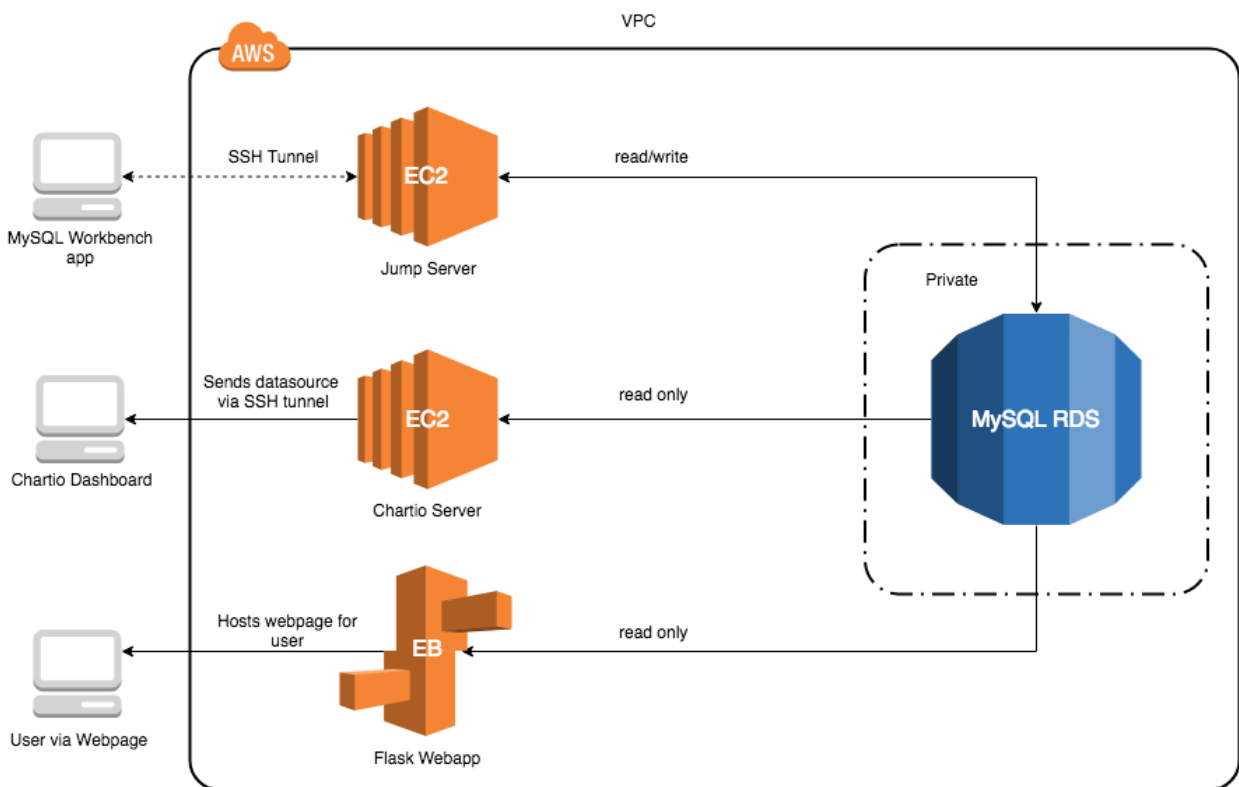


Goal:

Create a basic server configuration utilizing:

1. Private RDS instance using MySQL
2. Webapp displaying RDS data
3. Jump Server to modify RDS data and add authorized users
4. Chartio dashboard to provide visualization of RDS data

Final Architecture:



Summary:

The challenge of this assignment was understanding the intricacies of connecting a private RDS to various services, both inside and outside of AWS. The services include a webapp, a jump server for manipulating data inside the RDS, and a visualization tool.

With a public RDS, all that is needed to access the RDS is to create whitelisted security groups. However, when an RDS is private, only items in the VPC can access the RDS, this makes using third party applications difficult.

