**😊 Amazon Aurora**

Amazon Aurora is a relational database service offered by Amazon Web Services (AWS). It is compatible with both MySQL and PostgreSQL, providing high performance, availability, and scalability. Aurora is designed to deliver up to five times the performance of standard MySQL databases and up to three times the performance of standard PostgreSQL databases, making it a popular choice for applications with demanding performance requirements.

One of the key features of Aurora is its ability to automatically replicate data across multiple Availability Zones within a region, providing high availability and fault tolerance. It also offers continuous backup to Amazon S3 and automatic failover in the event of a hardware failure or other issue.

Aurora is a fully managed service, which means that AWS handles routine administrative tasks such as database setup, patching, and backups, allowing developers to focus on building their applications rather than managing infrastructure.

**😄 Use cases of Amazon Aurora:**

Amazon Aurora is a versatile database service that can be used in various use cases across different industries. Here are some common scenarios where Aurora shines:

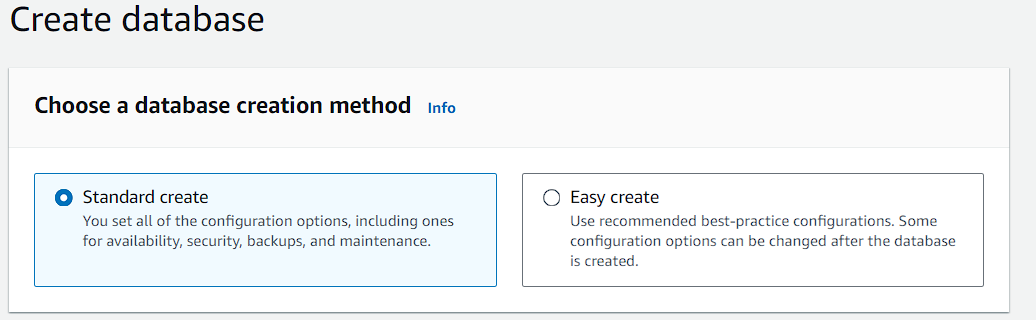
Highly Scalable Web Applications: Aurora's ability to automatically scale storage and compute resources makes it ideal for web applications experiencing unpredictable or rapid growth in user traffic. Whether it's an e-commerce platform, a content management system, or a social media platform, Aurora can handle the workload efficiently.

1. **Real-Time Analytics:** Organizations dealing with large volumes of data and requiring real-time analytics can benefit from Aurora's performance and scalability. By leveraging Aurora's compatibility with popular analytics tools and its ability to handle complex queries efficiently, businesses can gain valuable insights from their data without compromising on performance.
2. **Mission-Critical Enterprise Applications:** For businesses that rely on highly available and durable databases for critical operations, Aurora's built-in replication and failover capabilities offer peace of mind. Whether it's managing inventory, processing financial transactions, or handling customer data, Aurora ensures data integrity and availability.
3. **Software as a Service (SaaS) Platforms:** SaaS providers looking to offer scalable and reliable database services to their customers can leverage Aurora's multi-tenancy support and pay-as-you-go pricing model. Aurora's ability to isolate resources for different tenants while ensuring high performance and availability makes it a preferred choice for SaaS applications.
4. **Data Warehousing:** Aurora's compatibility with popular analytics and BI tools, along with its ability to handle large volumes of data, makes it suitable for building data warehousing solutions. Whether it's storing historical data for analysis or consolidating data from multiple sources, Aurora can provide the performance and scalability needed for data warehousing workloads.
5. **IoT and Time-Series Data:** Applications dealing with IoT (Internet of Things) devices and time-series data often require a database that can handle high write throughput and efficiently query data over time. Aurora's performance optimizations and storage scalability make it well-suited for capturing, storing, and analyzing time-series data generated by IoT devices.
6. **Content Management Systems (CMS):** CMS platforms that require fast read and write access to content, such as blogs, news websites, or digital publishing platforms, can benefit from Aurora's high performance and scalability. Aurora's compatibility with popular CMS frameworks and its ability to handle concurrent user interactions make it a reliable choice for content-driven applications.

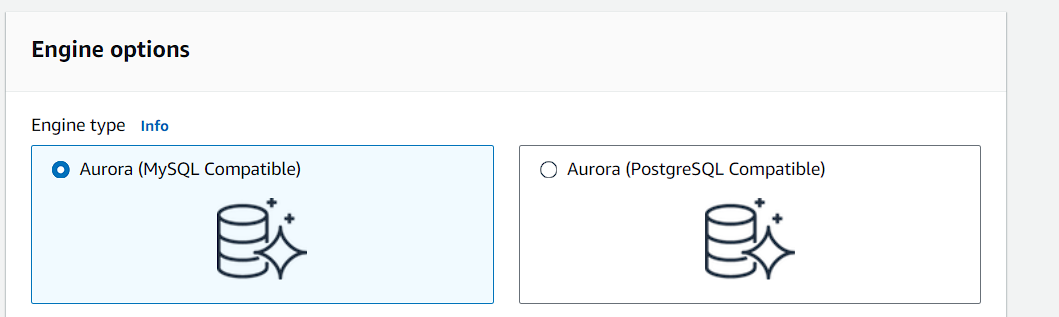
**In this guide, we're setting up Amazon Aurora database instances, both standard and serverless, using the AWS Console. The end goal is to demonstrate how to create, configure, and manage Aurora instances for various use cases, such as highly scalable web applications, real-time analytics, and mission-critical enterprise applications. Additionally, we explore failover capabilities to ensure high availability and fault tolerance. Through step-by-step instructions and examples, users learn to leverage Aurora's performance, availability, and scalability features effectively.**

**😄 To begin with the Lab:**

1. Log in to AWS Console and navigate to RDS.
2. There you are going to create a database for Amazon Aurora.
3. Choose a standard for creation method.



