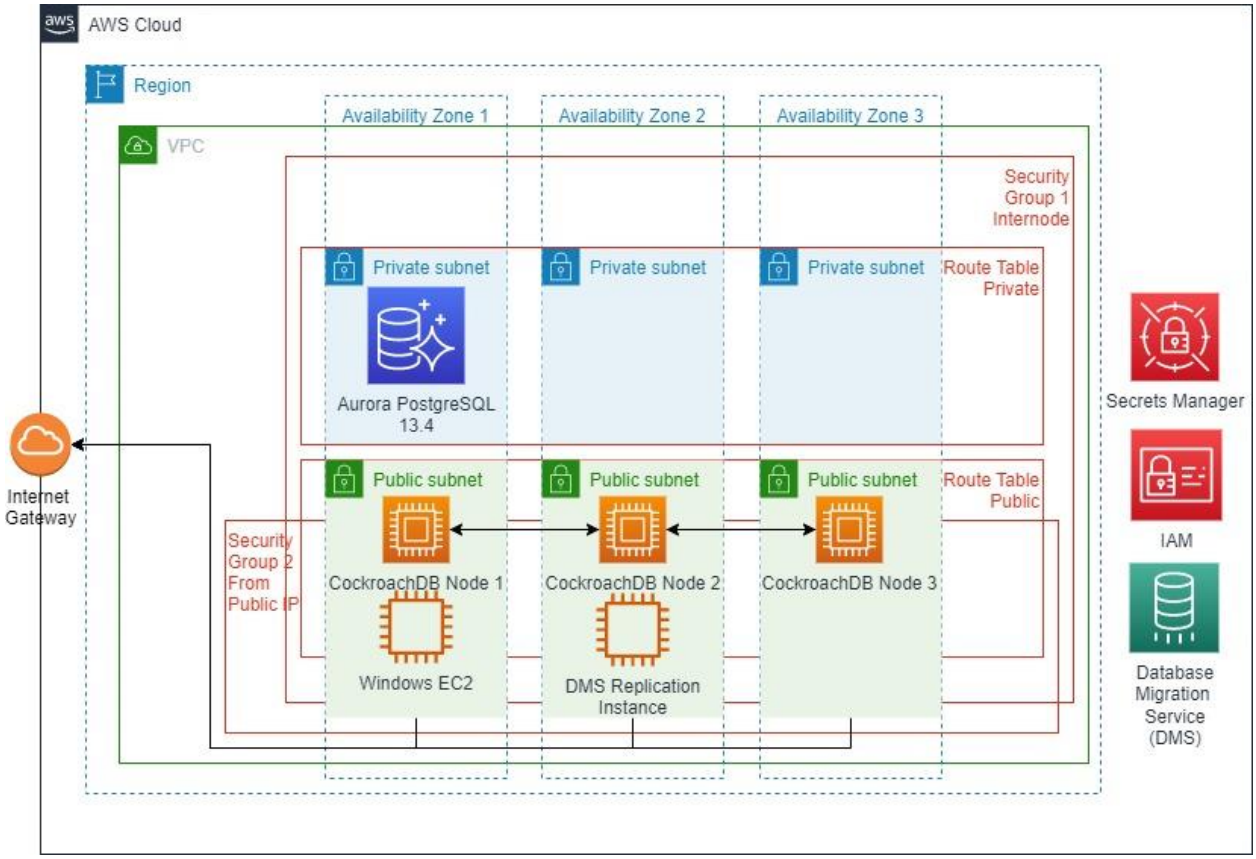
In this blog, I'm going to walk through (in tedious detail), how to set up SSL DMS between Aurora Postres and CockroachDB.

(By the way, unlike most of my peers at Cockroach Labs, I am working on a Windows Desktop, so you may have to map some of what I'm doing to your OS.)

Overview

Let's start with an overview of what we're going to build:

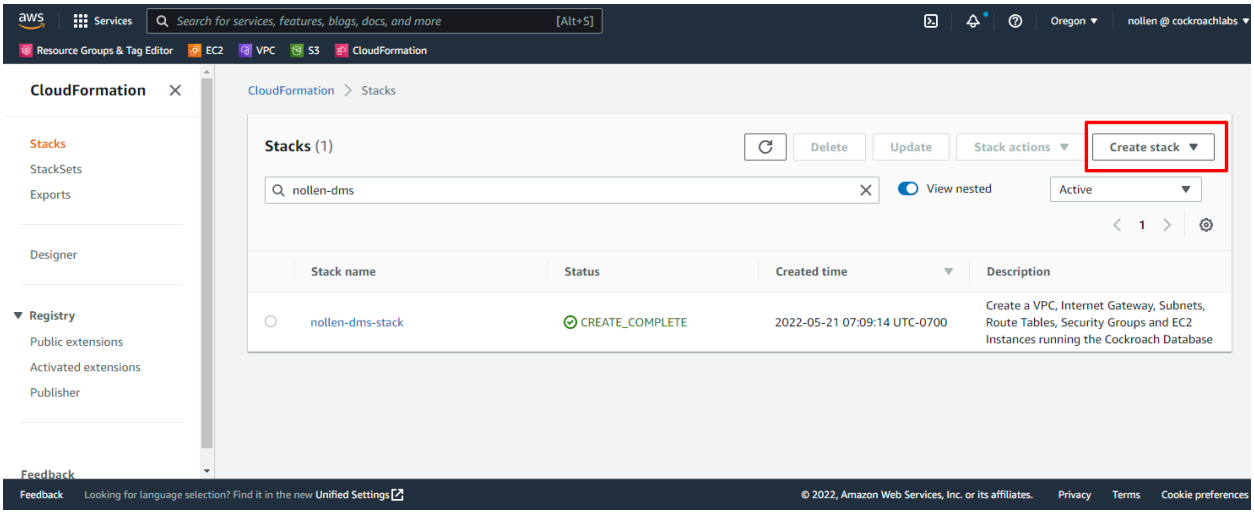




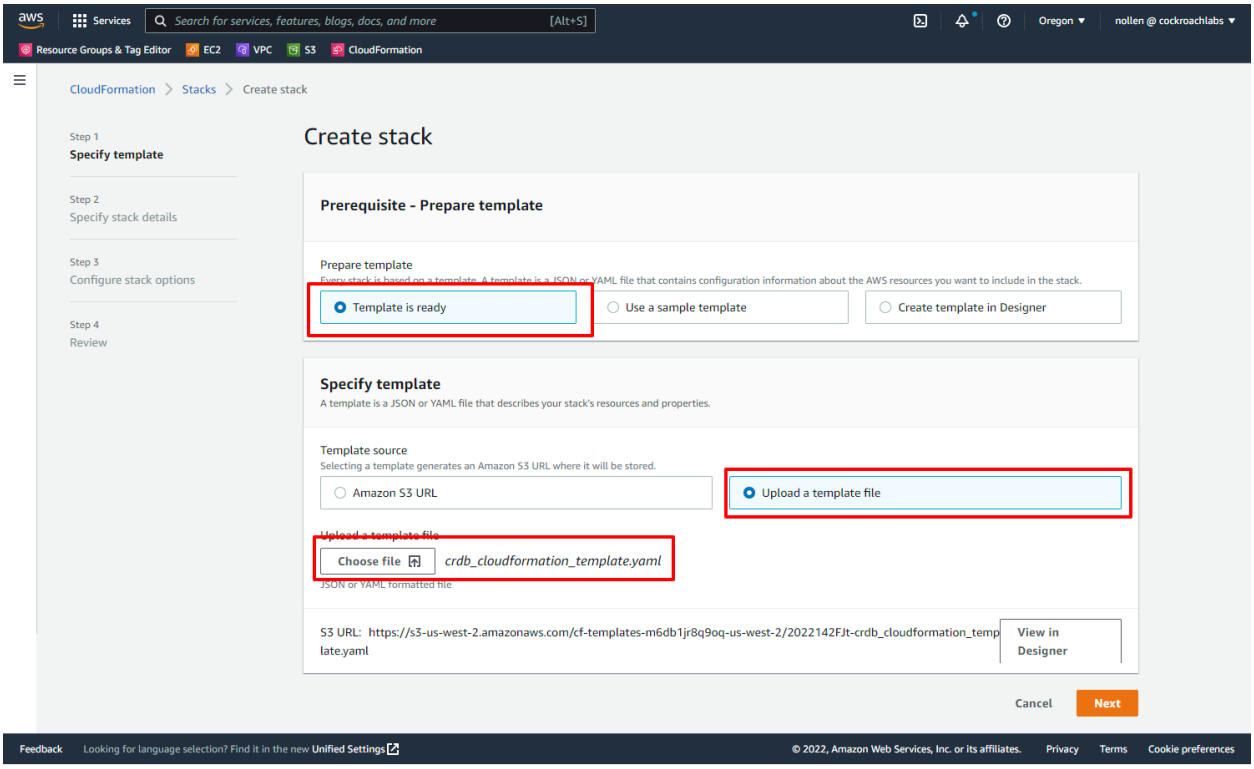
Our architecture includes a VPC where we will run both our Aurora Postgres Database in a private subnet along with a 3 node CockroachDB cluster in public subnets (please note, the CockroachDB could also be in the private subnets which in a production setting would be much more advisable). I'm using AWS Linux

In addition to the databases, I am also including a Windows server so that I can run things like DBeaver and connect to both databases as well the DMS Replication Instance. Since I placed the Aurora database in a private subnet, I will only be able to connect to the database from within the VPC so a Windows instance in the VPC make that very easy.

I have an AWS CloudFormation Script available here, which will create VPC, Internet Gateway, Subnets, RouteTables, Security Groups EC2 Instances and the CockroachDB Cluster including all the necessary certs and locality settings – basically everything you see in the diagram above except for the Aurora Database, Windows Instance and DMS Replication Instance. To use the CloudFormation template, check out the github link. Briefly, navigate to the CloudFormation page in AWS, select create stack [With new resources (standard)]:



On the next page, choose to use the crdb_cloudformation_template.yaml available in the github repo.

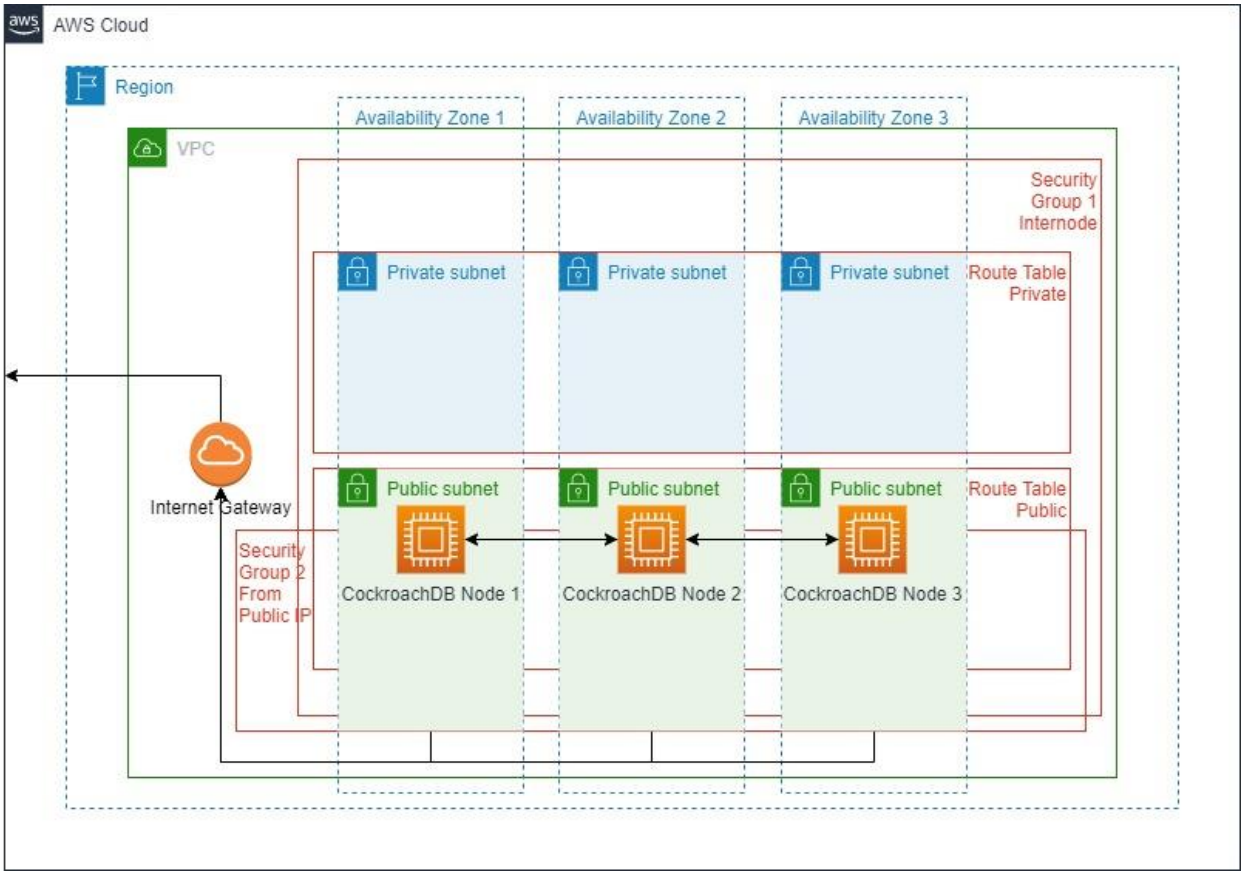


From there, you'll be able to input the parameters to create the architecture to your specification. The repo has information on the parameters, and what you'll need to set up before using the CloudFormation Template (such as a Key-Pair for the region in which you are working). Here are the parameters I used to create this CockroachDB Cluster:

CRDBAMIID	ami-00f7e5c52c0f43726
ClusterName	nollen-dms-target-cluster

CockroachVersion	21.2.10
ExistingJoinString	NONE
Installpsql	YES
InstanceType	m4.large
KeyPairName	nollen-cockroach-us-west-2-kp01
MyIP	72.132.195.192
NumberOfNodes	3
RunInit	YES
VpcAzs	us-west-2a,us-west-2b,us-west-2c
VpcCidrParameter	192.168.5.0/24
VpcName	nollen-dms-vpc-us-west-2

At this point, I'm going to assume you have the following:



- The VPC in the AWS region of your choice
- At least 3 Availability Zones (AZ) in the VPC
- 3 public and 3 private subnets (1 public and 1 private subnet in each AZ)
- Route Tables for the public and private subnets. The public route table should include an internet gateway so that instances in the public subnet can reach the internet).
- Security Groups. I have 2 security groups, one which allows all internode communicate within the VPC, and the other which allows only my IP to access instances in the public subnets on very

specific ports (3389 for RDP to my windows box, 8080 for access to the CockroachDB UI, 26257 for access to the CockroachDB and 22 for SSH access)

- CockroachDB Cluster: a 3 node CockroachDB Cluster running in 3 AZs

A few free tools that I will be using to access the different instances:

- putty
- winscp

The Cockroach Cluster – The Target for my DMS Migration

Here is a snaphot of my CockroachDB Cluster as created by my CloudFormation Template.

```
[ec2-user@ip-192-168-5-132 ~]$ cockroach node status
  id |             build              |             locality             | is_available | is_live
-----+--------------------------------+----------------------------------+--------------+--------
   1 | v22.1.0-beta.4-234-g5c9bf148ea | region=us-west-2,zone=us-west-2c | true         | true
   2 | v22.1.0-beta.4-234-g5c9bf148ea | region=us-west-2,zone=us-west-2b | true         | true
   3 | v22.1.0-beta.4-234-g5c9bf148ea | region=us-west-2,zone=us-west-2a | true         | true
(3 rows)
```

The database nodes are running Amazon Linux 2 Kernel 5.10 AMI 2.0.20211201.0 x86_64 HVM gp2 on m4.large instance types.

This is a 3 node cluster running CockroachDB Version 22.1.

Create a User and Password

Login to the cluster:

```
[ec2-user@ip-192-168-5-4 ~]$ cockroach sql
#
# Welcome to the CockroachDB SQL shell.
# All statements must be terminated by a semicolon.
# To exit, type: \q.
#
# Server version: CockroachDB CCL v22.1.0-rc.1 (x86_64-pc-linux-gnu, built
2022/05/09 14:20:38, go1.17.6) (same version as client)
# Cluster ID: d491e7b3-6cf7-43b2-a507-49ed1470710a
#
# Enter \? for a brief introduction.
#
root@:26257/defaultdb>
```

And create a user with admin privileges

```
root@:26257/defaultdb> create user ron with password "<redacted>";
CREATE ROLE


Time: 178ms total (execution 177ms / network 0ms)
root@:26257/defaultdb> grant admin to ron;
GRANT


Time: 92ms total (execution 92ms / network 0ms)
```

Create a DMS database

```
root@:26257/defaultdb> create database dms;
CREATE DATABASE


Time: 22ms total (execution 21ms / network 0ms)
```

Create a ca.pem file for SSL communication in DMS.