I used various different methods of connection to get to the RDS:

1. The first EC2 instance used a desktop SQL client to SSH into the jump server and then connect to the RDS. The jump server has full read/write privileges.
2. The webapp needed to be connected to the private RDS. Because I wanted to ensure clean working code, I created a process to incrementally test and debug my code. This meant initially leaving RDS public. The steps I took were:
 - a. Ran server locally and connected to local MySQL database
 - b. Ran server locally and connected to public RDS
 - c. Ran server using Elastic Beanstalk and connected to public RDS
 - d. Ran server using Elastic Beanstalk and connected to private RDS
3. The second EC2 instance was for connecting to Chartio. Because I was using a private RDS, I had to create a reverse SSH tunnel to connect them. There was some difficulty because when I initially tried this, Chartio had some internal issues.

Some difficulties I encountered were:

1. Using the AWS Elastic Beanstalk CLI. While I did not have to do this, I felt using the EB CLI would provide good practice and faster deployment. Unfortunately, the current version of the EB CLI had dependency conflicts in the libraries it used. Trying to diagnose the problem was trickier than expected.
2. Debugging on Elastic Beanstalk. Before I deployed my code, I tested it out on my local computer. Everything ran smoothly, but I ran into issues after deployment because I did not realize that EB was not running my main function.

Webapp Technologies:

I chose to keep my webapp simple and used Flask with SQLAlchemy on Python 3.6 as the web framework because this assignment emphasized understanding how the public and private services on AWS interact with external applications.

Possible Enhancements:

I wanted to embed Chartio visualizations into the web app because I wanted to provide an interesting page for the user. While Chartio does appear to have embedding support, accessing this feature required calling their sales department, so I skipped this enhancement idea.

How to set up Server Configuration

Web Application:

1. [Install AWS Elastic Beanstalk CLI](#) to quickly access logs, deploy updates, and open browser
 1. Follow instructions from Amazon until AWS CLI does not properly install.
 2. Use these commands to downgrade the version

1. `pip install --upgrade awsebcli==3.14.5`
2. `pip install urllib3==1.22`

2. [Install MySQL](#).

3. Write code for webapp. Use local machine as a database before creating RDS instance and test everything locally. Set up for RDS will be later.

Tips about Elastic Beanstalk:

- Best practice: Name the file you wish to run application.py.
- Elastic Beanstalk will not execute main function of application.py.
- If using Flask, the Flask application object should be named application.

RDS Instance:

1. Log on to AWS account and go to RDS. Click on Create Database and choose a MySQL engine.

Select engine

Engine options

☐ Amazon Aurora

Amazon Aurora

☒ MySQL

☐ MariaDB

☐ PostgreSQL

☐ Oracle

☐ Microsoft SQL Server

MySQL

MySQL is the most popular open source database in the world. MySQL on RDS offers the rich features of the MySQL community edition with the flexibility to easily scale compute resources or storage capacity for your database.

- Supports database size up to 16 TiB.
- Instances offer up to 32 vCPUs and 244 GiB Memory.
- Supports automated backup and point-in-time recovery.
- Supports cross-region read replicas.

☐ Only enable options eligible for RDS Free Usage Tier [Info](#)

Cancel **Next**

2. Choose Dev/Test as use case.

Choose use case

Use case

Do you plan to use this database for production purposes?

Use case

- ☐ **Production - Amazon Aurora** Recommended
MySQL-compatible, enterprise-class database at 1/10th the cost of commercial databases.
- ☐ **Production - MySQL**
Use [Multi-AZ Deployment](#) and [Provisioned IOPS Storage](#) as defaults for high availability and fast, consistent performance.
- ☒ **Dev/Test - MySQL**
This instance is intended for use outside of production or under the [RDS Free Usage Tier](#).

Billing is based on [RDS pricing](#).