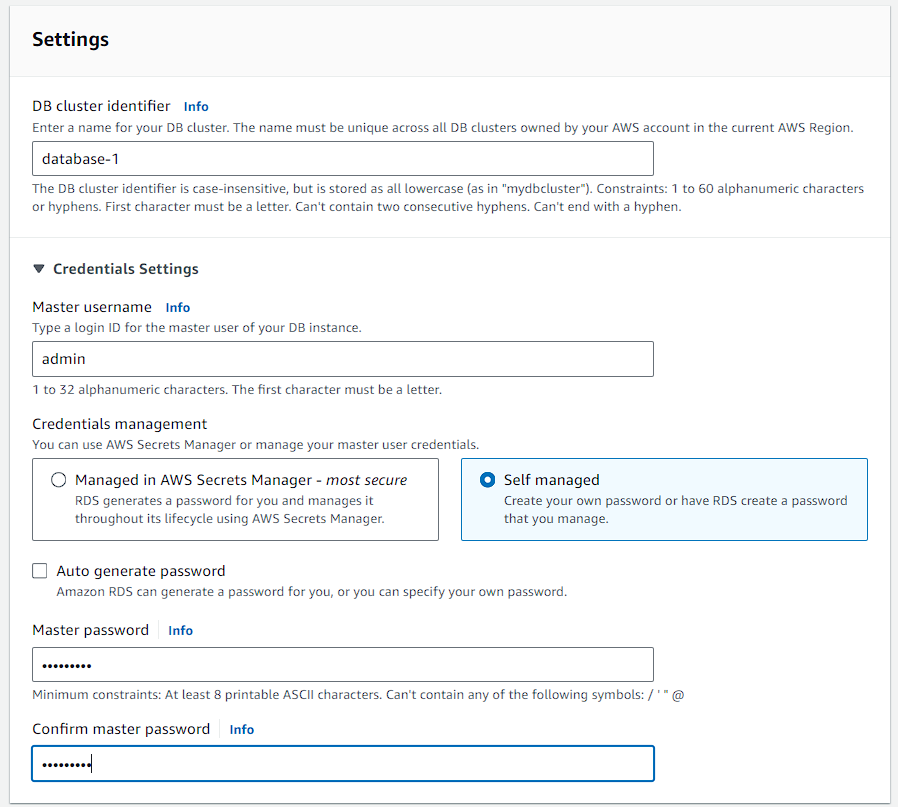
1. Then in the engine options choose Amazon Aurora and select the latest engine version which is available.



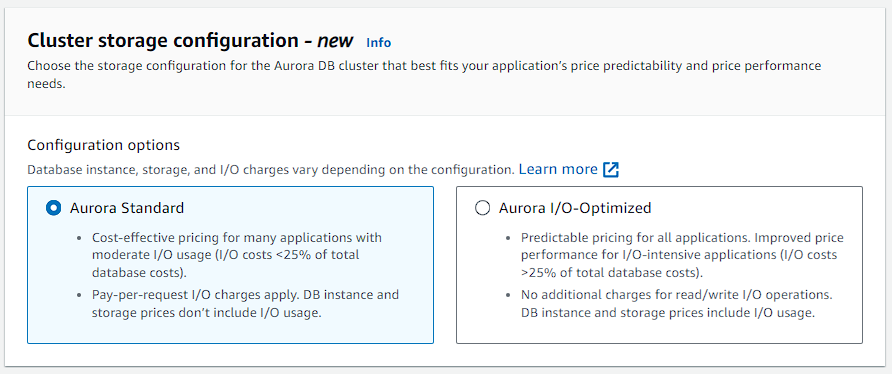
1. Choose Dev/Test for your template.



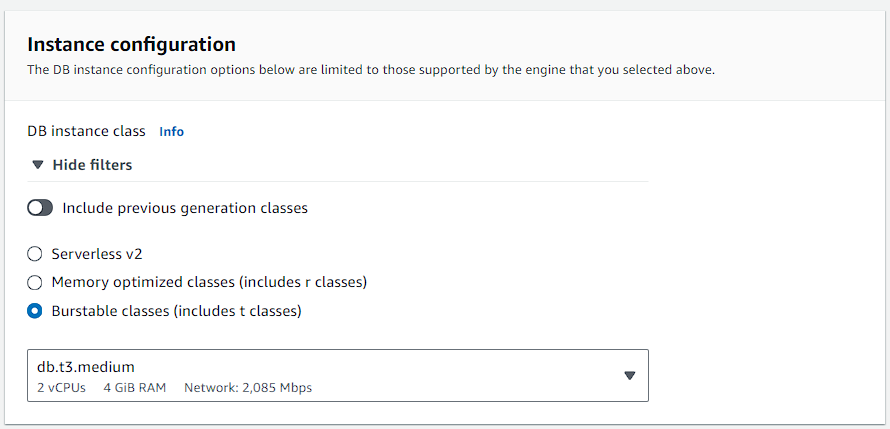
1. Then in the settings you can give your cluster identifier a name or you can leave it to default.
2. Then give it a master username and then select self-managed password and give a password of your choice.



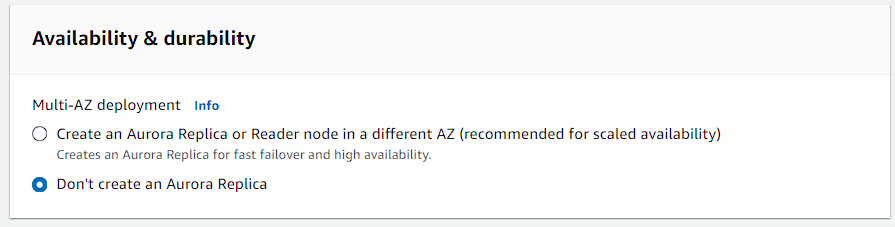
1. For cluster storage choose the default Aurora standard.



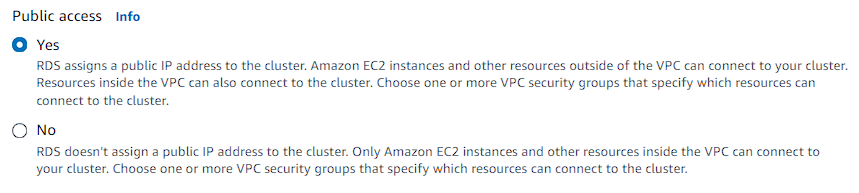
1. In the instance configuration choose burstable classes and choose the smallest instance type possible.



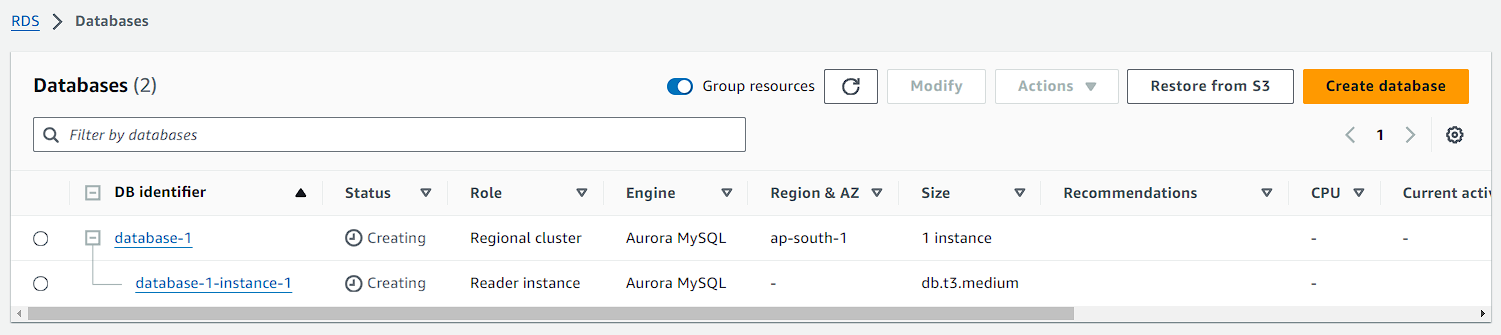
1. Keep it to default in availability and durability as we don’t need any replica for now.