After creating the Cockroach Cluster, you'll need to generate a ca.pem file for SSL communication in DMS. Unfortunately, DMS will not work with ".crt" (certificates), so we must convert the cert to a ".pem" or base-64 encoding of the certificate.

To create the ".pem", use putty (or the tool of your choice) to log onto one of the nodes of your CockroachDB cluster, navigate to the "certs" directory and issue the following:

```
[ec2-user@ip-192-168-5-4 certs]$ openssl x509 -in ca.crt -out ca.pem -outform
PEM
```

Now, you'll need to use winscp (or the tool of your choice) to download that file. We'll upload this file to DMS later in order to establish SSL connectivity.

While you're at it, also download the ca.crt if you plan on using DBeaver to connect to your secure CockroachDB Cluster.

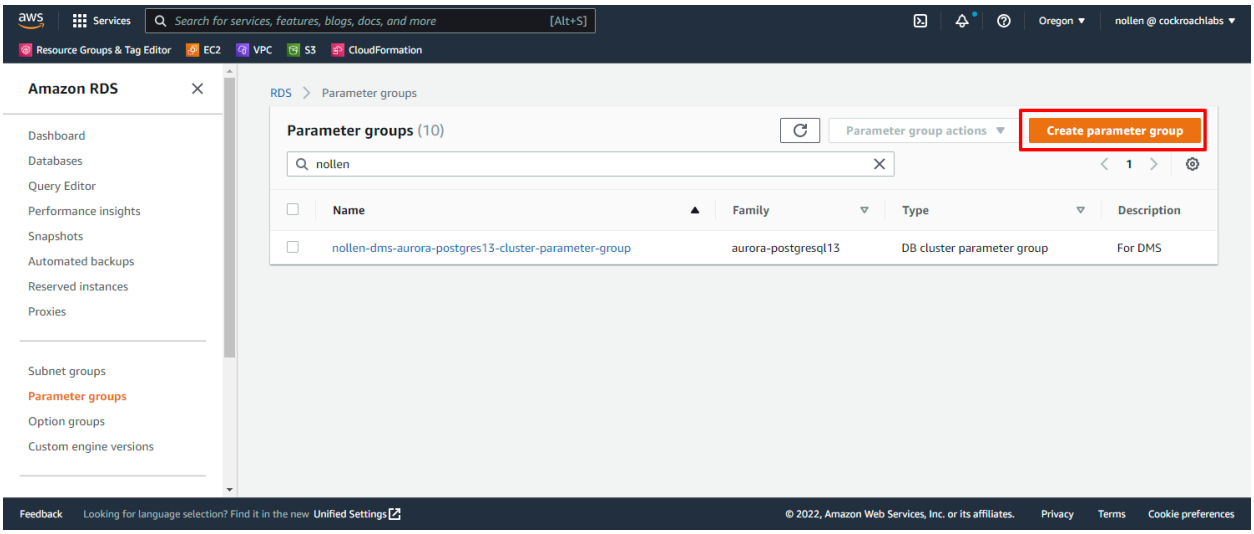
Create the Aurora Database Instance – The Source Database for my DMS Migration

Chances are you already have an Aurora PostgreSQL database, but there may be some modifications you may need to make to enable DMS to use the database as a source for a migration or replication. We'll walk through those steps now.

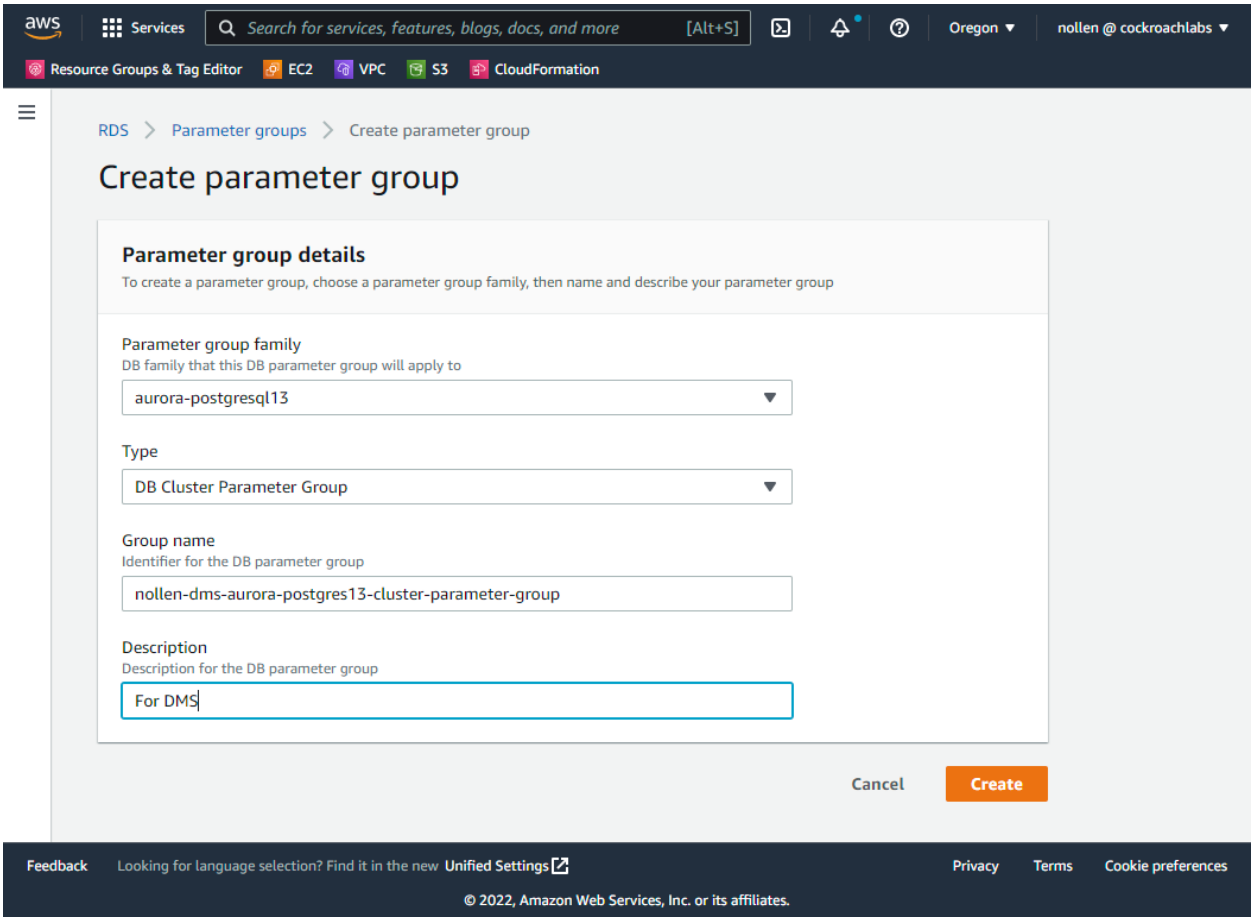
Aurora Parameter Group

In order for Aurora to be a source for DMS, we need to enable replication. To do that, we'll need to create a custom cluster parameter group. AWS Documentation on the requirements are available here.

We'll create the parameter group by navigating to AWS RDS, choose "Parameter Groups" from the list on the left and then "Create Parameter Group":

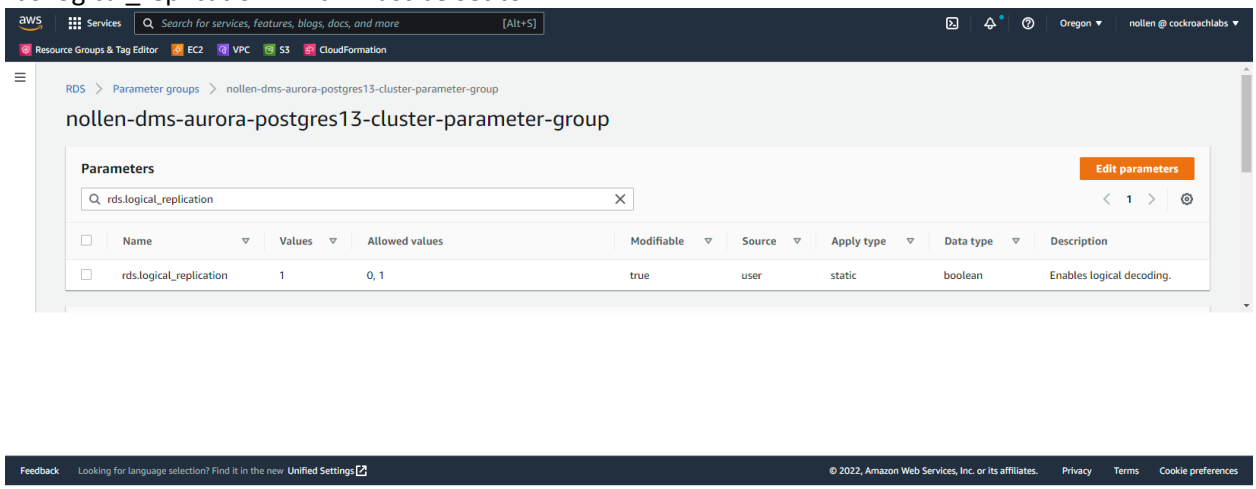


We need to create a parameter group for the version of Aurora PostgreSQL we going to create and the type is “DB Cluster Parameter Group”:

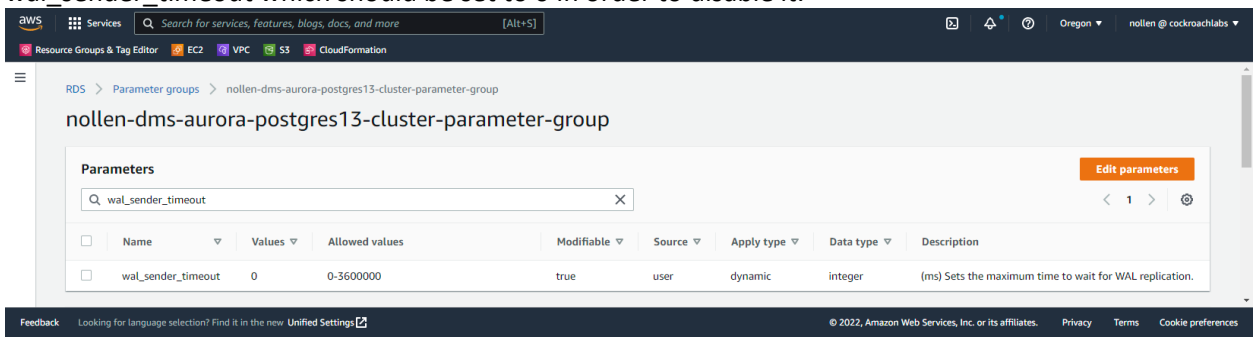


There are at least 2 parameters we need to change:

- rds.logical_replication which must be set to “1”



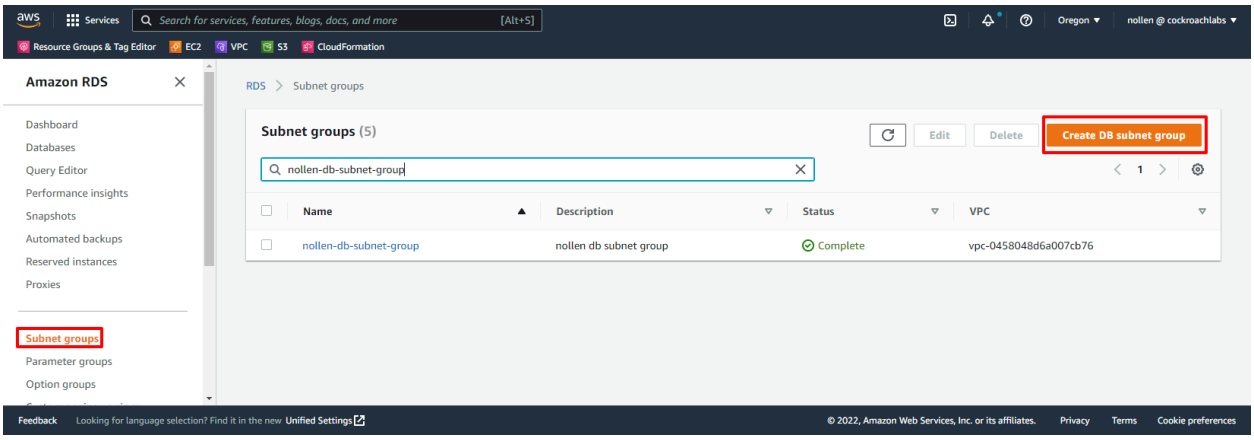
- wal_sender_timeout which should be set to 0 in order to disable it.



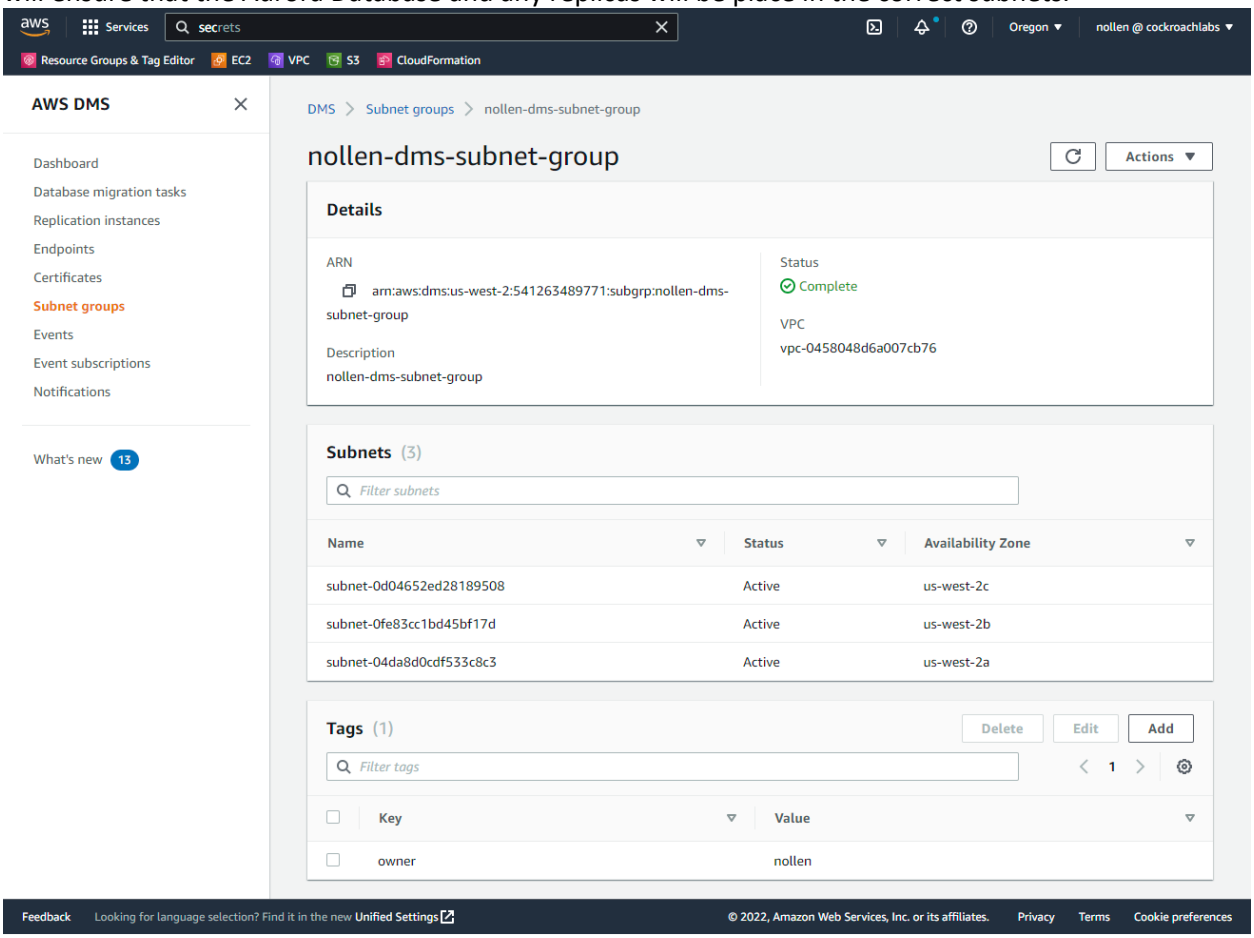
Again, please reference the AWS Documentation mentioned above for the background on these parameters and why they must be set to the values shown here.