Cancel

Previous

Next

3. Click on free tier button to prevent unneeded costs. Then fill out settings.



Free tier

The Amazon RDS Free Tier provides a single db.t2.micro instance as well as up to 20 GiB of storage, allowing new AWS customers to gain hands-on experience with Amazon RDS. Learn more about the RDS Free Tier and the instance restrictions [here](#).



Only enable options eligible for RDS Free Usage Tier [Info](#)

Settings

DB instance identifier [Info](#)

Specify a name that is unique for all DB instances owned by your AWS account in the current region.

mydbinstance

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance". Must contain from 1 to 63 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens.

Master username [Info](#)

Specify an alphanumeric string that defines the login ID for the master user.

Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.

Master password [Info](#)

Confirm password [Info](#)

Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", "", or "@".

Cancel

Previous

Next

Do not put non-alphanumeric characters in the password, as MySQL has issues recognizing punctuation in passwords.

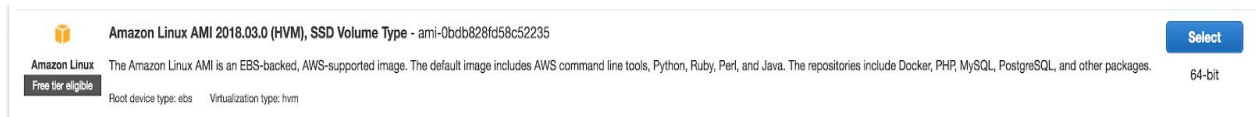
4. Choose VPC, subnet group, and security group. Default options are fine.

Do not touch public accessibility **yet**. Everything will be connected to a private RDS eventually, but initially leave it public for testing purposes.

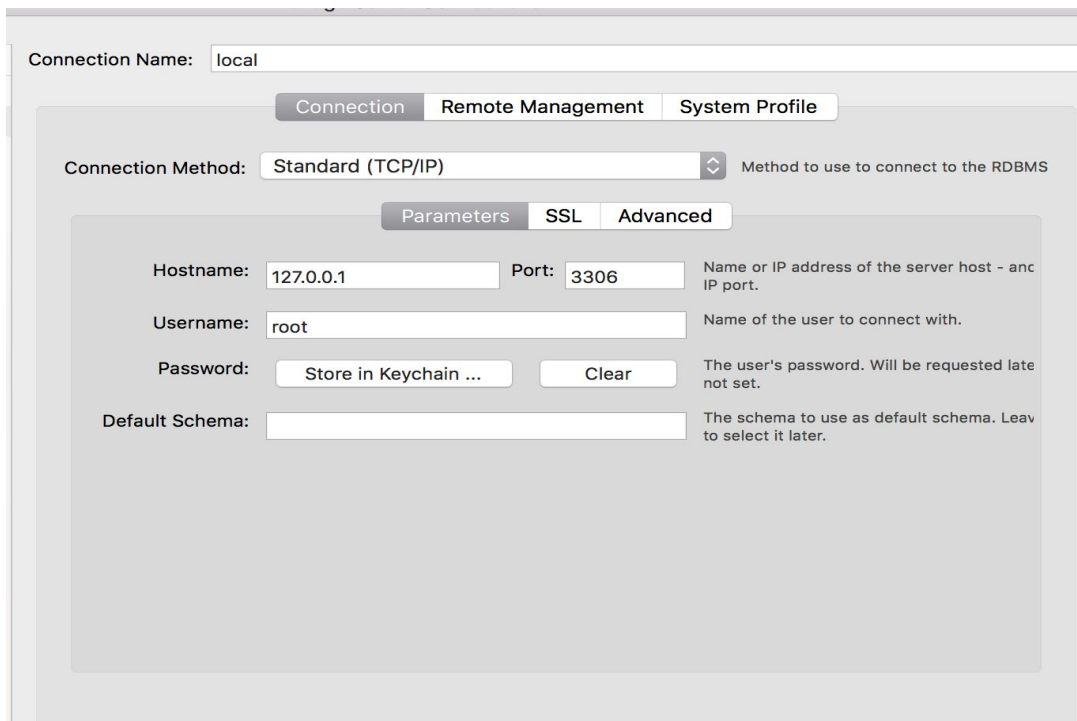
5. Configure database options. Default options are fine.
6. Scroll to the end of the page and click Create Database button.