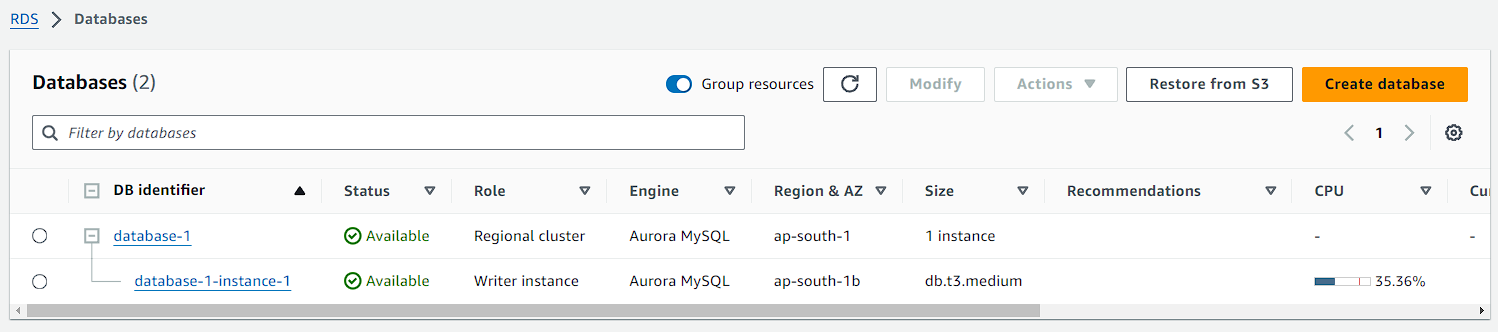
1. Now in the connectivity keep everything to default just make your RDS publicly accessible so that you can connect to it.



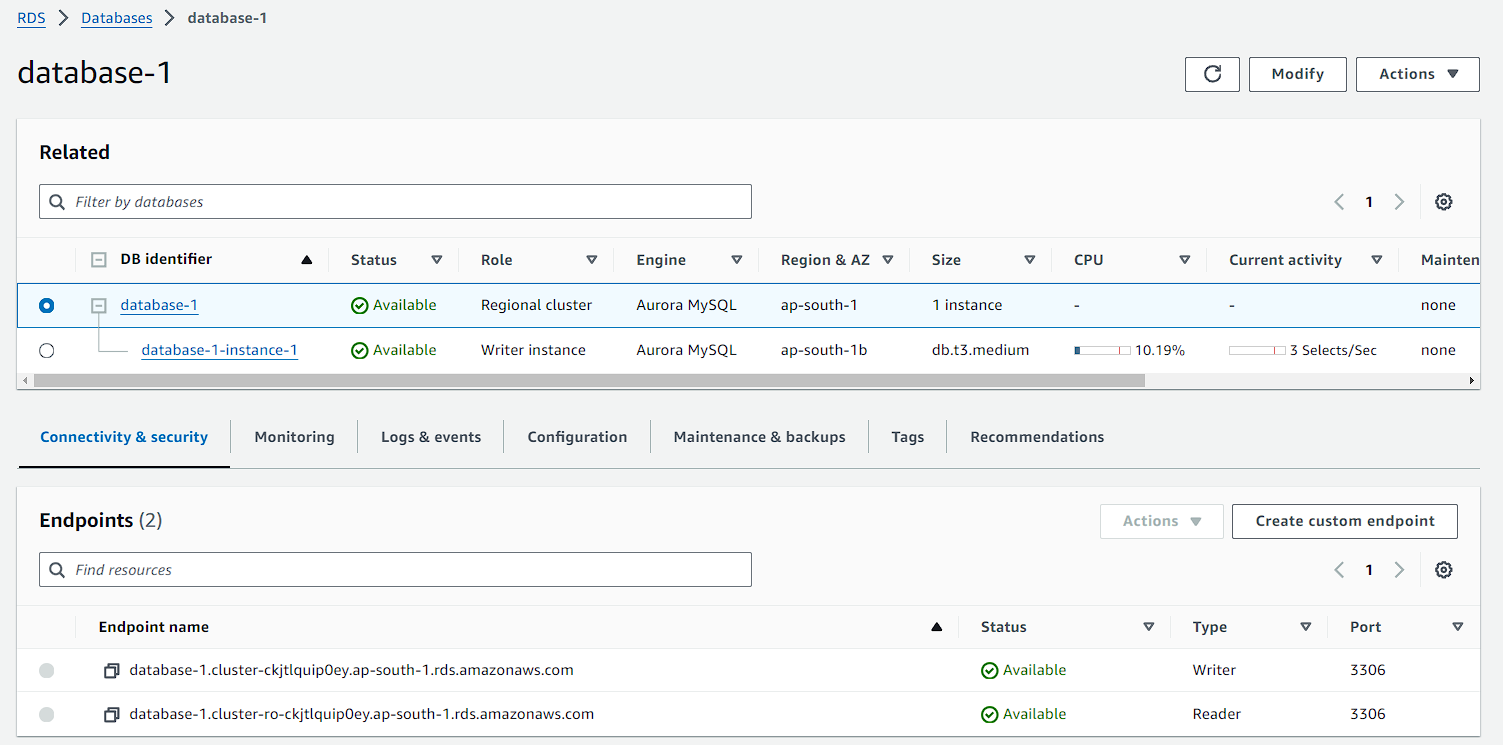
1. With that you can also disable enhanced monitoring and then create your instance.
2. Now you have to wait for some time because it will take several minutes to create your database.