Create the DB Subnet Group

Prior to creating the Aurora PostgreSQL database, we should also create a DB Subnet Group. To do that, navigate to RDS, select “Subnet groups” from the left hand panel and select “Create DB subnet group”.



When creating your subnet groups, but sure to select the VPC we are using for this implementation, select multiple Availability Zones and then select the Private Subnets from the Drop Down List Box. This will ensure that the Aurora Database and any replicas will be place in the correct subnets.



Create the Aurora PostgreSQL Database

I won't go into too much depth on creating the RDS Aurora PostgreSQL database other than to highlight a couple of important things (a summary of my selections when I created the database are shown below).

In the "Engine options" be sure to select "Amazon Aurora" and from the Edition Radio buttons be sure to select "Amazon Aurora PostreSQL – Compatible Edition. For the Version, select "Aurora PostgreSQL (Compatible with PostgreSQL13.4). I would suspect that other versions (10 and higher) would also work.

In the "Connectivity" section, be sure to select the correct VPC (the one we're currently working in) and in the "Subnet group", select the DB subnet group we created in the above step. For "Public Access" select "No". For VPC security Group, select "Choose existing" and use the 2 security groups we've been using throughout this blog.

Under "Additional Configuration" you can select "defaultdb" as the "initial database name" to make things easier.

AWS Services Search for services, features, blogs, docs [Alt+S] Oregon nolan @ rockmantaba

Resource Groups & Tag Editor EC2 VPC S3 CloudFormation

RDS > Create database

Create database

Choose a database creation method Info

- Standard create
  You set all of the configuration options, including ones for availability, security, backups, and maintenance.
- Easy create
  Use recommended best practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

- Amazon Aurora
- MySQL
- MariaDB
- PostgreSQL
- Oracle
- Microsoft SQL Server

Edition

- Amazon Aurora MySQL-Compatible Edition
- Amazon Aurora PostgreSQL-Compatible Edition

Engine version Info

View the engine versions that support the following database features.

▶ Show filters

Available versions (22/22) Info

Aurora PostgreSQL (Compatible with PostgreSQL 13.4)

Templates

Choose a sample template to meet your use case.

- Production
  Use defaults for high availability and fast, consistent performance.
- Dev/Test
  This instance is intended for development use outside of a production environment.

Settings

DB cluster identifier Info

Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

nolion-dms-source-aurora-db

For DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "myDBcluster"). Constraints: 1 to 63 alpha-numeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username Info

Type a login ID for the master user of your DB instance.

postgres

1 to 16 alpha-numeric characters. First character must be a letter.

☐ Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password.

Master password Info

••••••••••••••••••••

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password Info

••••••••••••••••••••

Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB instance class Info

- Memory optimized classes (includes r classes)
- Burstable classes (includes t classes)

db.t3.medium
2 vCPU 4 GiB RAM Network: 2,085 Mbps

☐ Include previous generation classes

Availability & durability

Multi-AZ deployment Info

- Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
  Creates an Aurora Replica for fast failover and high availability.
- Don't create an Aurora Replica

Connectivity

Virtual private cloud (VPC) Info

VPC that defines the virtual networking environment for this DB cluster.

nolion-dms-tpc-us-west-2-tpc-0458f4b6d07c7f8

Only VPCs with a corresponding DB subnet group are listed.

ⓘ After a database is created, you can't change its VPC.

Subnet group Info

DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

nolion-db-subnet-group

Public access Info

- Yes
  Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.
- No
  RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

VPC security group

Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

- Choose existing
  Choose existing VPC security groups
- Create new
  Create new VPC security group

Existing VPC security groups

Choose VPC security groups

nolion-dms-stack-InstanceSecurityGroup2-1RJVSWWUBMC4K ✕

nolion-dms-stack-InstanceSecurityGroup2-7QIBB2TEHK9G ✕

default ✕

Availability Zone Info

No preference

▶ Additional configuration

Babelfish settings - new Info

☐ Turn on Babelfish
Makes porting T-SQL, T-Unique, and lower-risk migrations from Microsoft SQL Server to Aurora PostgreSQL.

ⓘ **PostgreSQL version selected does not include the latest Babelfish version**
The latest Babelfish capabilities, including enhancements to SQL client support, require a newer PostgreSQL version. Select the latest version of Aurora PostgreSQL to support the newest Babelfish version. Learn more about Babelfish versions.

Database authentication Info

Password authentication is always enabled for your DB engine. You can enable additional authentication methods for your database below.

☐ IAM database authentication
Authenticates using IAM database authentication.

☐ Kerberos authentication
Authenticates using Kerberos authentication through an AWS Directory Service for Microsoft Active Directory.

▶ Additional configuration

Database options, encryption turned on, failover, backup turned on, backtrack turned off, Performance Insights turned on, Enhanced Monitoring turned on, maintenance, CloudWatch Logs, delete protection turned on

ⓘ You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

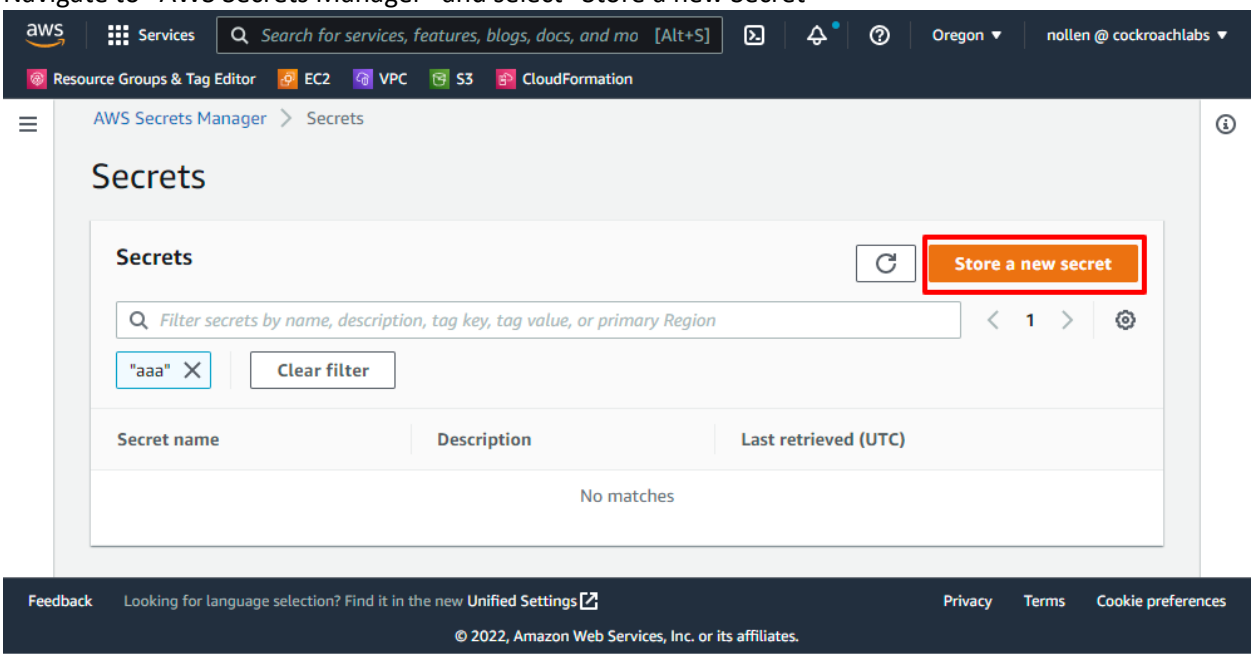
Cancel **Create database**

Configure Secrets Manager

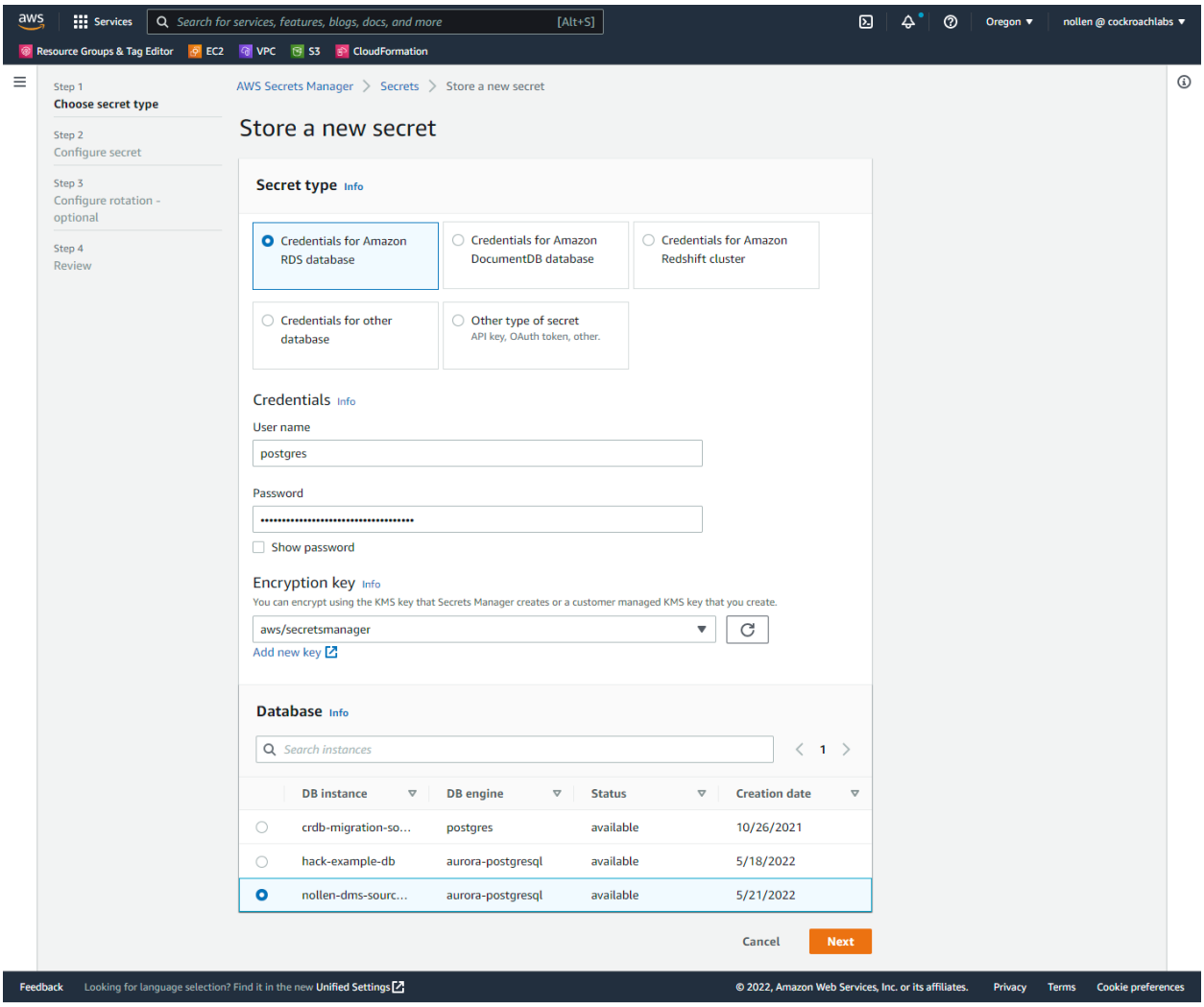
To make configuration with DMS easier, we're going to create a couple of secrets in Secret manager for connecting to our databases (both Aurora and CockroachDB).

Store a Secret for Aurora

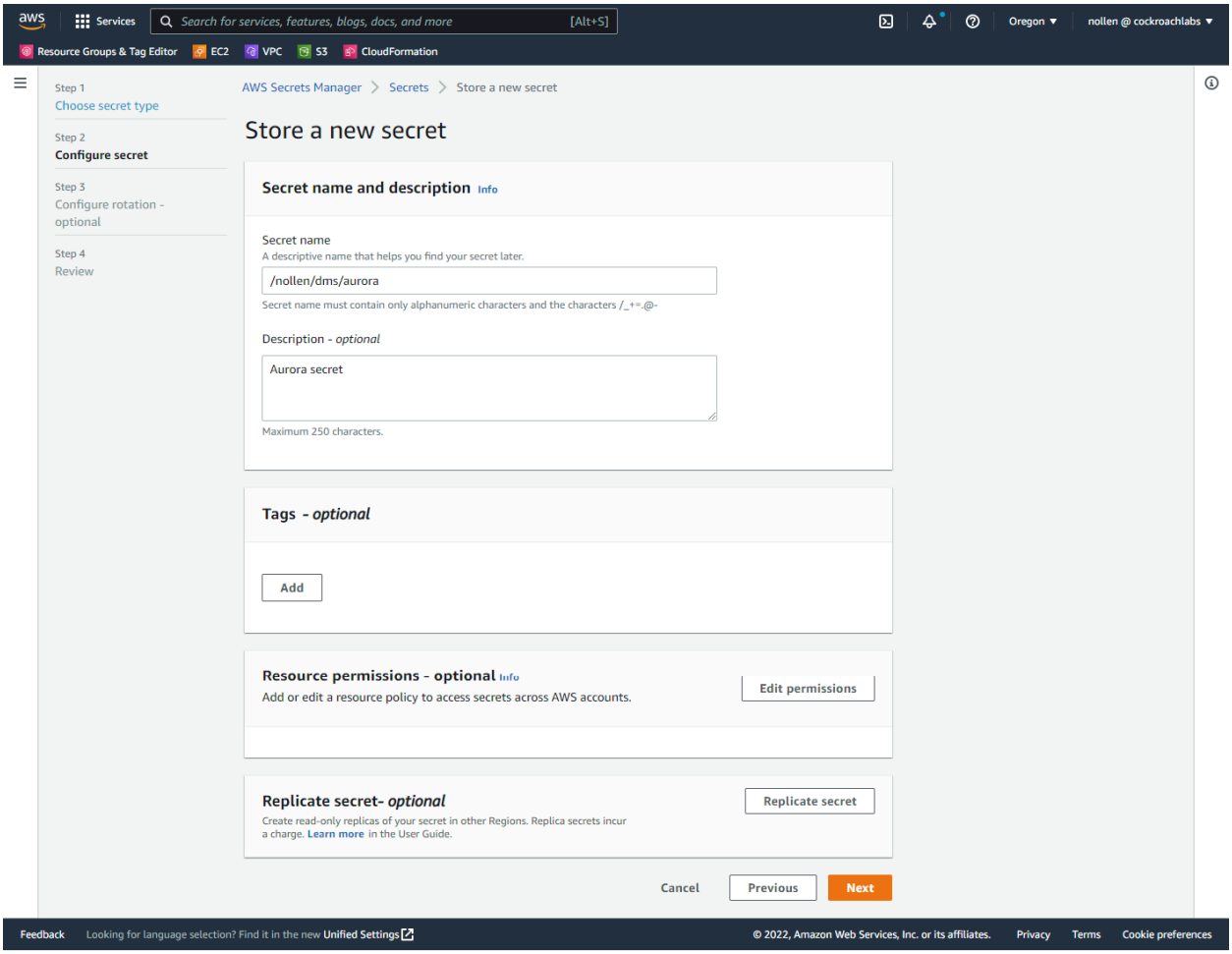
Navigate to "AWS Secrets Manager" and select "Store a new Secret"



Enter the information requested on page 1 of "Store a New Secret":



Continue to enter information for your Aurora PostgreSQL database:



You can leave the defaults on page 3 and select “next”. And then click “store” on the “Review” page.

When the secret is created, it will automatically contain all of the necessary connection information including the Username and Password you entered:

Secret value Info

Retrieve and view the secret value.