Jump Server:

1. Create an EC2 instance for Jump Server.



2. Choose free tier option. Launch EC2 instance.
3. Go to security groups page. Click on security groups and click on the RDS security group. Click inbound and add the security group for the jump server with permissions for MySQL/Aurora. Now the jump server will be able to connect to RDS.
4. Install MySQL Workbench (or other desktop SQL client).
5. Connect local database to SQL Client. This will make importing data into the remote RDS easier.



6. Connect RDS instance to SQL Client via SSH Tunnel.

The screenshot shows the AWS CloudFormation console configuration for a connection named 'amazonrds'. The 'Connection' tab is active, showing the 'Connection Method' as 'Standard TCP/IP over SSH'. Below this, the 'Parameters' tab is selected, displaying various fields for SSH and MySQL configuration. The fields include SSH Hostname, SSH Username, SSH Password (with 'Store in Keychain' and 'Clear' buttons), SSH Key File, MySQL Hostname, MySQL Server Port, Username, Password (with 'Store in Keychain' and 'Clear' buttons), and Default Schema. Each field has a corresponding description on the right.

Field	Value	Description
Connection Name	amazonrds	
Connection Method	Standard TCP/IP over SSH	Method to use to connect to the RDBMS
SSH Hostname	ec2-13-56-184-204.us-west-1.compute.amaz	SSH server hostname, with optional po
SSH Username	ec2-user	Name of the SSH user to connect with.
SSH Password	Store in Keychain ... Clear	SSH user password to connect to the S
SSH Key File	/Users/Malia/Downloads/app_keys.pe ...	Path to SSH private key file.
MySQL Hostname	aale9ucmpeo0cp.crlmjmm3gshz.us-west-1.rd	MySQL server host relative to the SSH s
MySQL Server Port	3306	TCP/IP port of the MySQL server.
Username	admin	Name of the user to connect with.
Password	Store in Keychain ... Clear	The MySQL user's password. Will be rec later if not set.
Default Schema		The schema to use as default schema. l to select it later.

7. Test connections.

8. Jump Server is completed.

Web Application:

- Now that most of the code for web app is ready, it's time to [initialize the Elastic Beanstalk](#) instance using EB CLI:
 - Follow the instructions, adjusting for application needs.
 - Set up SSH for instances.
- Now that EB is initialized, connect RDS to EB.
 - Add EB security group to RDS' acceptable inbound security groups.
 - Within the EB environment configurations, create a variable with connection endpoint and authentication information for RDS.
- Check if the EB is running
 - Check to make sure everything runs fine locally.
 - Using the jump server, export data in the local machine's database to the RDS.
 - Using EB CLI, deploy application.
 - Open EB instance link and see if the webapp is executing properly.
 - If not, use logs to debug.

RDS:

1. Once the jump server and application server are working, there are still more items to take care of. The RDS is still not private and there needs to be a read-only user to connect to Chartio.
2. Modify RDS security group to allow SSH from local IP.

Once the RDS is private, there will be no direct logging in from the local machine. That's why it is important to create the read only user now.

3. SSH into the RDS instance using
`mysql -u {{ username for RDS }} -p -host={{ RDS endpoint }}`
4. Create a read-only user. (Replace 'tester' and 'password' with appropriate values)
 1. `CREATE USER 'tester' IDENTIFIED BY 'password';`
 2. `GRANT SELECT ON *.* TO 'tester';`
5. Now that the jump server and app server are working, modify RDS to private accessibility. Only other instances in the VPC will be able to connect to RDS now.