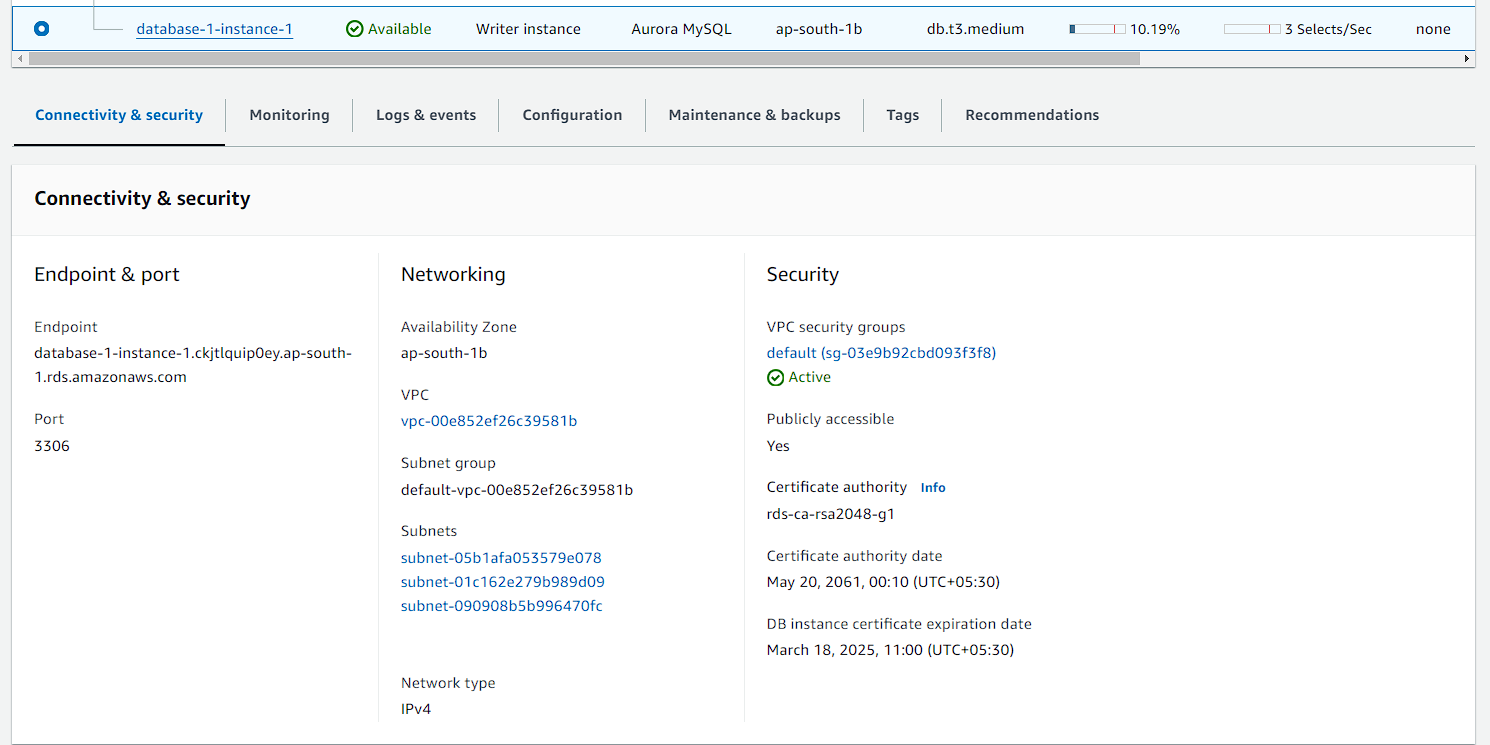
1. Once your RDS is available open it.



1. In your database you can see the endpoints for reader and writer.

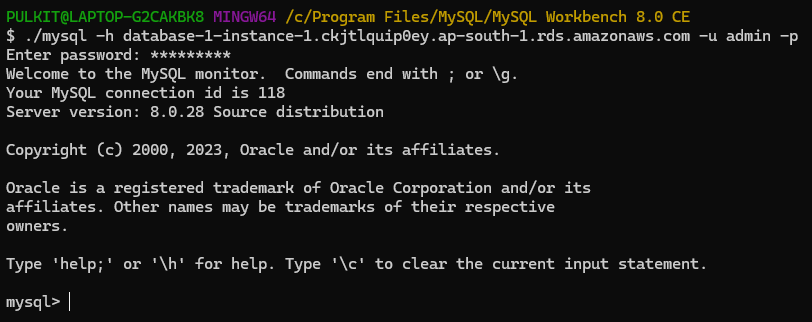


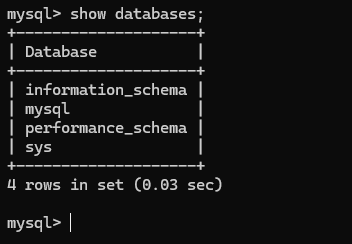
1. And in your database instance you can see the rest of the things.
2. Now you are going to copy this endpoint and use this to connect your database.



1. Here I have used GitBash to connect with Aurora MySQL database and run this command.
2. But first you should have MySQL workbench installed on your local machine then you have to move to the folder where your Workbench is installed then open GitBash from that folder.

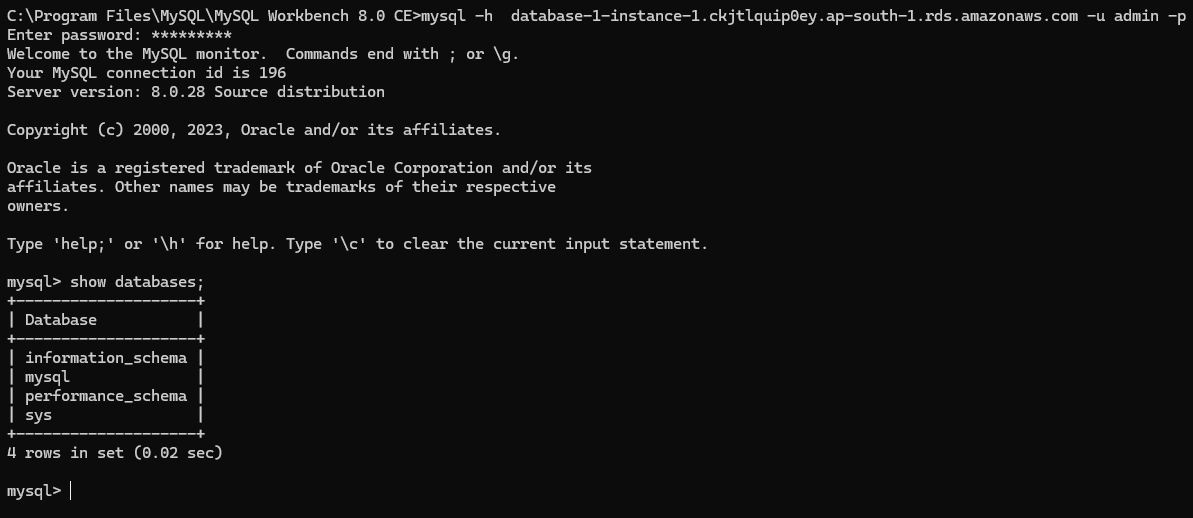
**./mysql -h paste your writer instance endpoint -u admin -p**



