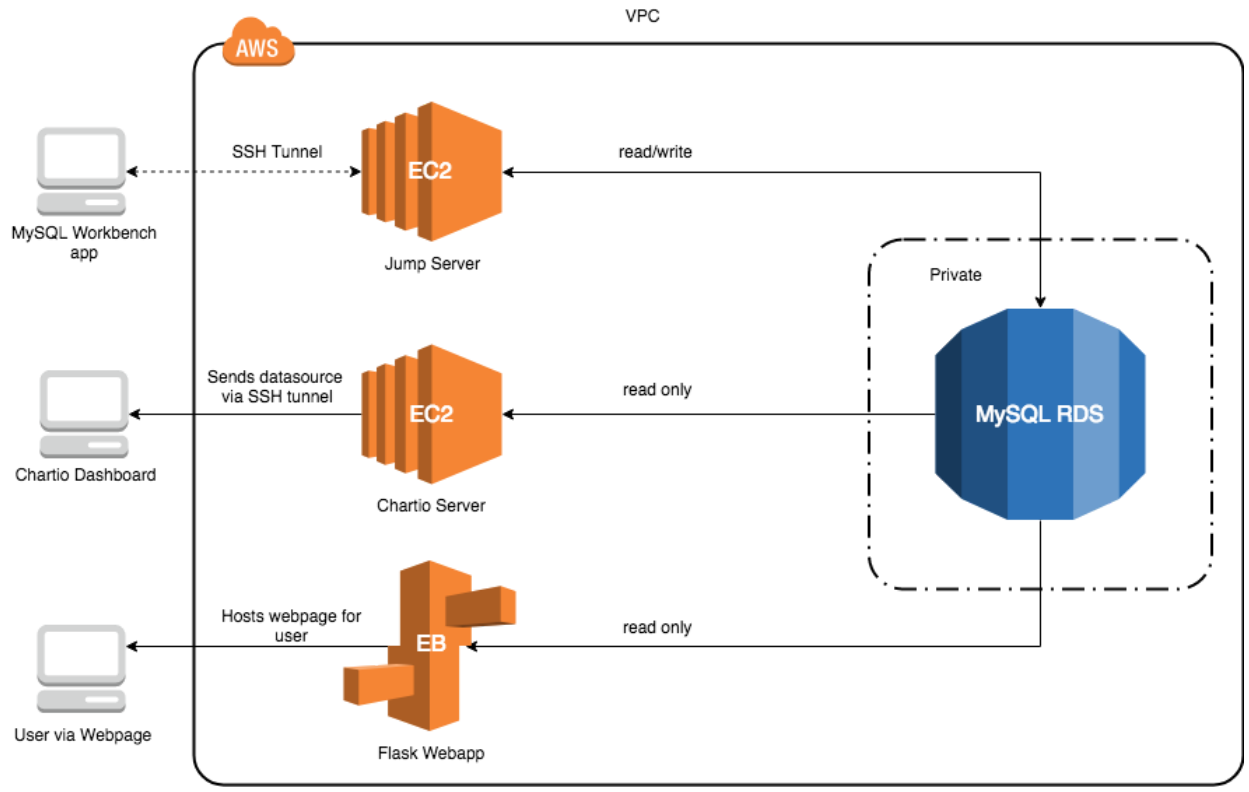
Goal:

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 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

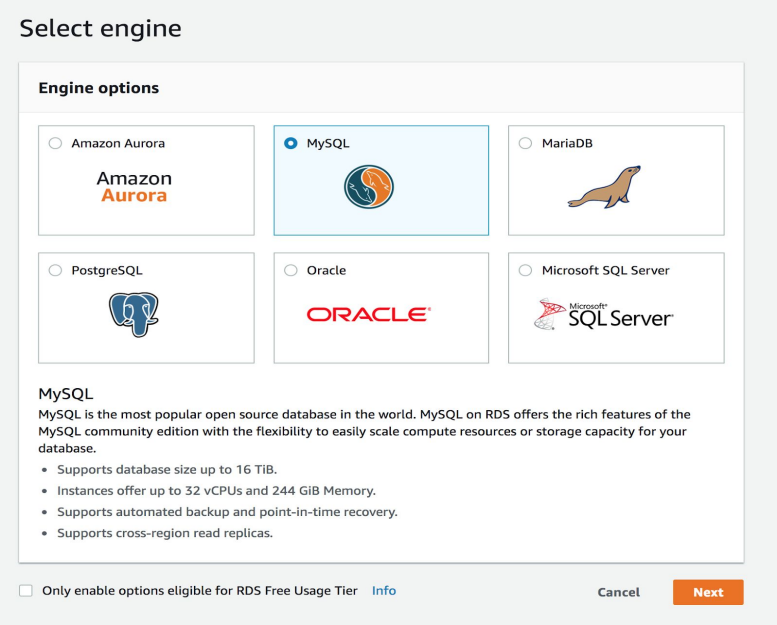
1. `pip install --upgrade awsebcli==3.14.5`
2. `pip install urllib3==1.22`

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.