Network & Security

Subnet group
Use this field to move the DB instance to a new subnet group in another vpc. [Learn more.](#)
default ▼

Security group
List of DB security groups to associate with this DB instance.
Choose security groups ▼
rds-awseb-e-er5yu2qmx-stack-awsebrdsdbsecuritygroup-18nokv6lpfnva-2gop (sg-0dd5ea473ab69c9a9) (vpc-67fadb00) ✕

Certificate authority
Certificate authority for this DB instance
rds-ca-2015 ▼

Public accessibility [Info](#)
☐ Yes
EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You must also select one or more VPC security groups that specify which EC2 instances and devices can connect to the DB instance.
☒ No
DB instance will not have a public IP address assigned. No EC2 instance or devices outside of the VPC will be able to connect.

6. Test to make sure the servers still work. Once this is finished, RDS and Web Application steps are completed! Now all that is left is dealing with Chartio

Chartio:

1. Create a new EC2 instance using the same strategy as we used for the Jump Server.
2. Modify RDS security group to allow inbound traffic from Chartio EC2 instance.

3. Modify RDS security group to allow all outbound traffic.
4. SSH into EC2 instance and [install autossh](#).
5. Put a private key inside Chartio EC2 so Chartio can recognize the EC2 as valid instance.
6. Log in to Chartio and click Add New Datasource.
7. Do not choose RDS as datasource.
8. Add data source as MySQL and use SSH tunnel connection. Fill in variables with the read-only MySQL user created earlier.

Because Chartio only offers direct connections for RDS instances, I used the MySQL SSH tunnel connection option to connect to the private RDS instance.

The screenshot shows the 'Connect a MySQL Data Source' interface in Chartio. The 'SSH Tunnel Connection' tab is selected. The form includes the following fields and options:

- Connect using SSL:** A checkbox that is checked.
- Database Username:** A text input field containing 'admin'.
- Database Password:** A text input field containing seven dots.
- Database Name:** A text input field containing 'monster_teams'.
- SSH public key:** A large, empty text area for pasting a public key.
- Time Zone Support:** A dropdown menu set to 'UTC Offset'.
- Data Source Alias:** A text input field containing 'monster_teams'.

Below the 'Data Source Alias' field, there is a small note: 'This is the nickname we will use for this data source in Chartio.' A blue 'Connect' button is located at the bottom right of the form.

9. Set up SSH Tunnel using RDS Endpoint and location of private key in Chartio EC2 instance. SSH into the Chartio EC2 instance and follow the instructions given. Once finished, type in test connection. If it works, Chartio will now be able to create charts.

Setting up an SSH tunnel

Fill out the following fields so that Chartio can generate connection commands for you:

Database host	<input type="text" value="127.0.0.1"/>
Database port	<input type="text" value="3306"/>
Private key location	<input type="text" value="~/ssh/id_rsa"/>

This should be the private key that matches the public key you gave us.

Before you start your autossh tunnel, test the SSH connection with the following command:

```
ssh tunnel15240@connect.chartio.com -i ~/ssh/id_rsa
```

You should see a Chartio banner along with a welcome message from our server that looks like:

```
You have successfully logged into Chartio! You are logged in as tunnel15240. You are now ready to set up a reverse tunnel with
autossh: https://support.chartio.com/docs/data-sources#tunnel-connection.
```

Once you have determined that the SSH connection is successful, run the following command to start your autossh tunnel:

```
autossh -M 0 -f -N -R 15240:127.0.0.1:3306 tunnel15240@connect.chartio.com -g -i ~/ssh/id_rsa -o ServerAliveInterval=10 -o
ServerAliveCountMax=3 -o ExitOnForwardFailure=yes
```

This command should print nothing and appear to exit immediately. It will be running in the background.

Test Connection

10. Congratulations, the Basic Server Configuration using RDS and Chartio is finished!