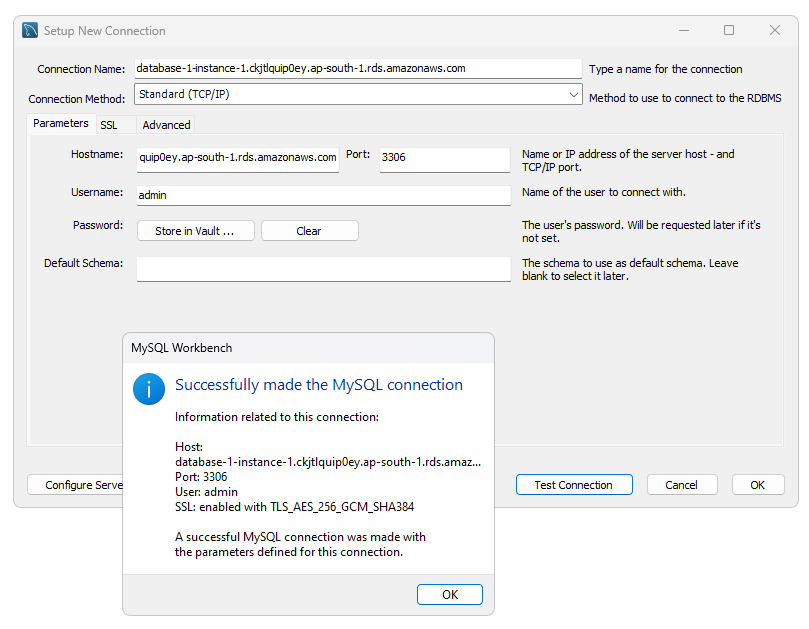


1. The same thing can be done on the Command Prompt too.

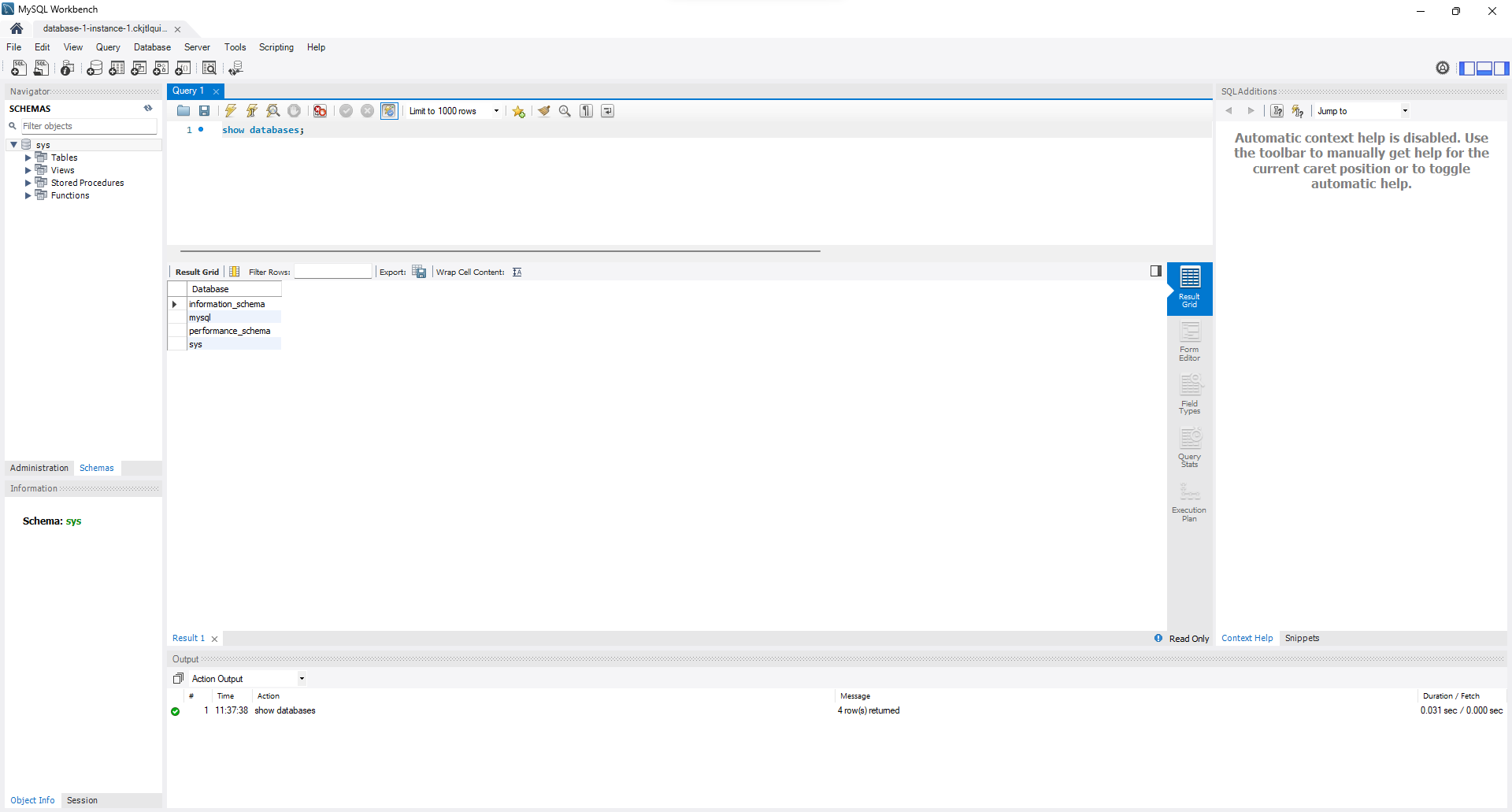
**mysql -h paste your writer instance endpoint -u admin -p**



1. Now if you want to use MySQL Workbench, you can do that too. Just copy the endpoint in the Hostname then describe your username and password and click on test connection.
2. After that it will state that successfully made the MySQL connection then click on your connection to open it.

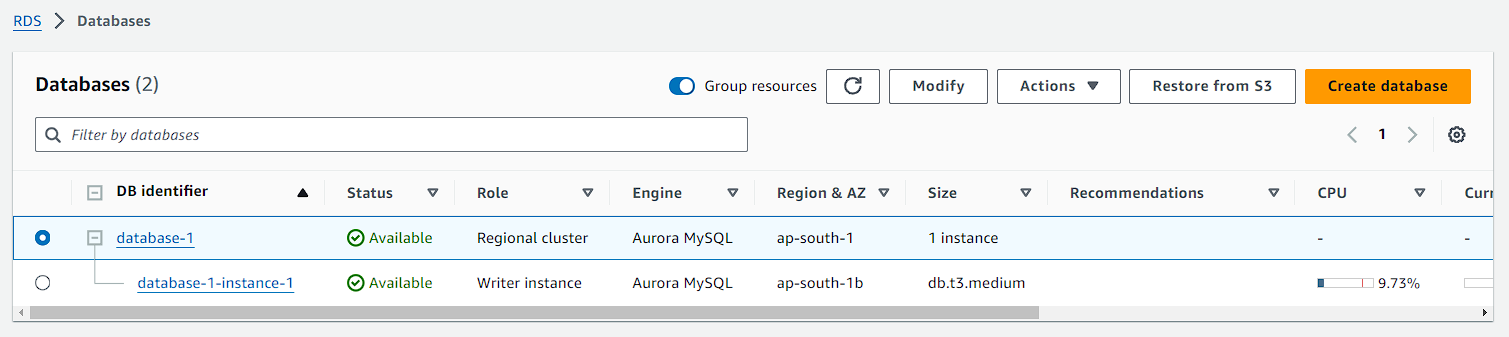


1. Here if you run the same command to show the databases you can see the same thing. And here you can create new databases to easily.