Close | Edit

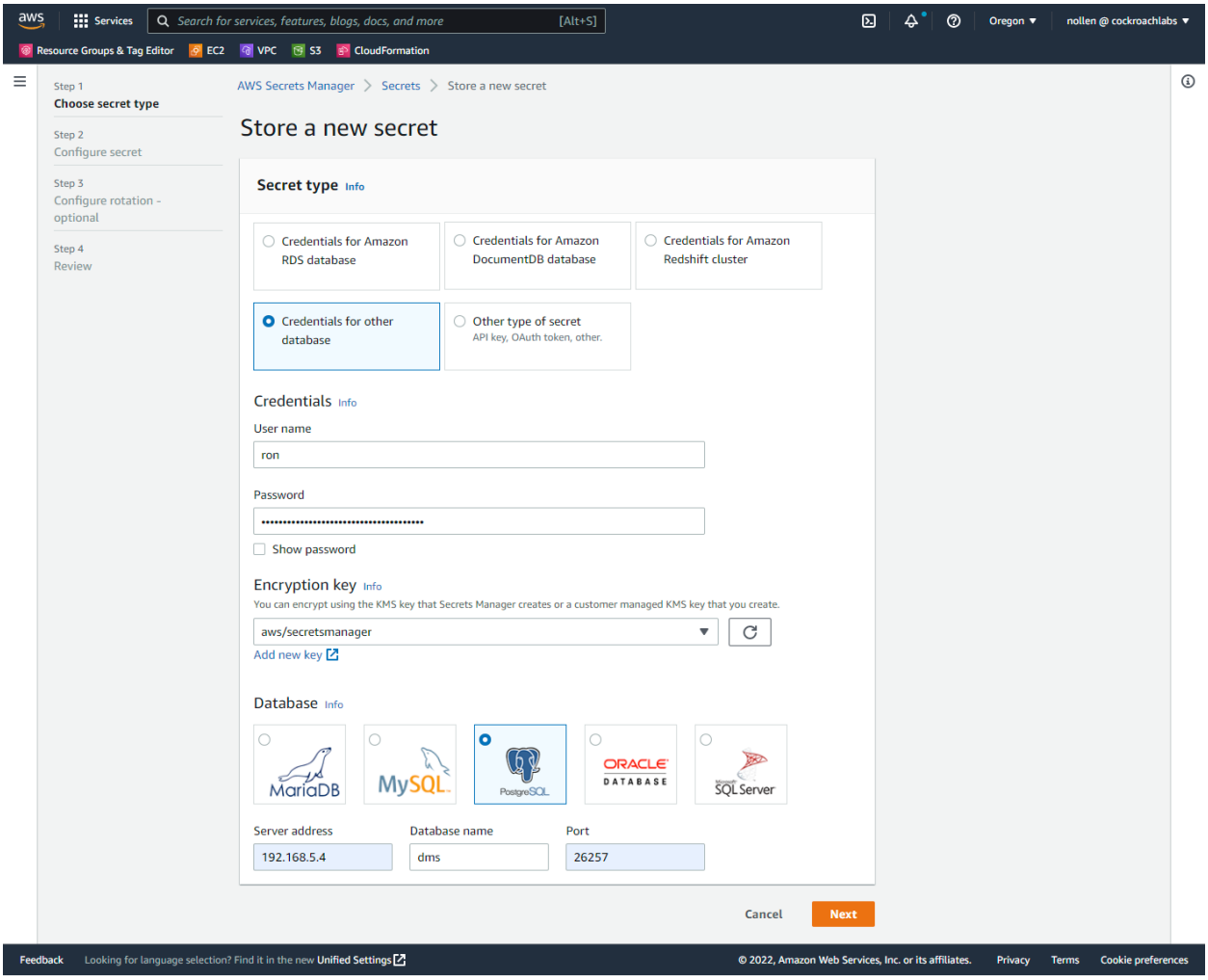
Key/value | Plaintext

Secret key	Secret value
username	postgres
password	[redacted]
engine	postgres
host	nollen-dms-source-aurora-db.cluster-c24vvqv1mozy.us-west-2.rds.amazonaws.com
port	5432
dbClusterIdentifier	nollen-dms-source-aurora-db

Secrets manager was able to get this information for you directly from RDS without you having to enter it.

Store a Secret for CockroachDB

The process for storing a Secret for CockroachDB will basically be the same as above, except that you'll need to provide the Server Address, Database name and Port. To get started, select "Credentials for other database" under "Secret Type", choose PostgreSQL as the database and then enter the requested information. DMS will use this to create a connection to your Cockroach Database.



This is what my completed secret for CockroachDB looks like:

aws | Services | Search for services, features, blogs, docs, and more [Alt+S] | Oregon | nollen @ cockroachlabs

Resource Groups & Tag Editor | EC2 | VPC | S3 | CloudFormation

AWS Secrets Manager > Secrets > /nollenr/dms/cockroachdb

/nollenr/dms/cockroachdb

Replicate secret to other Regions

Secret details

Actions

Encryption key
aws/secretsmanager

Secret name
/nollenr/dms/cockroachdb

Secret ARN
arn:aws:secretsmanager:us-west-2:541263489771:secret:/nollenr/dms/cockroachdb-SCpuiM

Secret description
-

Tags

Edit tags

Find by key or value...

Key	Value
owner	nollen

Secret value Info

Retrieve and view the secret value.

Close | Edit

Key/value | Plaintext

Secret key	Secret value
username	ron
password	[redacted]
engine	postgres
host	192.168.5.4
port	26257
dbname	dms

Rotation configuration Info

Edit rotation

Rotation status
Disabled

Rotation schedule
-

Resource permissions - optional Info

Add or edit a resource policy to access secrets across AWS accounts.

Edit permissions

Sample code

Use these code samples to retrieve the secret in your application.

Java | JavaV2 | JavaScript | C# | Python3 | Ruby | Go

```java
// Use this code snippet in your app.
// If you need more information about configurations or implementing the sample code, visit the AWS docs:
// https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/java-dg-samples.html#prerequisites

public static void getSecret() {

    String secretName = "arn:aws:secretsmanager:us-west-2:541263489771:secret:/nollenr/dms/cockroachdb-SCpuiM";
    String region = "us-west-2";

    // Create a Secrets Manager client
    AWSSecretsManager client  = AWSSecretsManagerClientBuilder.standard()
                                    .withRegion(region)
                                    .build();

    // In this sample we only handle the specific exceptions for the 'GetSecretValue' API.
    // See https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
    // We rethrow the exception by default.
```


Sample code

Use these code samples to retrieve the secret in your application.

Java | JavaV2 | JavaScript | C# | Python3 | Ruby | Go

```java
// Use this code snippet in your app.
// If you need more information about configurations or implementing the sample code, visit the AWS docs:
// https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/java-dg-samples.html#prerequisites

public static void getSecret() {

    String secretName = "arn:aws:secretsmanager:us-west-2:541263489771:secret:/nollenr/dms/cockroachdb-SCpuiM";
    String region = "us-west-2";

    // Create a Secrets Manager client
    AWSSecretsManager client  = AWSSecretsManagerClientBuilder.standard()
                                    .withRegion(region)
                                    .build();

    // In this sample we only handle the specific exceptions for the 'GetSecretValue' API.
    // See https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
```

Create the IAM Policy and Role

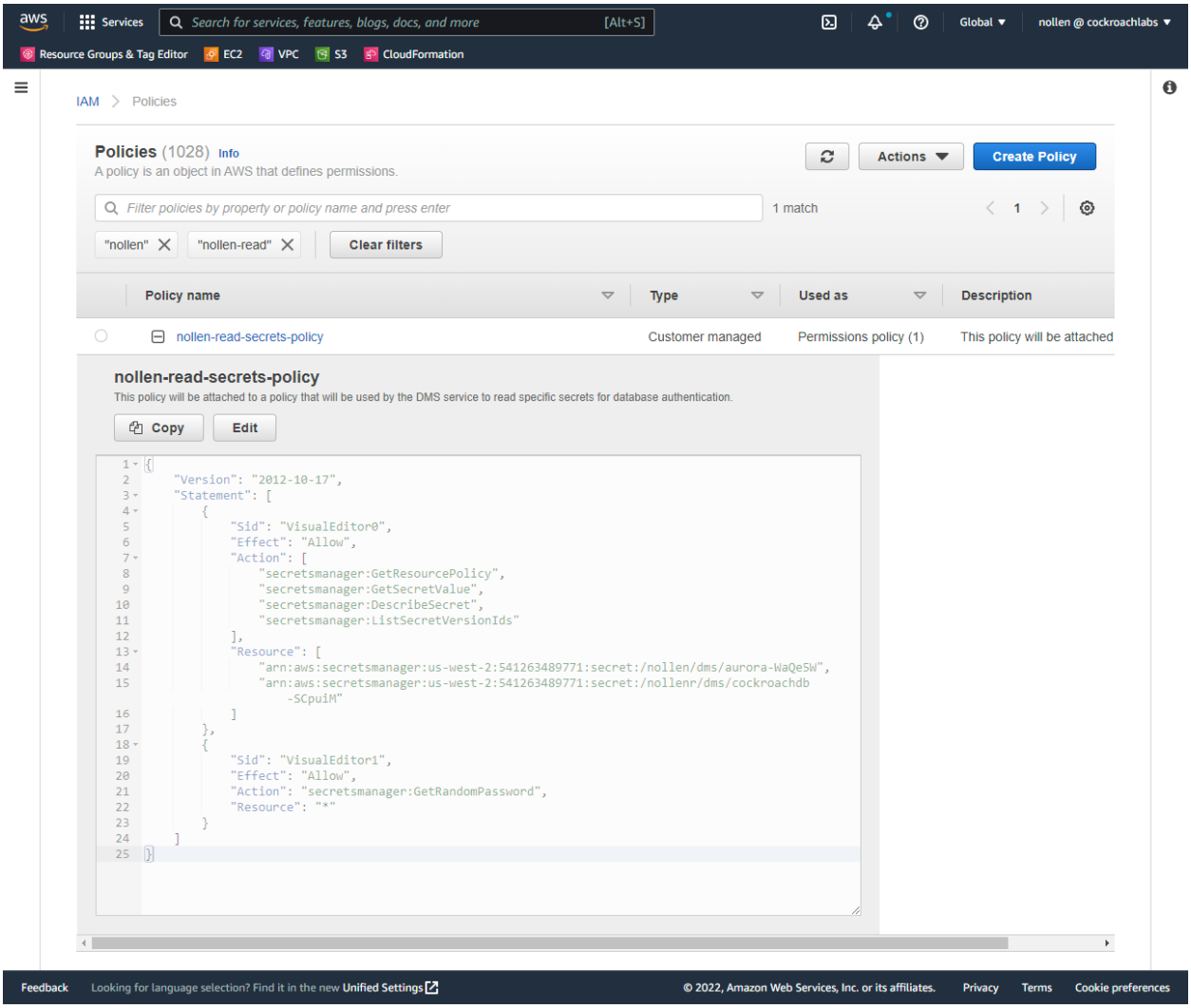
We need to create an IAM Policy and Role in order for DMS to be able to use the secrets we just created.

Create a Policy

Navigate to Identity and Access Management (IAM) and select “Policies” from the “Access Management” section on the left. Choose create policy