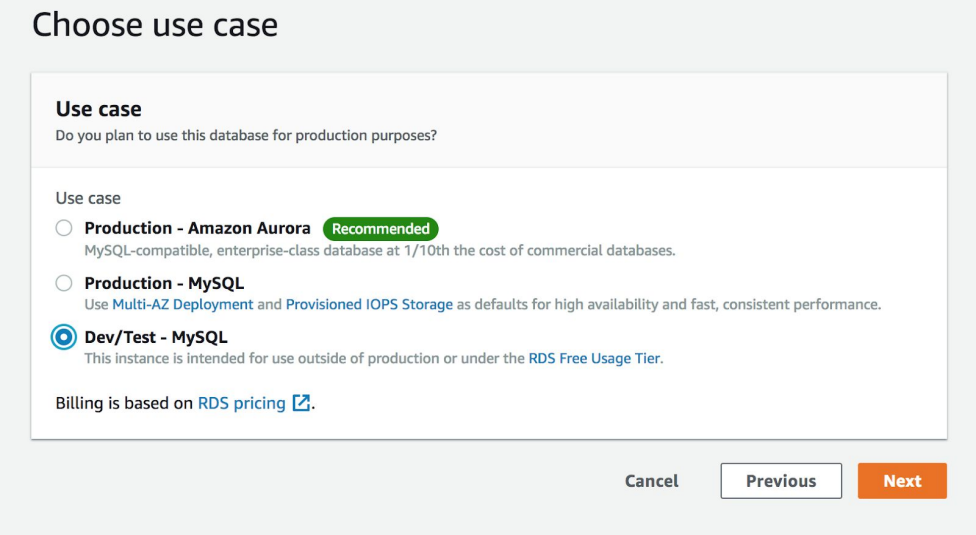
> **Tips about Elastic Beanstalk:**
> - Best practice: Name the file you wish to run application.py.
> - Elastic Beanstalk will not execute main function of application.py.
> - If using Flask, the Flask application object should be named application.

RDS Instance:

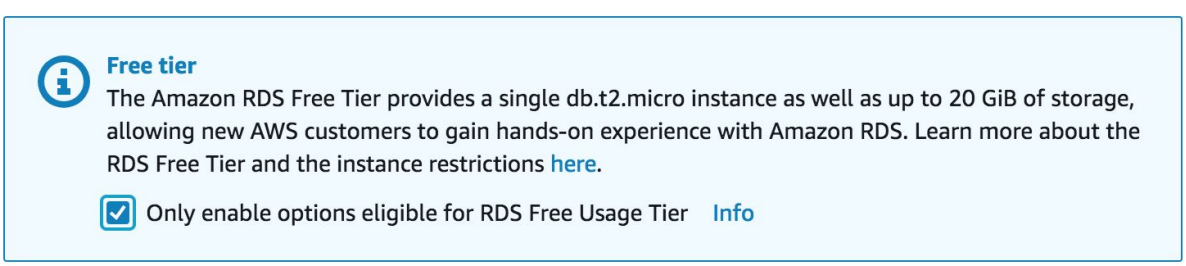
1. Log on to AWS account and go to RDS. Click on Create Database and choose a MySQL engine.