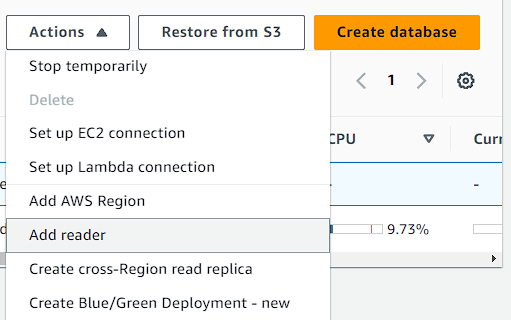


**😄 Creating Serverless Aurora instance:**

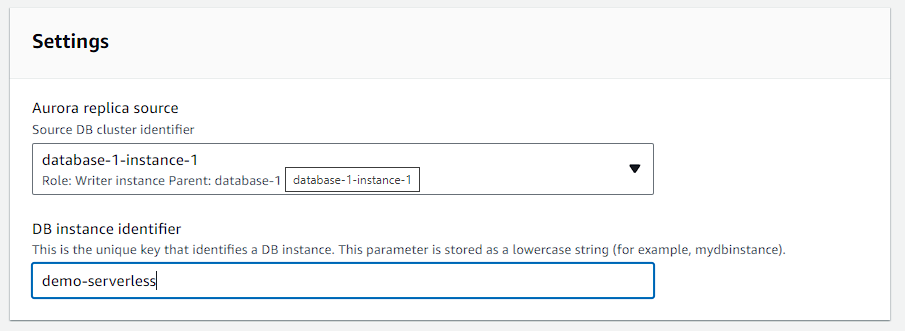
1. Now AWS provides us an option to add a serverless instance.
2. Here you can see our database instance, now you have to click on actions



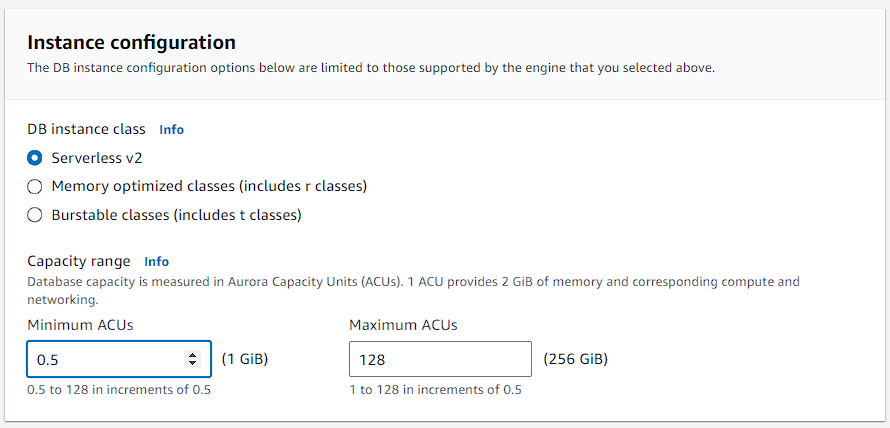
1. From actions click on Add Reader.



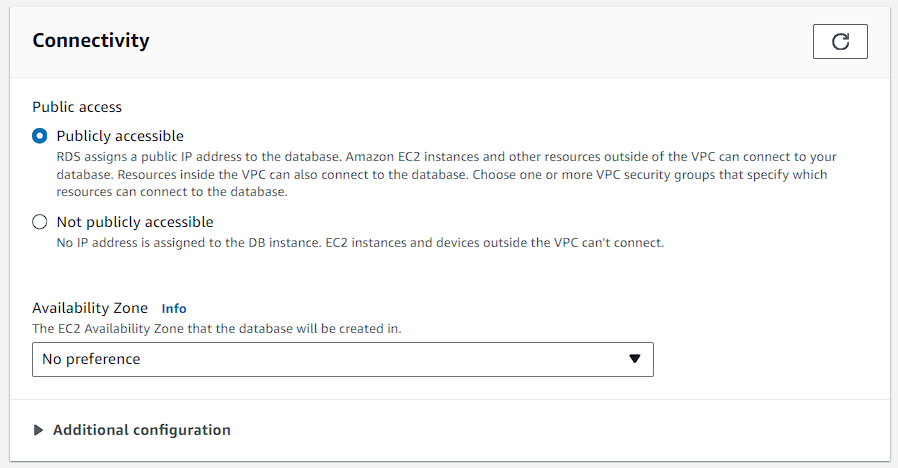
1. First you are going to provide it with a name.



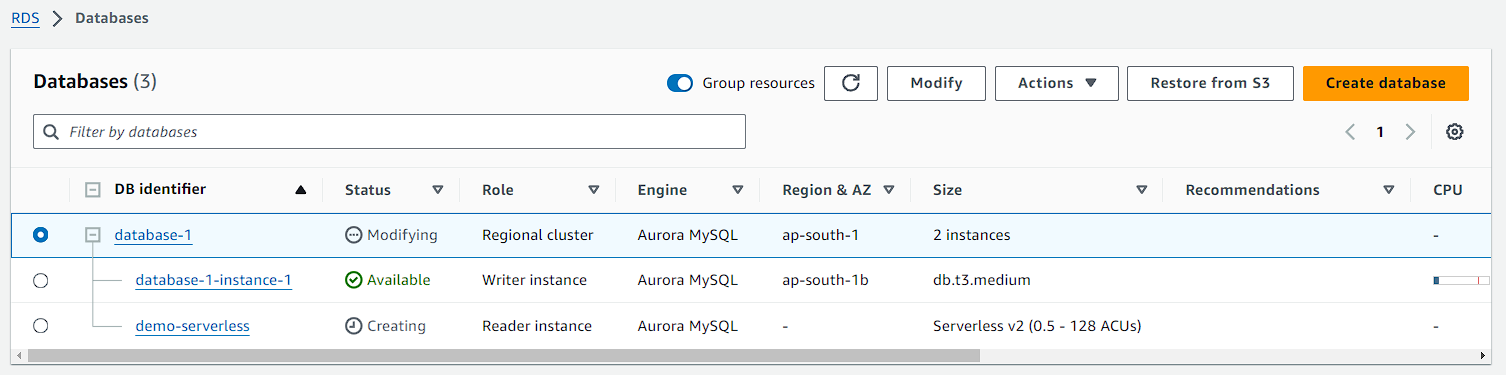
1. This time for the instance configuration you have to choose Serverless V2. Then go with the default capacity range.



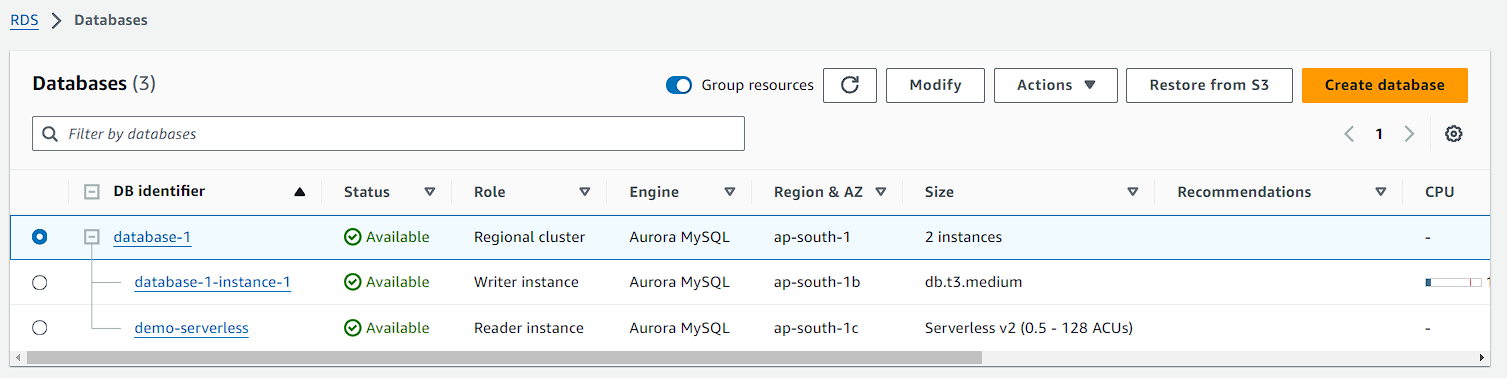
1. In the connectivity make it publicly accessible. After that keep everything to default and click on add reader.