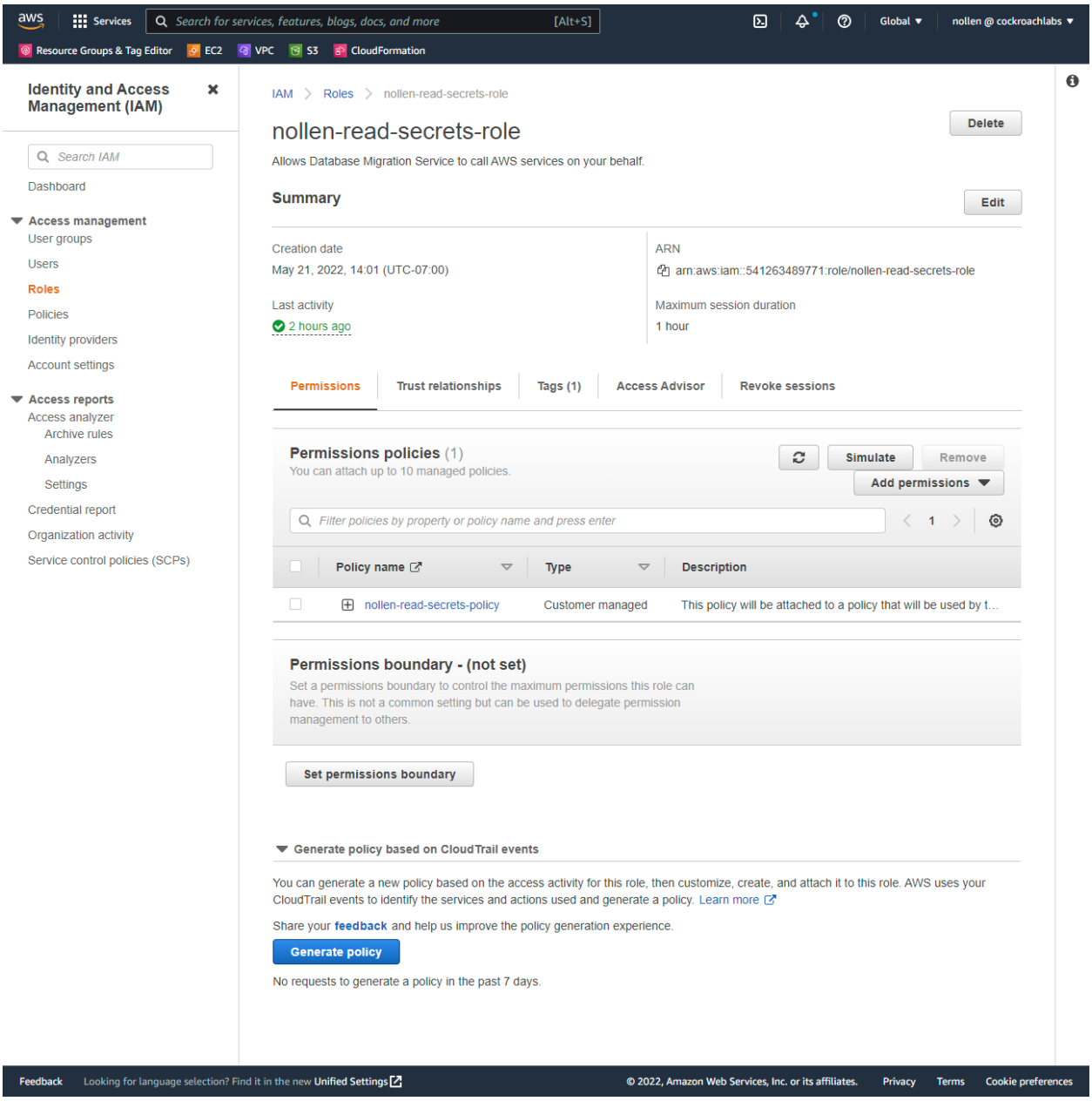
A policy (which I named “nollen-read-secrets-policy”) which works is:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-west-
2:541263489771:secret:/nollen/dms/aurora-WaQe5W",
                "arn:aws:secretsmanager:us-west-
2:541263489771:secret:/nollenr/dms/cockroachdb-SCpuiM"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "secretsmanager:GetRandomPassword",
            "Resource": "*"
        }
    ]
}
```



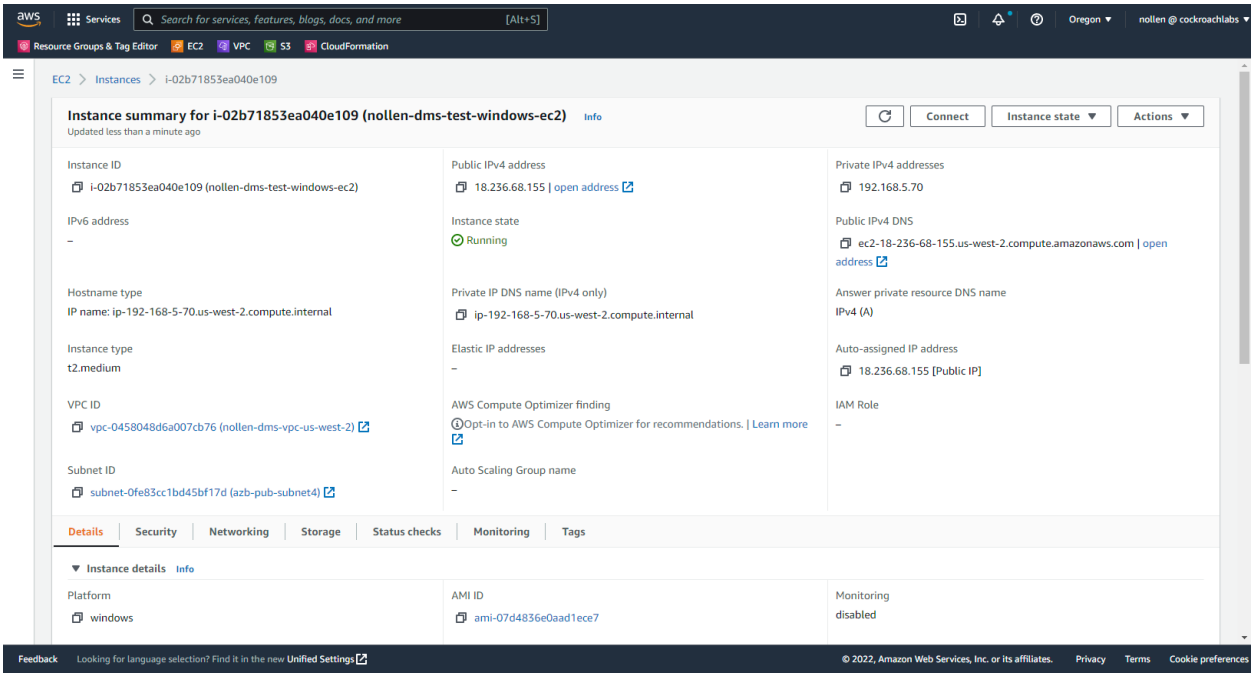
Create an IAM Role

Now we need to attach the policy we just created to a role. To do that, navigate to IAM and select "Roles" from the panel on the left. Choose "Create role" (which I named "nollen-read-secrets-role") and attach the policy we created above.



Create a Windows EC2 Instance (Optional) and Install DBeaver.

I created a windows instance in a public subnet (in any of the AZs of my VPC). I use this instance to install DBeaver and access my databases. I won't go into the details of creating an EC2 instance here, but below are some of the details of my instance:



Once I had my Windows instance and connected to it via RDP (use the “Connect” button on the EC2 Instance Screen to get the Username and Password), I installed DBeaver.

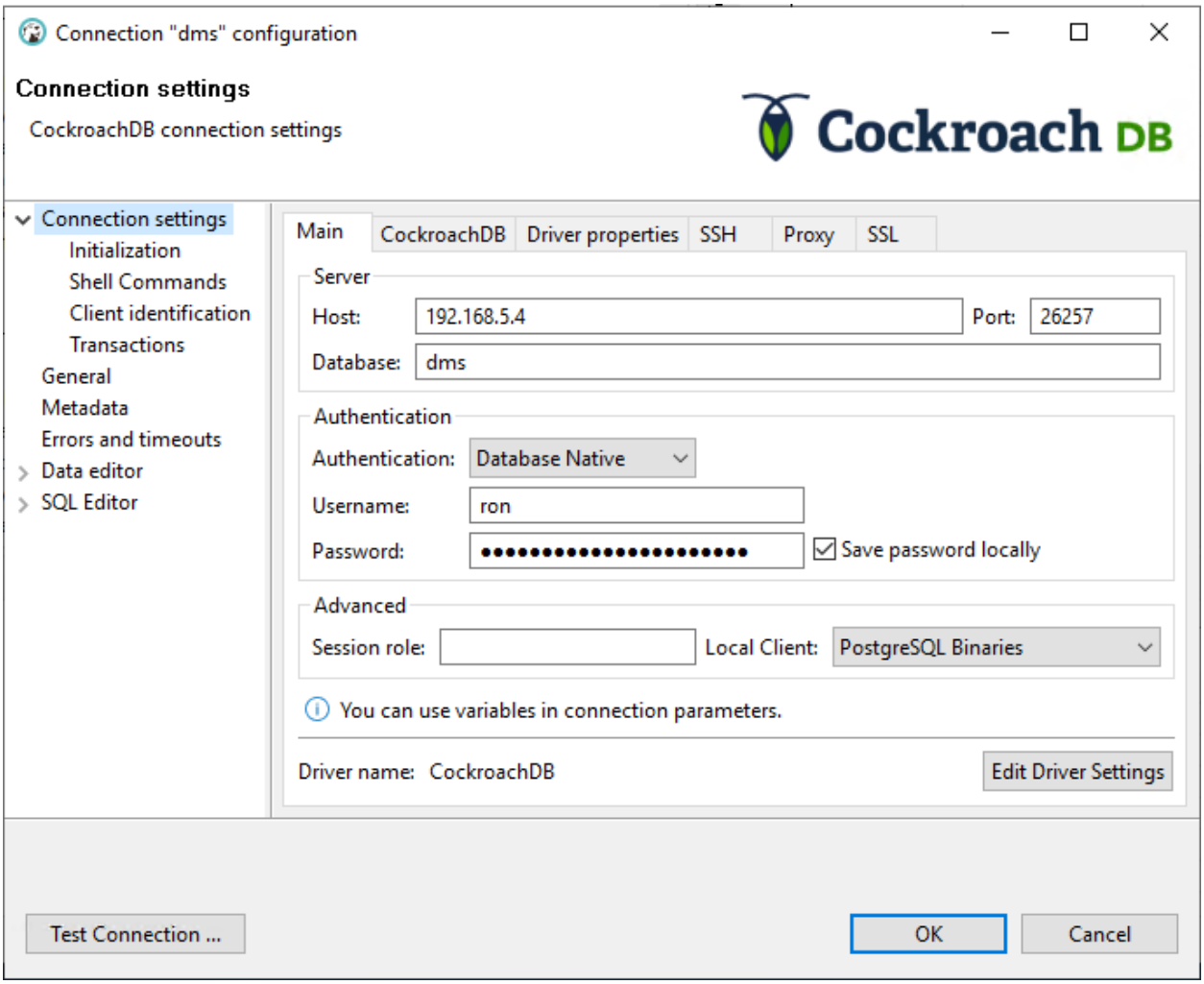
Connect to CockroachDB

I’m not going to go into installing DBeaver, but I think it might be helpful for me to explain how I connect to my Cockroach Instance from the utility.

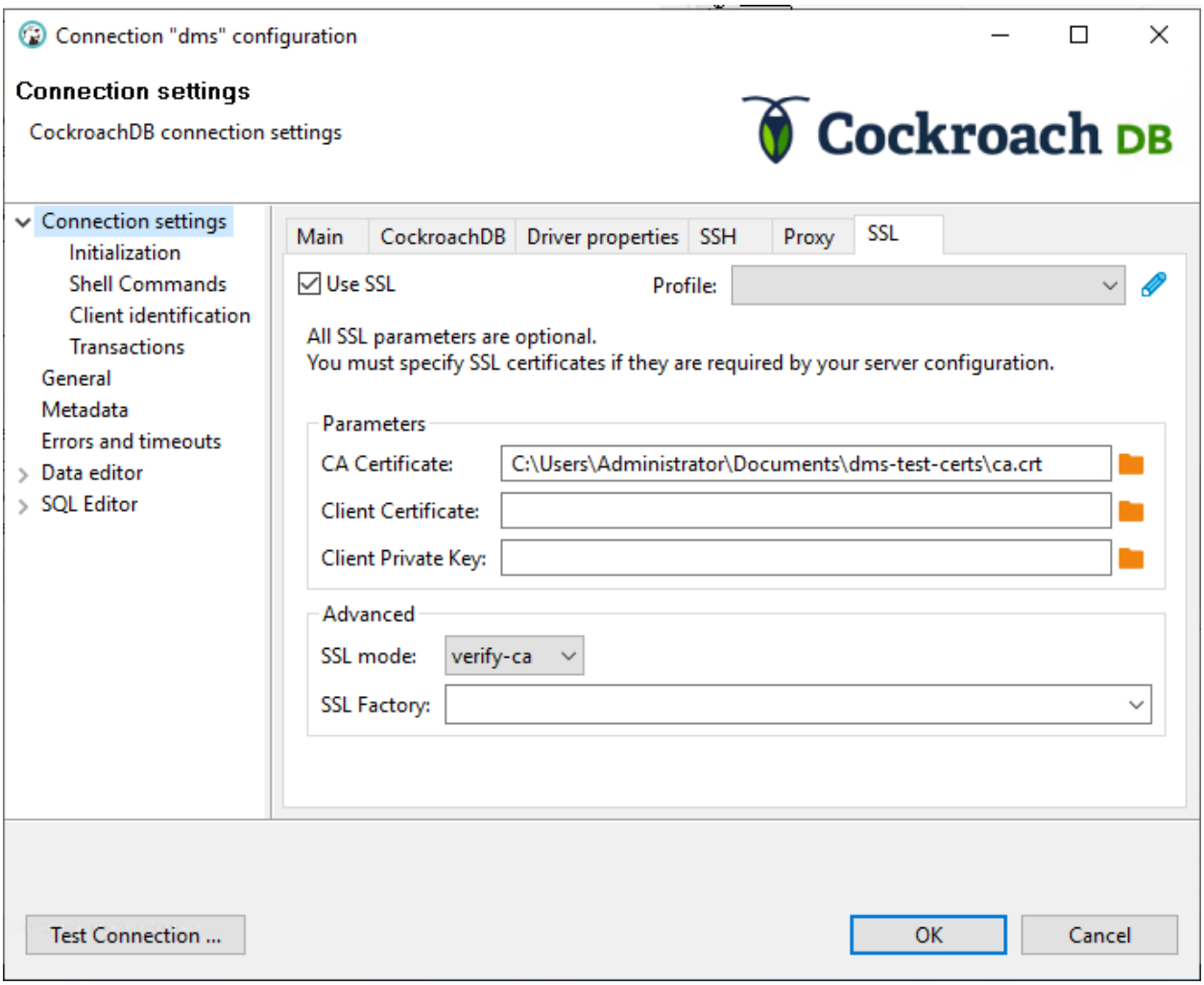
When you create a connection in DBeaver for CockroachDB, be sure to select “Cockroach” as the database type.

In the “Sever” section, you can choose any one of the 3 private IP addresses of the CockroachDB nodes; be sure to use Port 26257 and the DMS database.

Use the username and password we created above in the “Authentication” section.



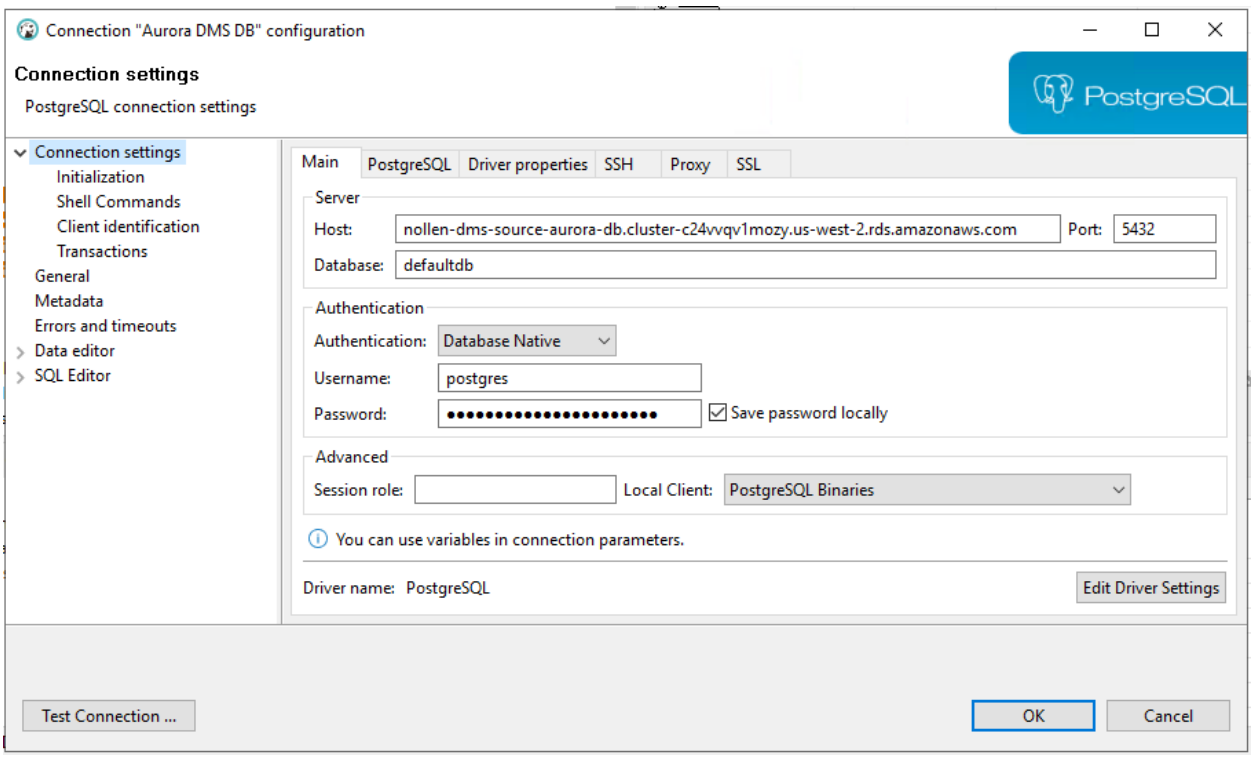
On the SSL Tab of the connection modal, be sure to use the ca.crt file we downloaded earlier as the parameter for “CA Certificate”. In the “advanced” section, choose “verify-ca” from the “SSL mode:” drop down list box.



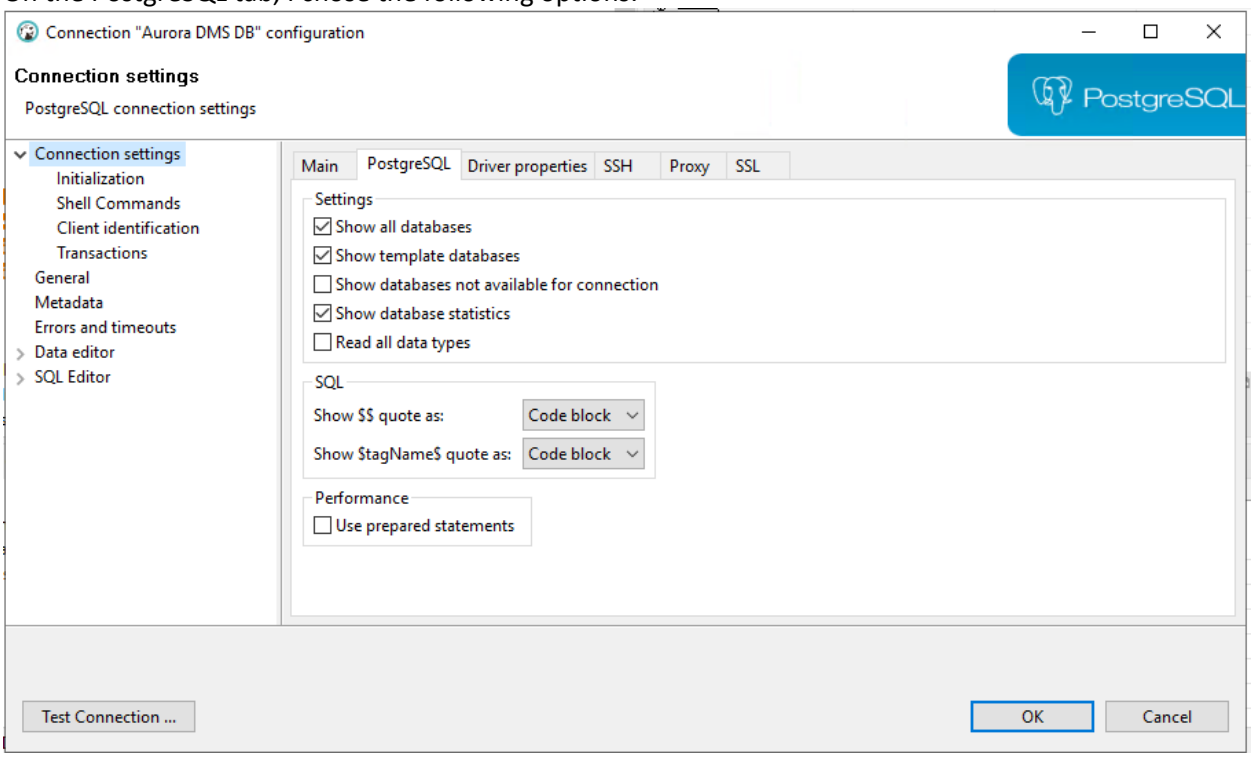
Be sure to test your connection.

Connect to Aurora PostgreSQL

To connect to the Aurora PostgreSQL database, you'll use the host provided by AWS when you created the instance and port 5432. You'll also use the master user and password you supplied when you created the instance.



On the PostgreSQL tab, I chose the following options:

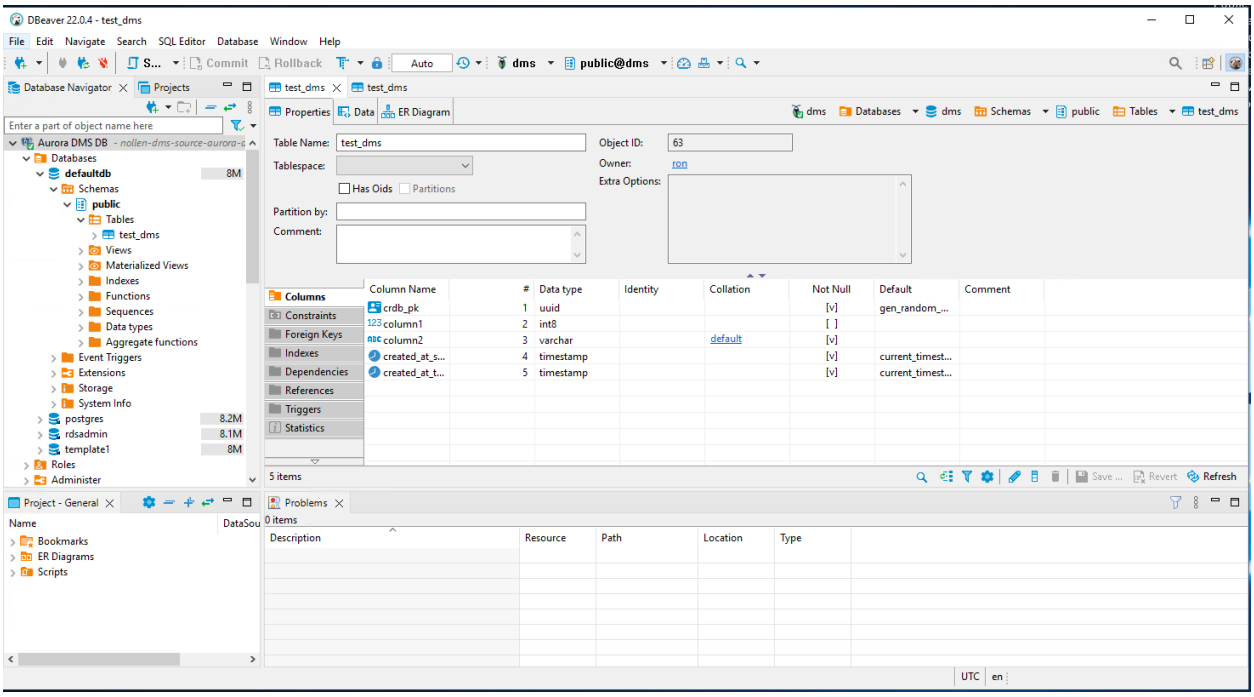


Configure DMS and Start On Going Replication

Create the PostgreSQL Source Table

Using DBeaver, I created the following table as my source in the “defaultdb” in Aurora PostgreSQL:

```
CREATE TABLE dms.public.test_dms (
	column1 serial4 NOT NULL,
	column2 varchar NOT NULL,
	created_at_source timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
	CONSTRAINT test_dms_pk PRIMARY KEY (column1)
);
```



Create the CockroachDB Target Table

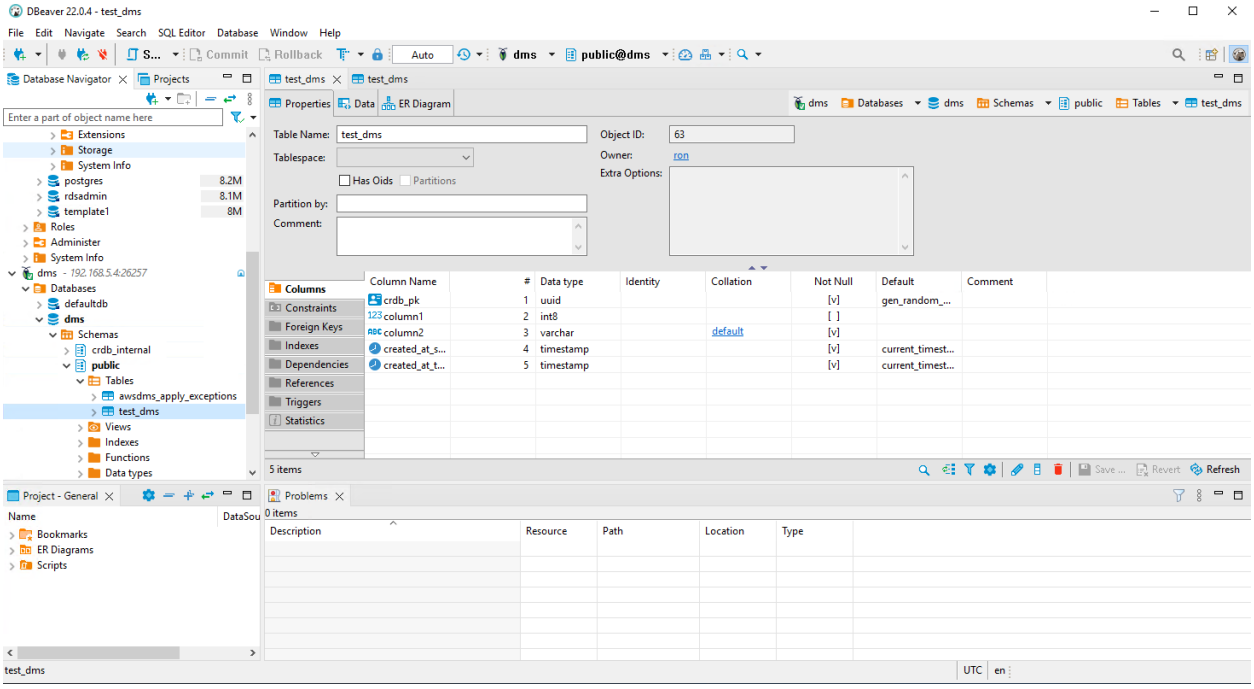
Using DBeaver, I created the following table as my target in the “dms” database of my CockroachDB Cluster

```
CREATE TABLE public.test_dms (
	crdb_pk UUID NOT NULL DEFAULT gen_random_uuid(),
	column1 INT8 NULL,
	column2 VARCHAR NOT NULL,
	created_at_source TIMESTAMP NOT NULL DEFAULT
current_timestamp():::TIMESTAMP,
	created_at_target TIMESTAMP NOT NULL DEFAULT
current_timestamp():::TIMESTAMP,
	CONSTRAINT test_dms_pkey PRIMARY KEY (crdb_pk ASC)
);
```

A couple of important notes:

- The DMS does not need me to create any mapping from the PostgreSQL table to the CockroachDB table even though the tables are considerable different. (Gotta love that!)

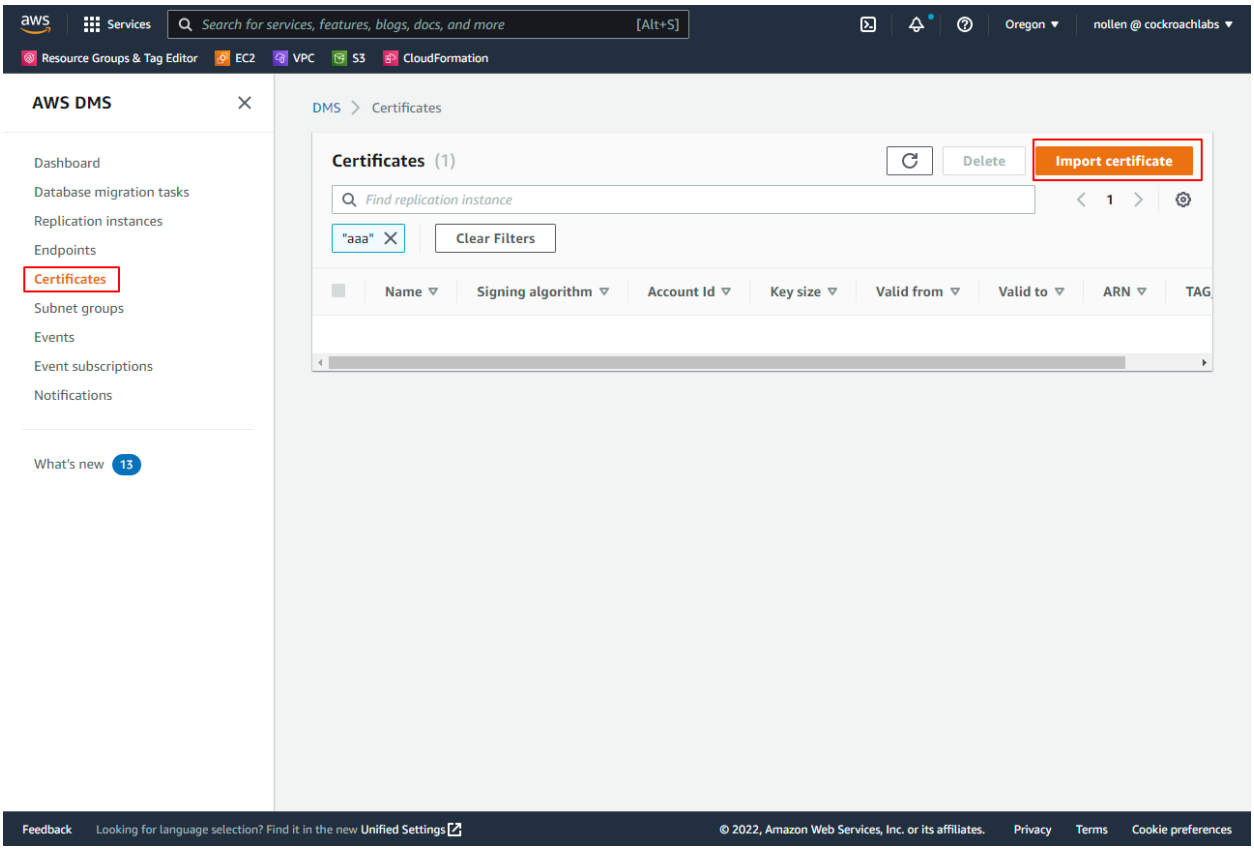
- We did not have to create a table in the target database. DMS will create the table automatically on our behalf if it does not exist.



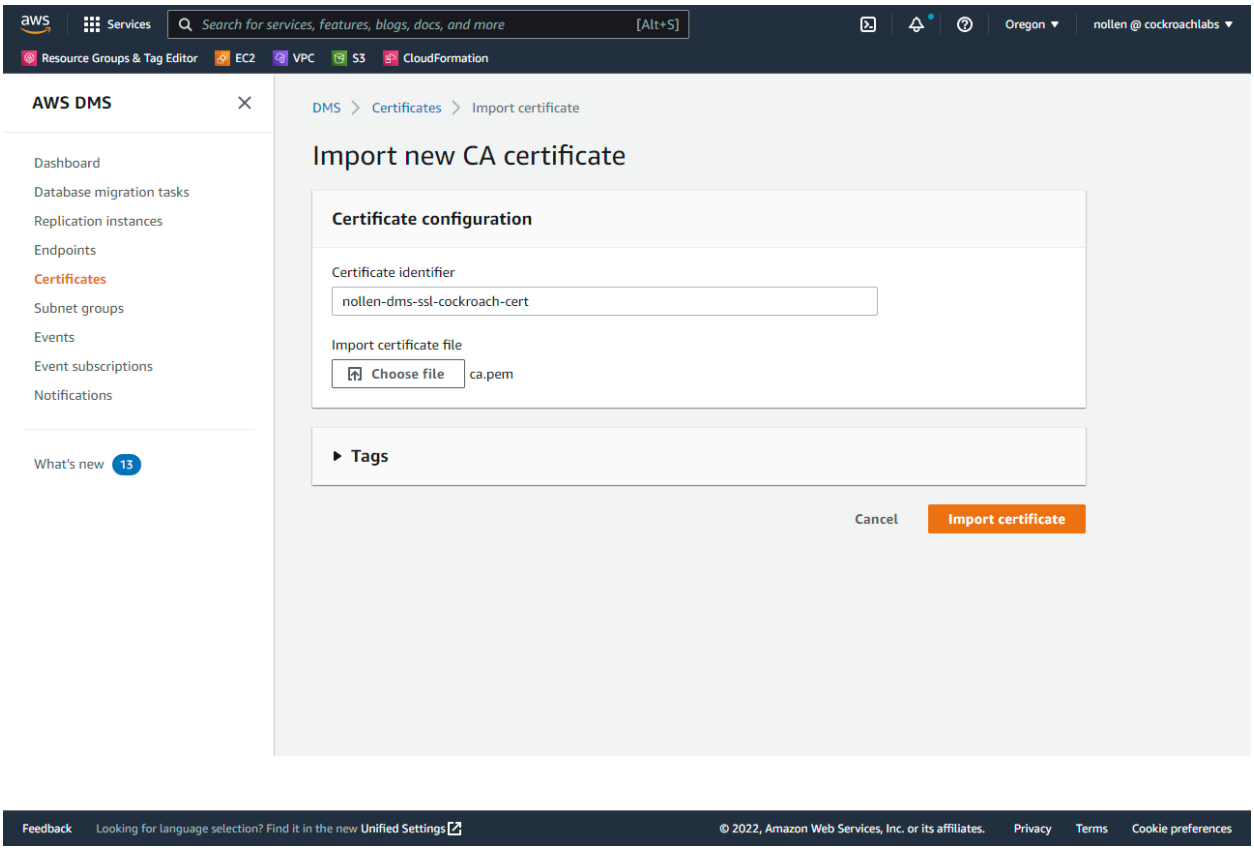
Create the Certificate

In order to have SSL communication to my CockroachDB, I'll need to upload the ".pem" cert I created earlier.

Navigate to the AWS DMS page and from the left-hand panel, choose "Certificates" and then select "Import certificate".



On the next page, give your certificate a name and then select the “Choose file” button to select the ca.pem file.

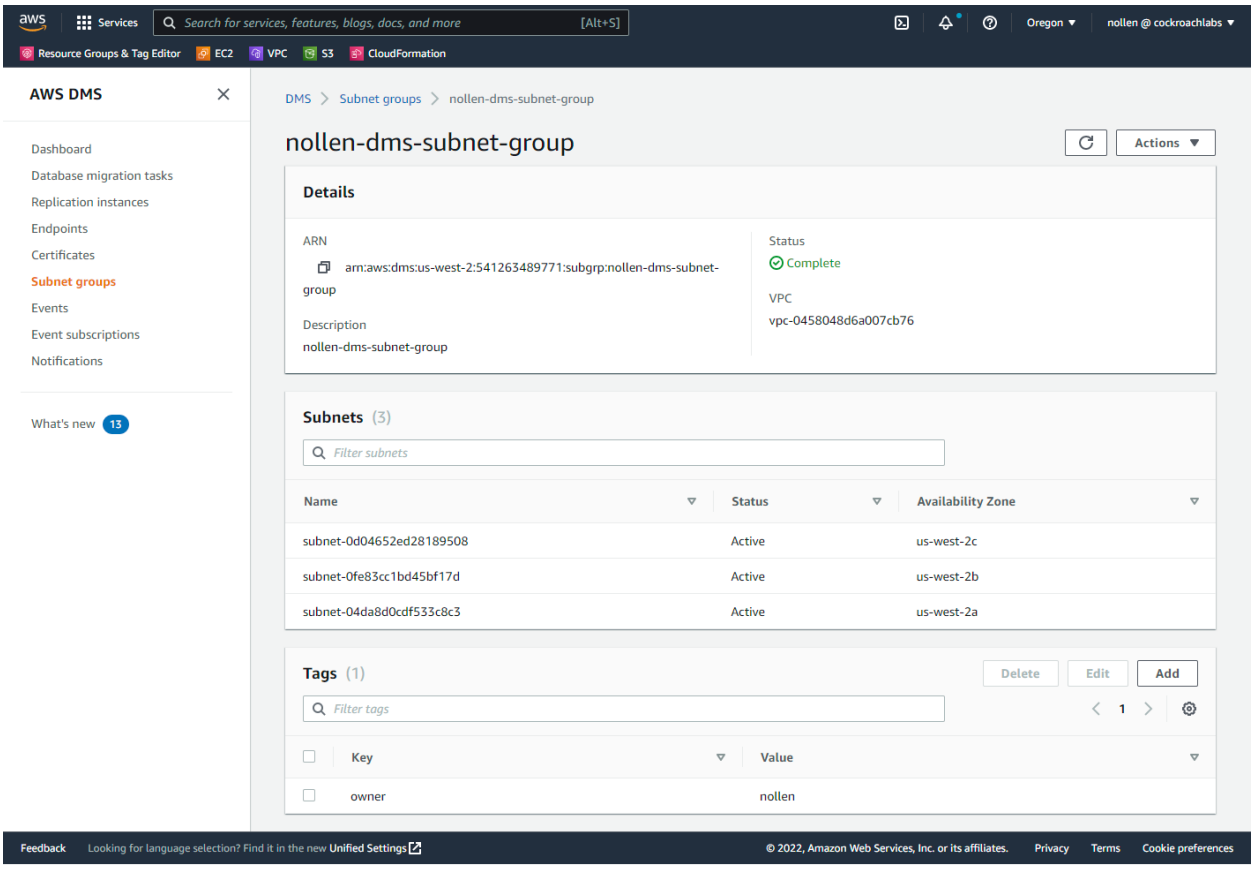


Create the Subnet groups

Much like we created subnet groups for the Aurora database, we should create a subnet group for the DMS Replication Instance.

To create the subnet group, navigate to AWS DMS and select “Subnet groups” from the left hand side and select “Create subnet group”.

Be sure to choose the same VPC we’ve been working in throughout this blog, and then create the subnets from the VPC into which we could place the replication instance. I’m going to choose the public subnets from my VPC to create the subnet group.

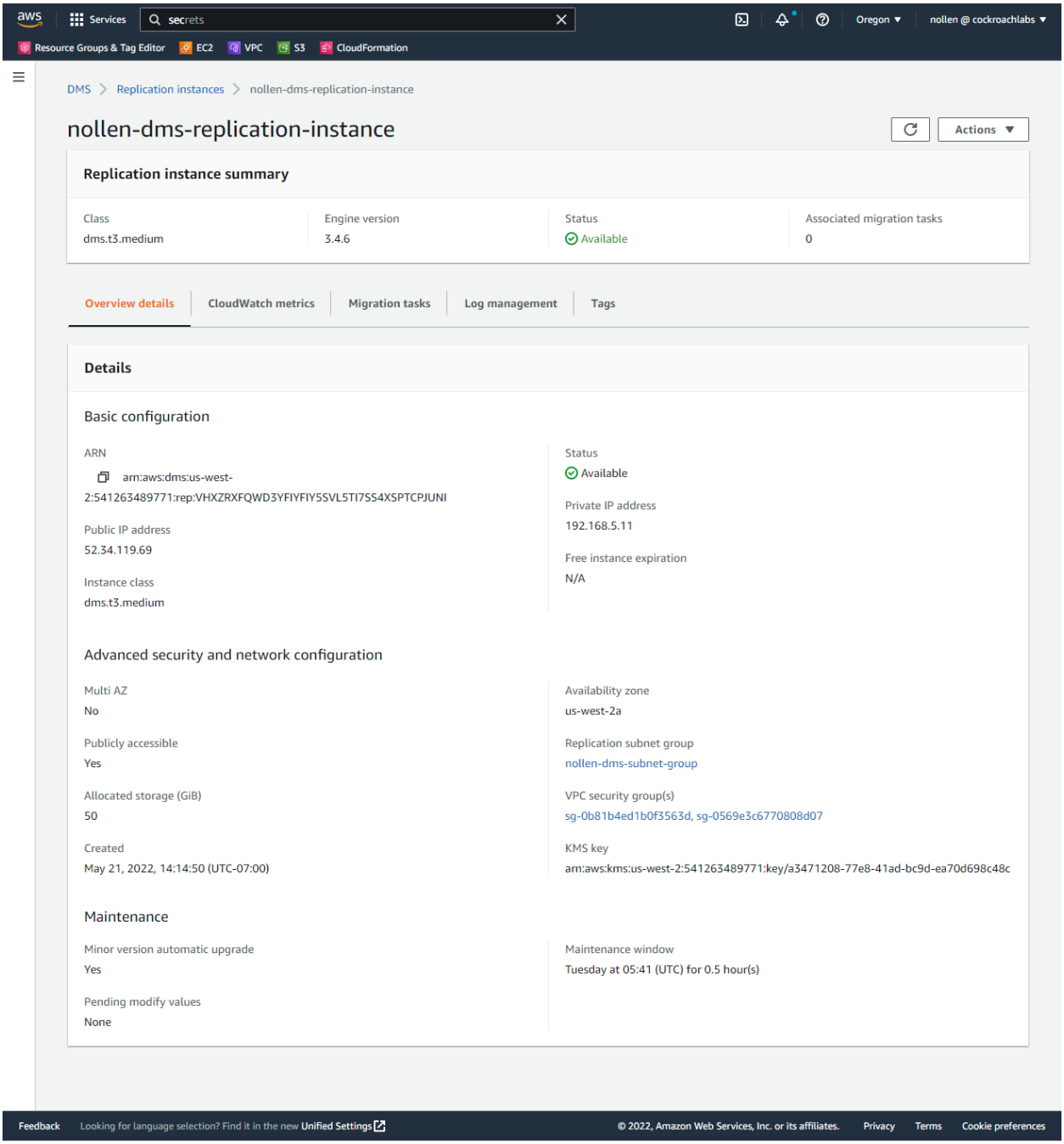


Create the Replication instance

DMS uses an EC2 instance to perform the replication between databases. In this step we create the replication instance.

Be sure to place the replication instance in the same VPC we've place both of our databases. Note that it is possible to create a Multi-AZ replication instance configuration (for failovers), but for this demonstration, I'm going to leave that off. The subnet group that I created above included the public subnets so that I can access the instance if I need.

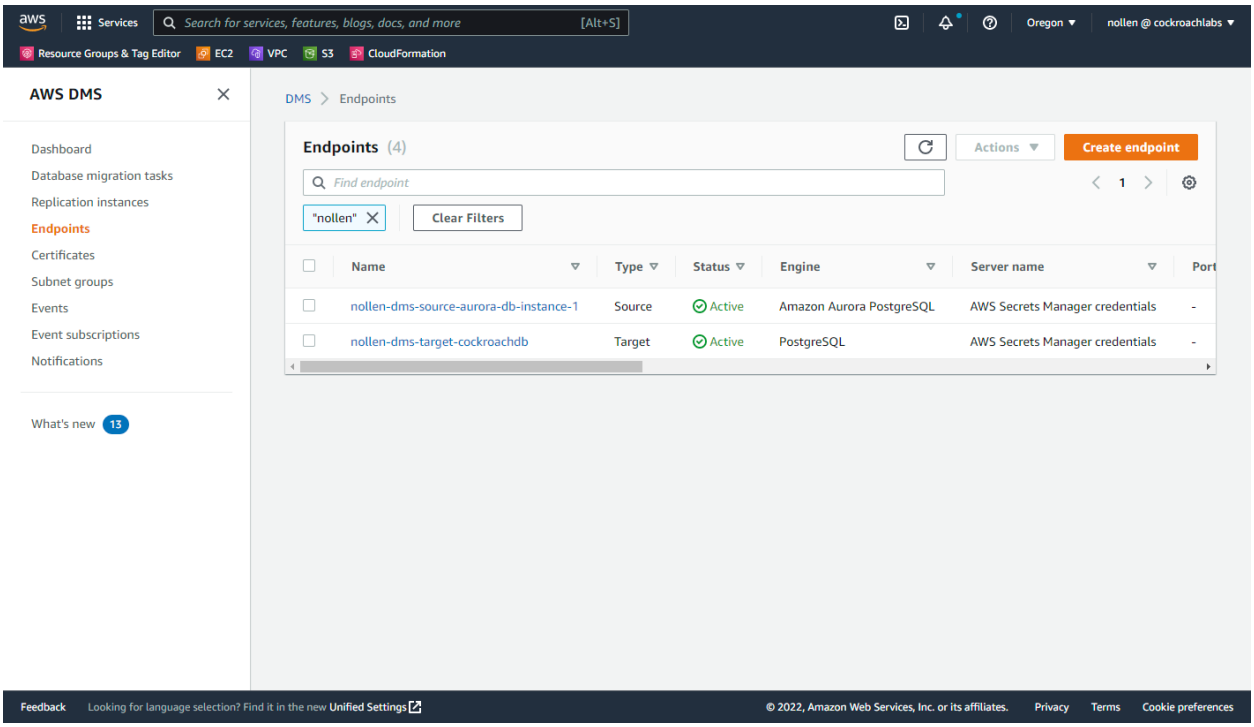
I also chose a small instance class for this demonstration (dms.t23.medium). Do not do this in any environment where you need substantial throughput (production or load testing). Choosing an instance class that is too small, along with storage is the Achilles heel of DMS.



Create the Endpoints

Now, we need to create the endpoints (the target and source) for DMS. To get started, navigate to the AWS DMS page and then select “Endpoints” from the left hand menu.

When we’re done, we should have the following:



Create the Source Endpoint

Our source endpoint is going to be our Aurora PostgreSQL database that we created above. I'll be using the ARN of the secret we created above, along with the IAM role (and it's included policy) in this set up.

aws | Services | secrets | Oregon | nollen @ cockroachlabs

Resource Groups & Tag Editor | EC2 | VPC | S3 | CloudFormation

AWS DMS

- Dashboard
- Database migration tasks
- Replication instances
- Endpoints
- Certificates
- Subnet groups
- Events
- Event subscriptions
- Notifications

What's new 13

DMS > Endpoints > Create endpoint

Create endpoint

Endpoint type Info

Source endpoint
A source endpoint allows AWS DMS to read data from a database (on-premises or in the cloud), or from other data source such as Amazon S3.

Target endpoint
A target endpoint allows AWS DMS to write data to a database, or to other data source.

☑ Select RDS DB instance

RDS Instance
Instances available only for current user and region
nollen-dms-source-aurora-db-instance-1

Endpoint configuration

Endpoint identifier Info
A label for the endpoint to help you identify it.
nollen-dms-source-aurora-db-instance-1

Descriptive Amazon Resource Name (ARN) - *optional*
A friendly name to override the default DMS ARN. You cannot modify it after creation.
Friendly-ARN-name

Source engine
The type of database engine this endpoint is connected to.