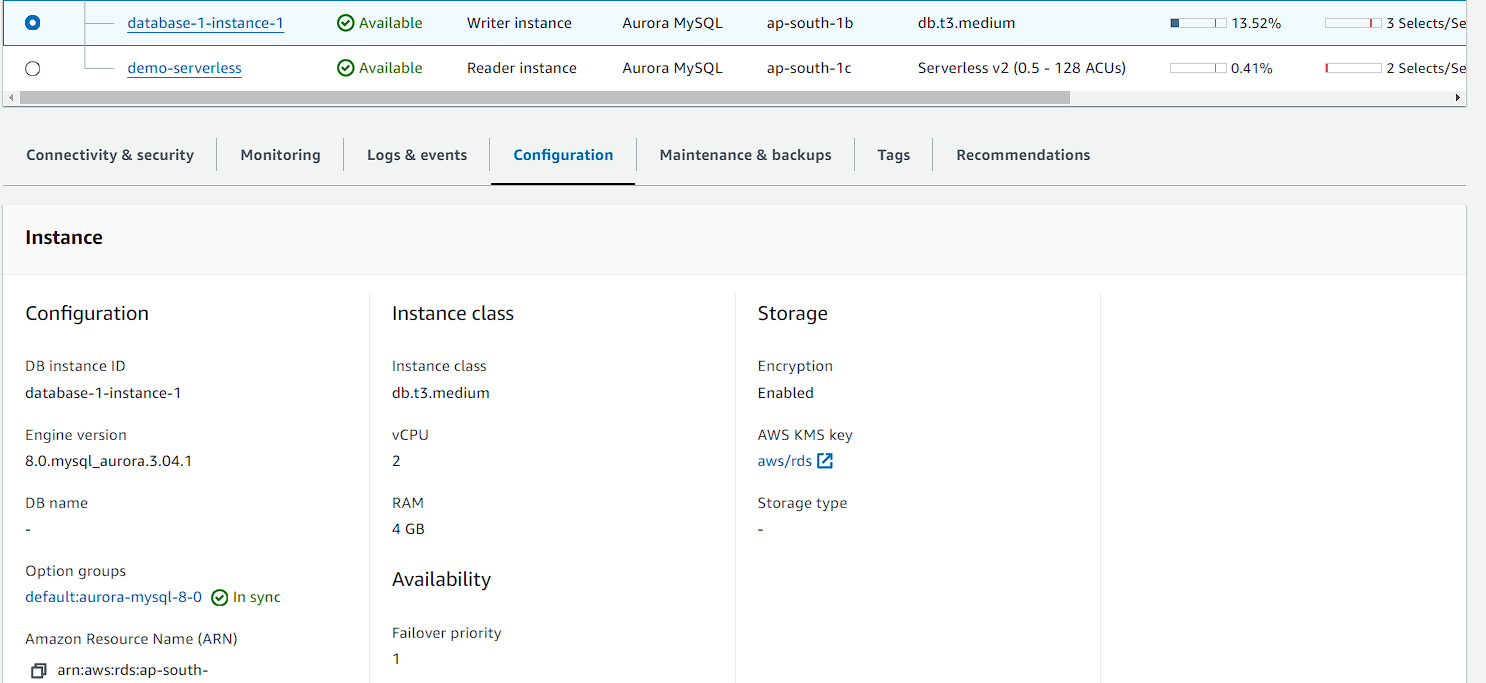
1. Here you can see that your database is modifying itself. It will take time.