Amazon Aurora PostgreSQL

Access to endpoint database
- (•) AWS Secrets Manager
- ( ) Provide access information manually

Secret ID
Amazon Resource Name (ARN) of the secret used to manage access to your endpoint database. If you need a new secret ARN, create it from AWS Secrets Manager.
arn:aws:secretsmanager:us-west-2:541263489771:secret:/nollen/dms/aurora-WaQe5W

IAM role
IAM role that grants Amazon DMS permissions to access the specified secret (and any required KMS encryption key). For a new IAM role, create it from IAM.
arn:aws:iam::541263489771:role/nollen-read-secrets-role

Secure Socket Layer (SSL) mode
The type of Secure Socket Layer enforcement
none

Database name
defaultdb

▸ Endpoint settings

▸ KMS key

▸ Tags

▾ Test endpoint connection (optional)

VPC
vpc-0458048d6a007cb76 - nollen-dms-vpc-us-west-2

Replication instance
A replication instance performs the database migration
nollen-dms-replication-instance

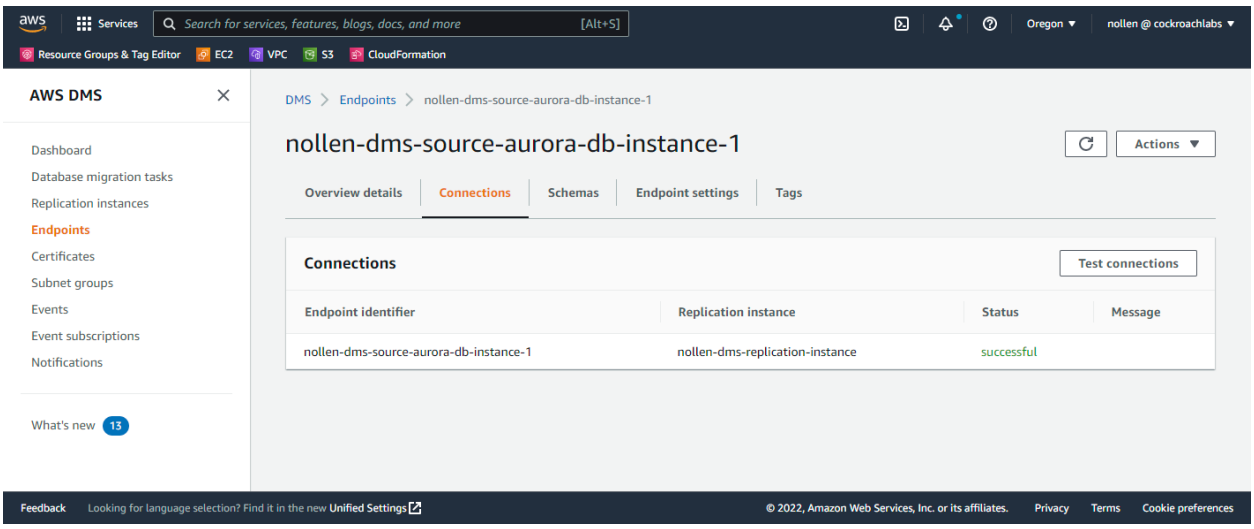
⚠ **Your endpoint will always be created even if the connection fails**
After clicking 'Run test', DMS creates the endpoint with the details you provided and attempts to connect to it. If the connection fails, you can edit the endpoint definition and test the connection again. You can also delete the endpoint manually.

Run test

Endpoint identifier	Replication instance	Status	Message
nollen-dms-source-aurora-db-instance-1	nollen-dms-replication-instance	successful	

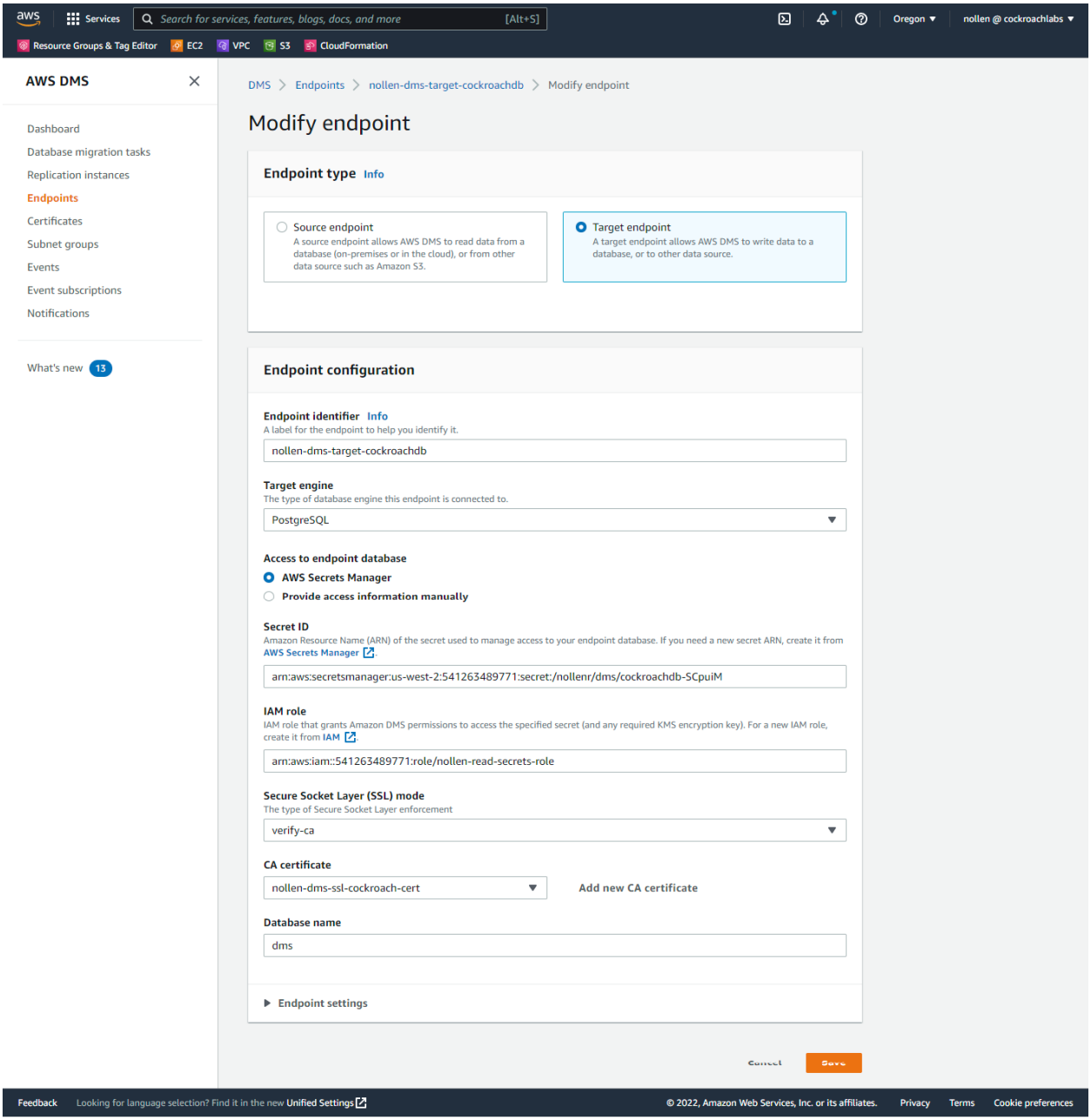
Cancel | Create endpoint

Once you've created the endpoint, be sure to test connectivity. Do not proceed until connectivity is successful.



Create the Target Endpoint

Creating the CockroachDB Target endpoint will include using the certificate we uploaded above. We'll chose "PostgreSQL as the Target Engine.



Be sure to test the connectivity to the CockroachDB instance, and fixing any problems, before going on to the next step.

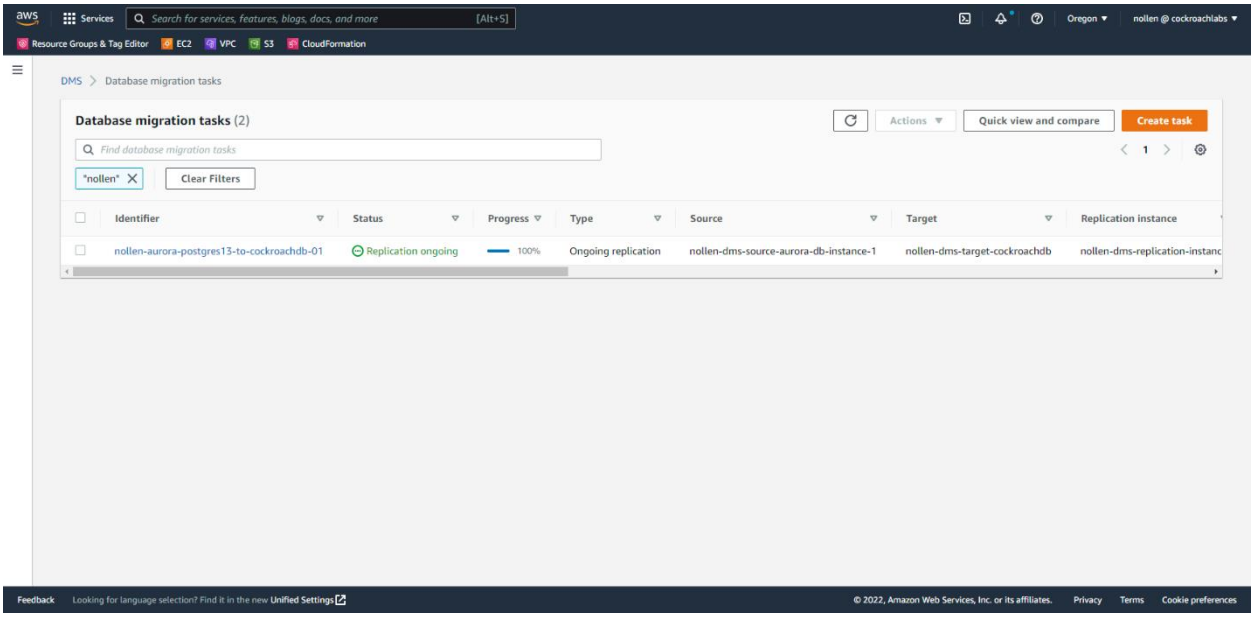
Create the Database migration task

At this point we've got:

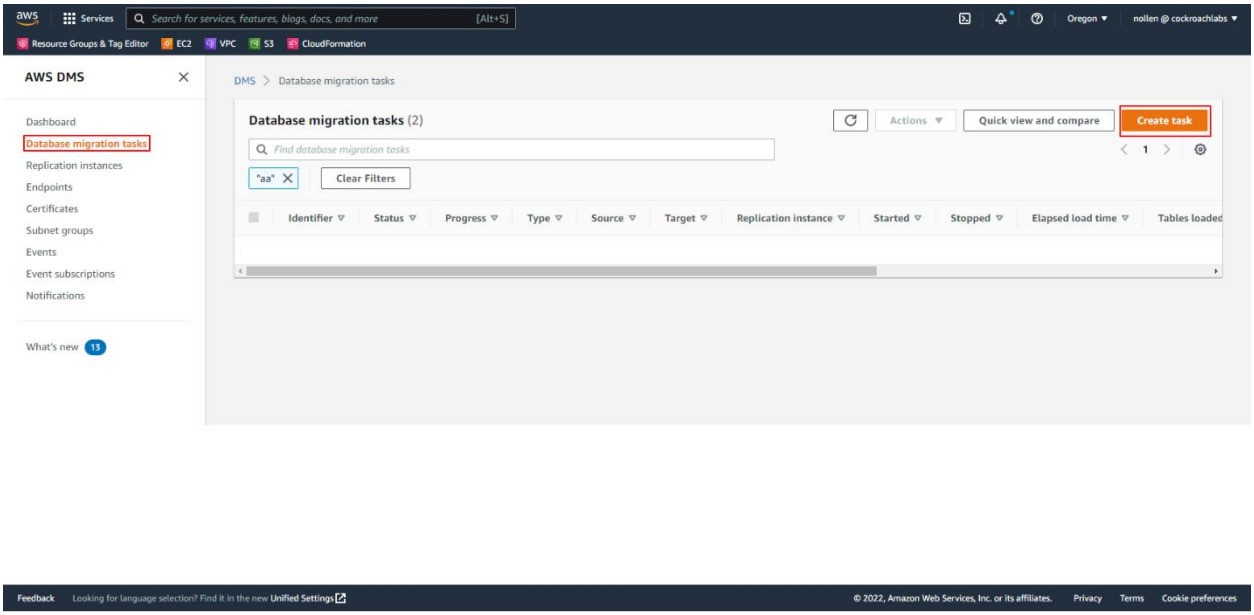
- A Source Aurora PostgreSQL Database configured with replication
- A 3 Node CockroachDB Cluster running in the same VPC as the Aurora Database
- DMS Endpoint for the Source Aurora Database
- DMS Endpoint for the Target Cockroach Database
- The source table created in the Aurora Database

- The target table create in the Cockroach Database (although this was not necessary)
- The DMS Replication instance is running

Now, we're ready to create the DMS task, which is the instruction set for the replication from the source to the target. When we've finished and the task is running it will look like this:



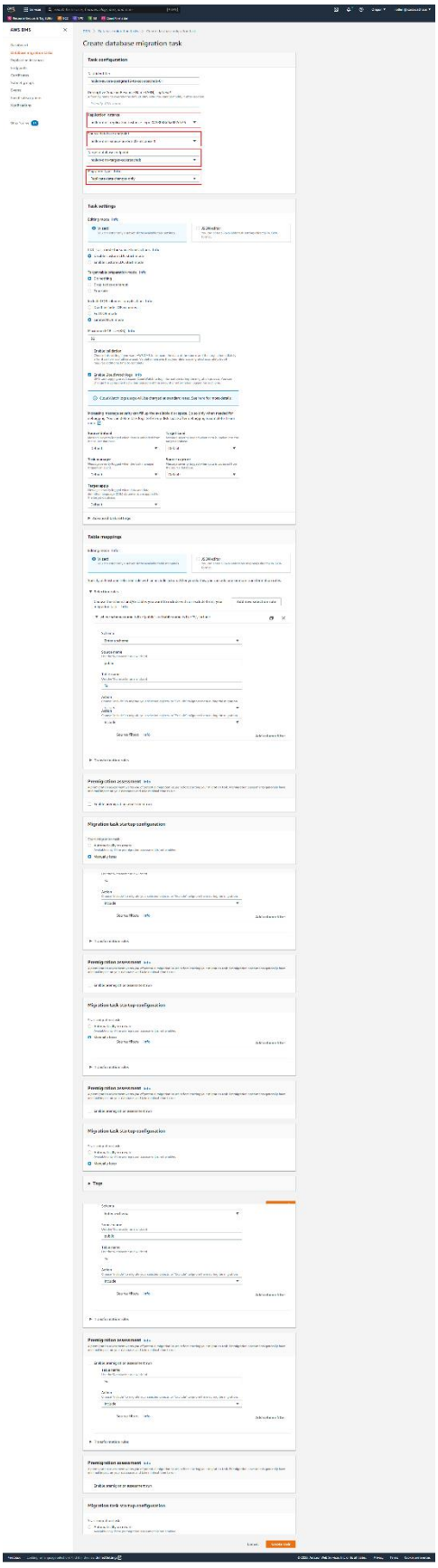
To create the task navigate to "AWS DMS", select "Database migration tasks" from the left hand side and then choose "Create task".



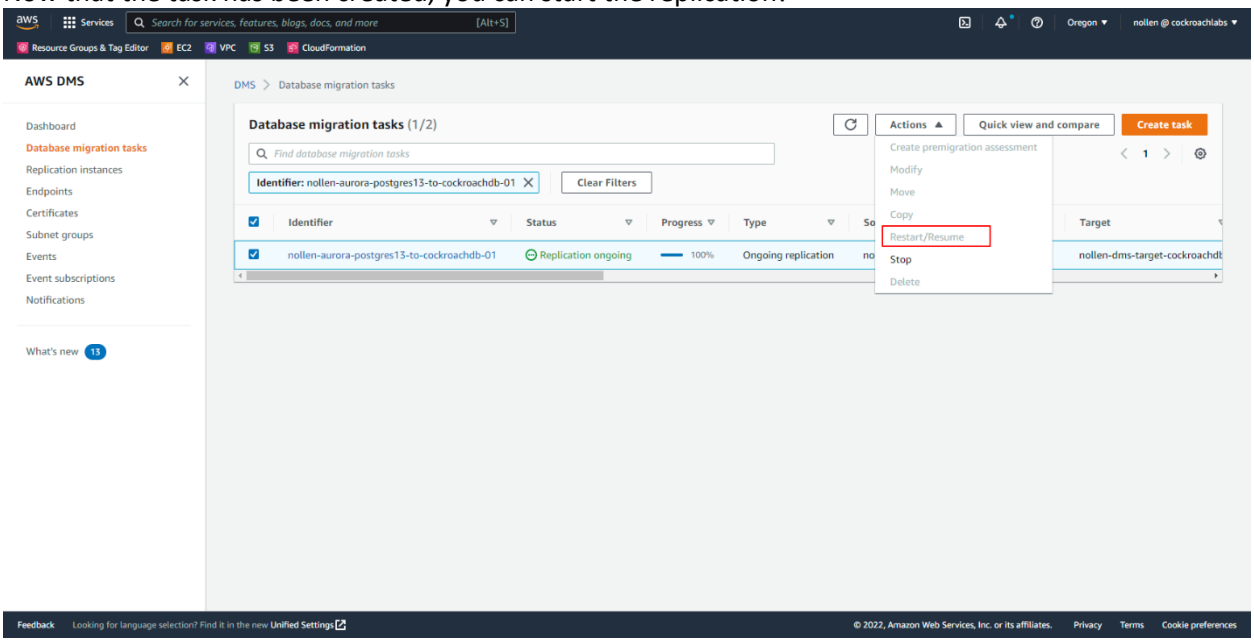
Basically, we've done all the hard work. In this step we're going to choose elements we've created in the steps above.

In my example below, I'm including CloudWatch logs (which we should modify after starting the replication so that logs are only kept for 5 or 7 days).

You also need to create at least one selection rule which tells DMS which objects to include in the replication task. In my example, I'm selecting all the tables in the public schema of the "defaultdb".

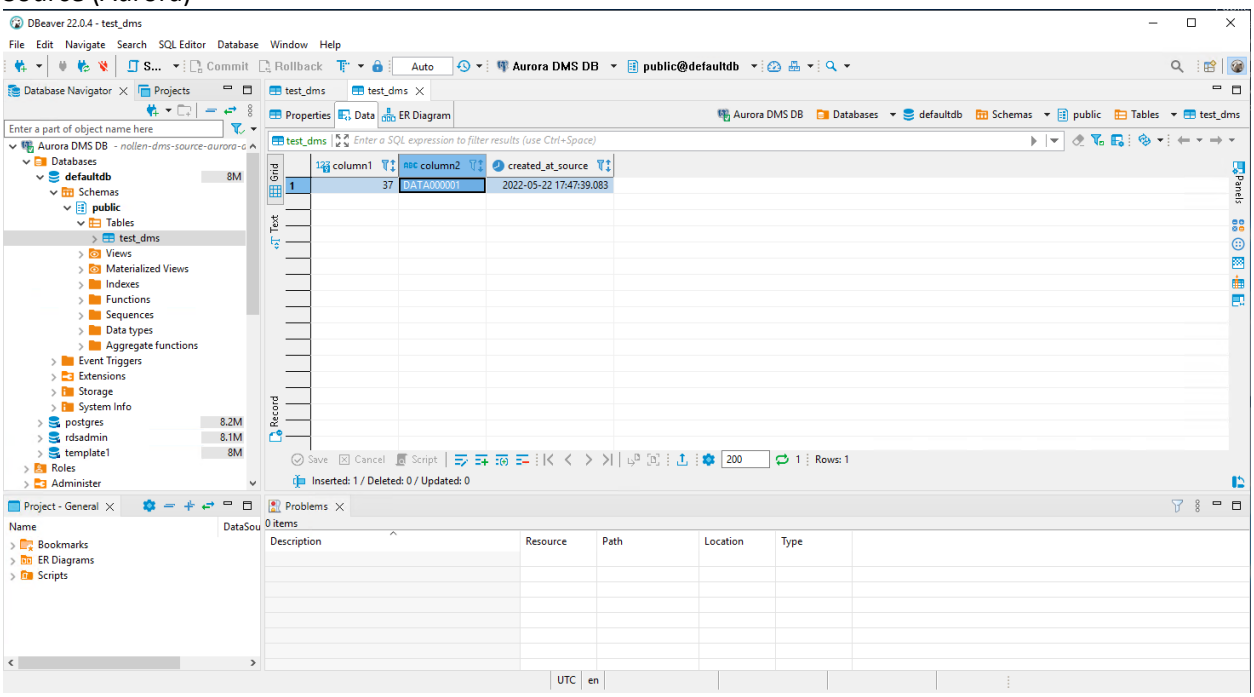

AWS DMS
Create database migration task
Task configuration
Task settings
Table mappings
Migration task startup configuration
Tags

Now that the task has been created, you can start the replication!

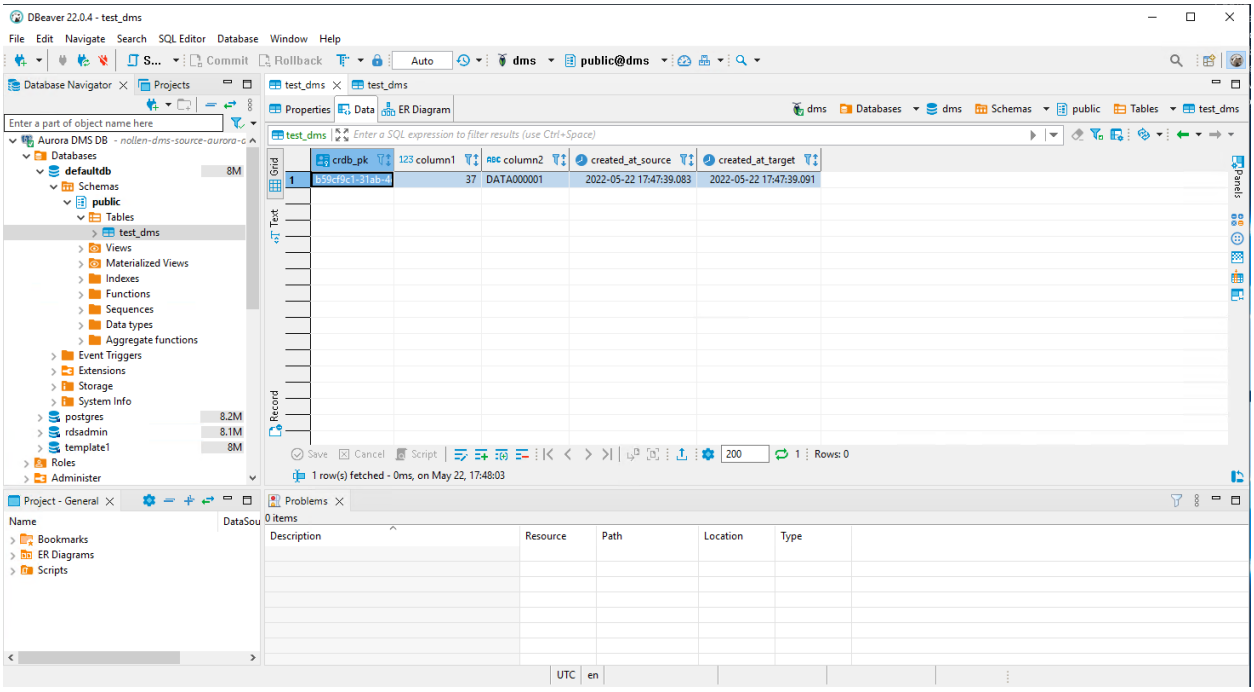


Once the task is running, you will be able to go to DBeaver and insert a row into the Aurora PostgreSQL database and see it replicated to CockroachDB

Source (Aurora)



Target (CockroachDB)



In our simple example, we can see it took about 8ms for the record to be replicated from Aurora to CockroachDB.

This was a very long DMS example and included things like Secrets, certification, subnet groups, etc. We could have made this much simpler, but leave it to you to carve out things that are not necessary for your implementation.

CHEERS!