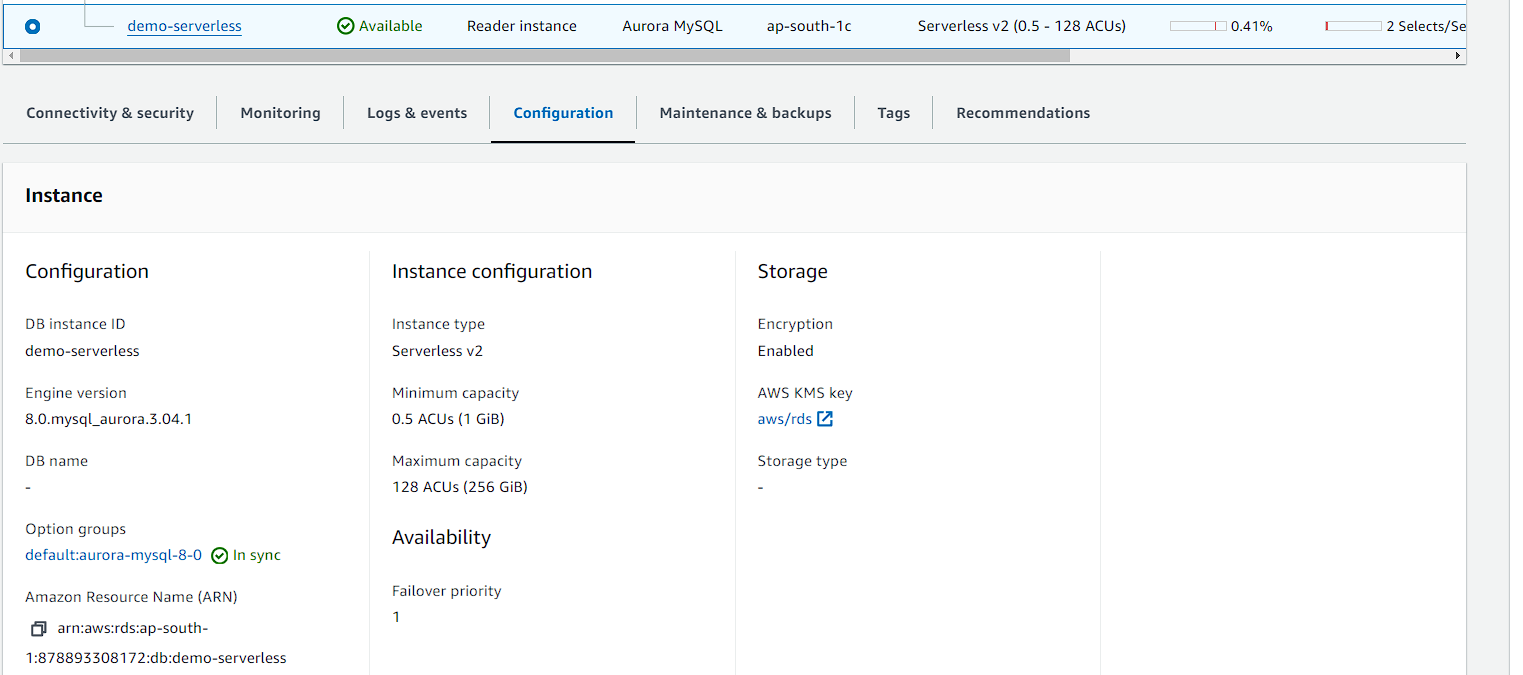


1. After some time, you will see that your serverless instance is available and it is a reader instance type.

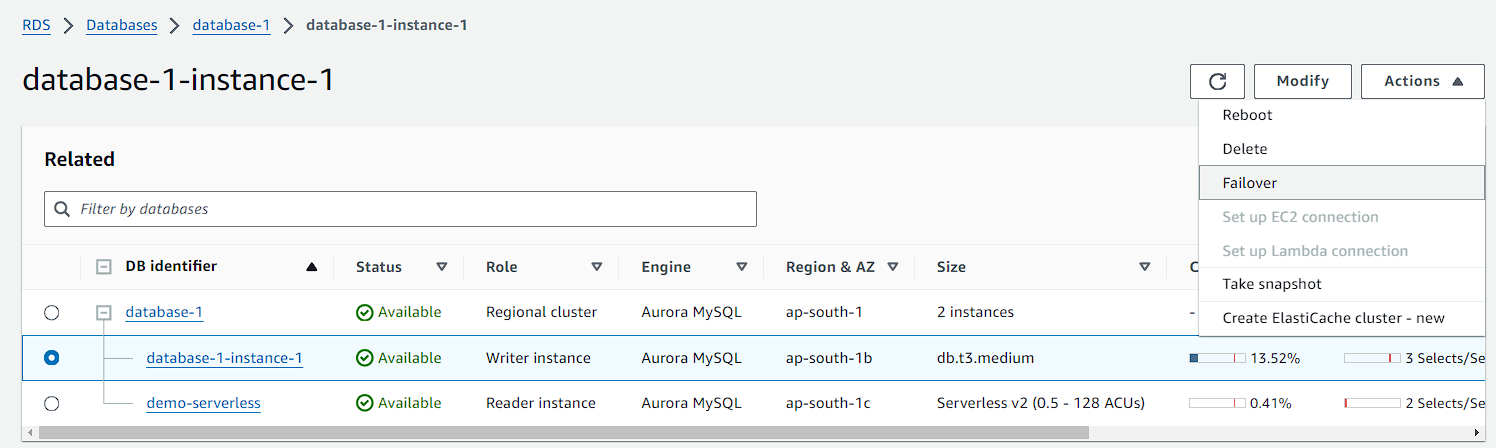


1. Now if you look at the configuration of both your instances reader and writer.
2. You will observe that both have the failover priority set to 1.
3. Which means that if one fails then the second will take over as writer instance.

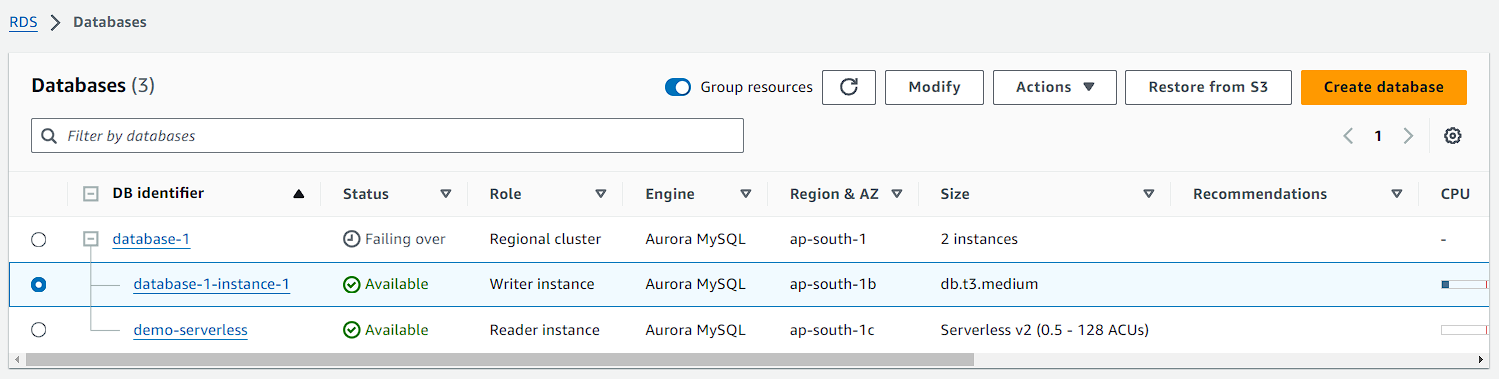




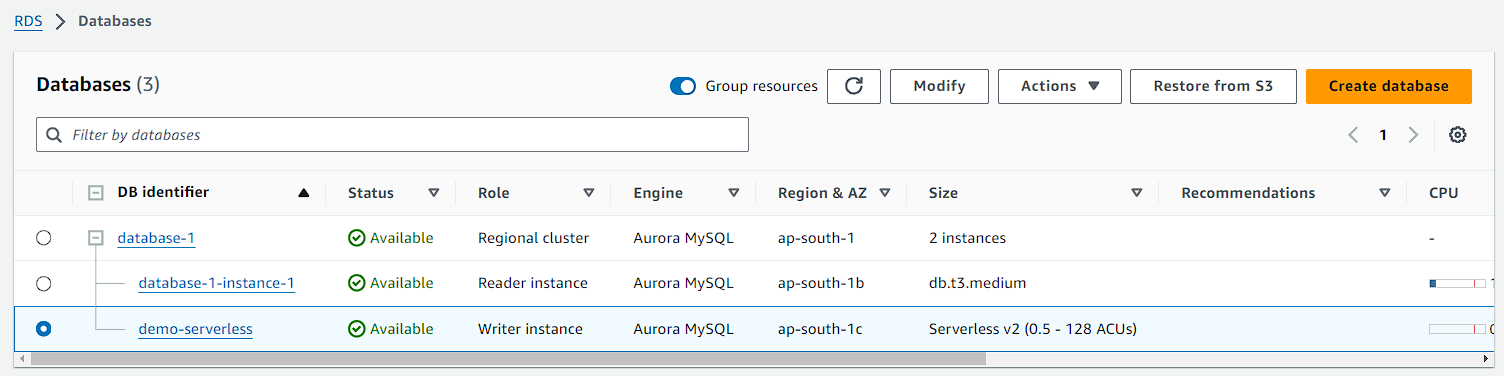
1. Now select your writer instance and then click on actions. There you have to click on failover.
2. After that you have to wait for a while. It will take some time to modify things.



1. You can also see that your database is failing over.



1. Once the failover modification is completed. You can see that the serverless instance in the writer instance now.



1. Once you are done just delete your instances one by one then delete your database.