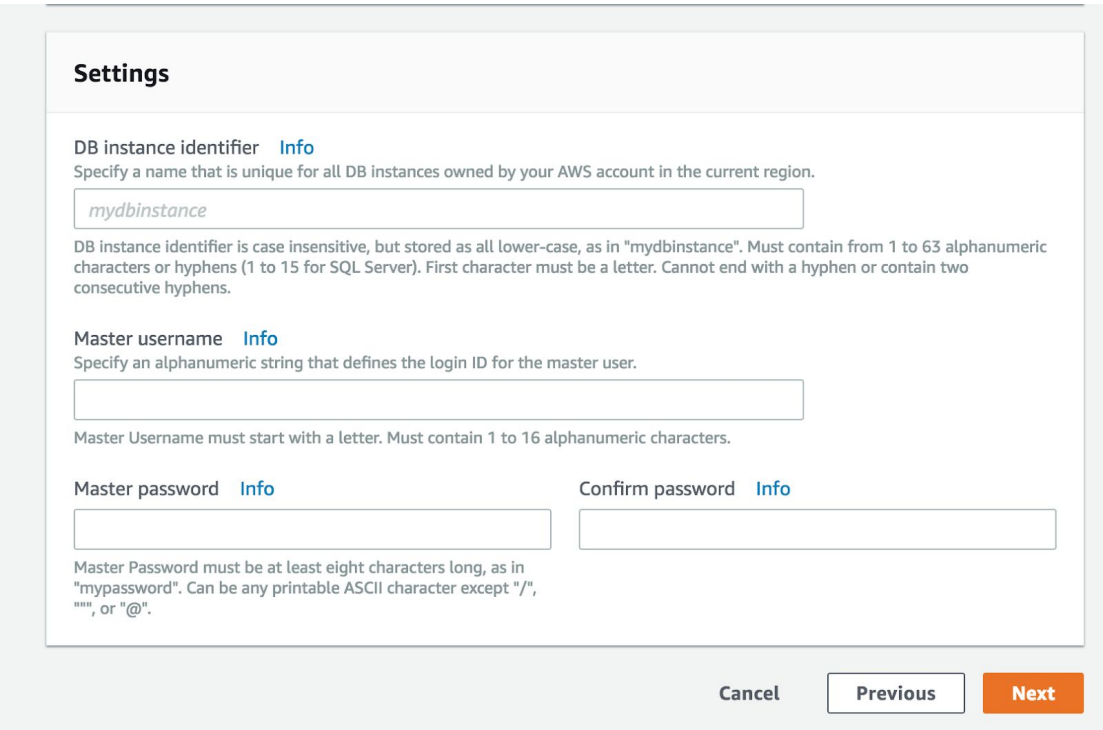


2. Choose Dev/Test as use case.



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





> 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.

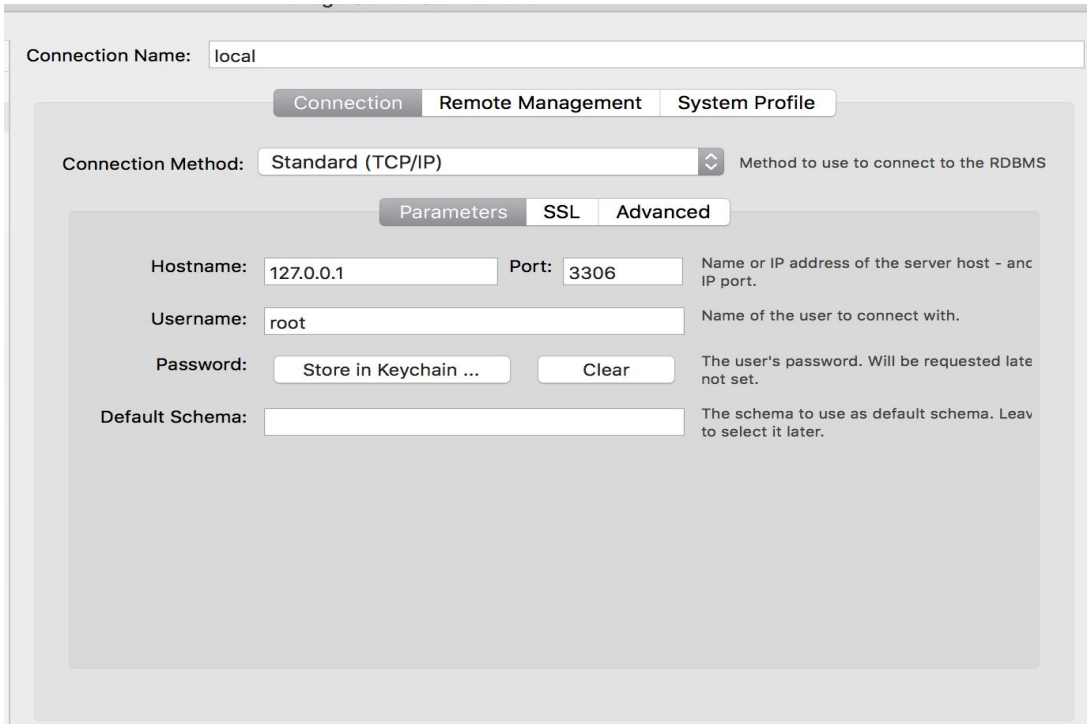
> Do not touch public accessibility **yet**. Everything will be connected to a private RDS eventually, but initially leave it public for testing purposes.

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

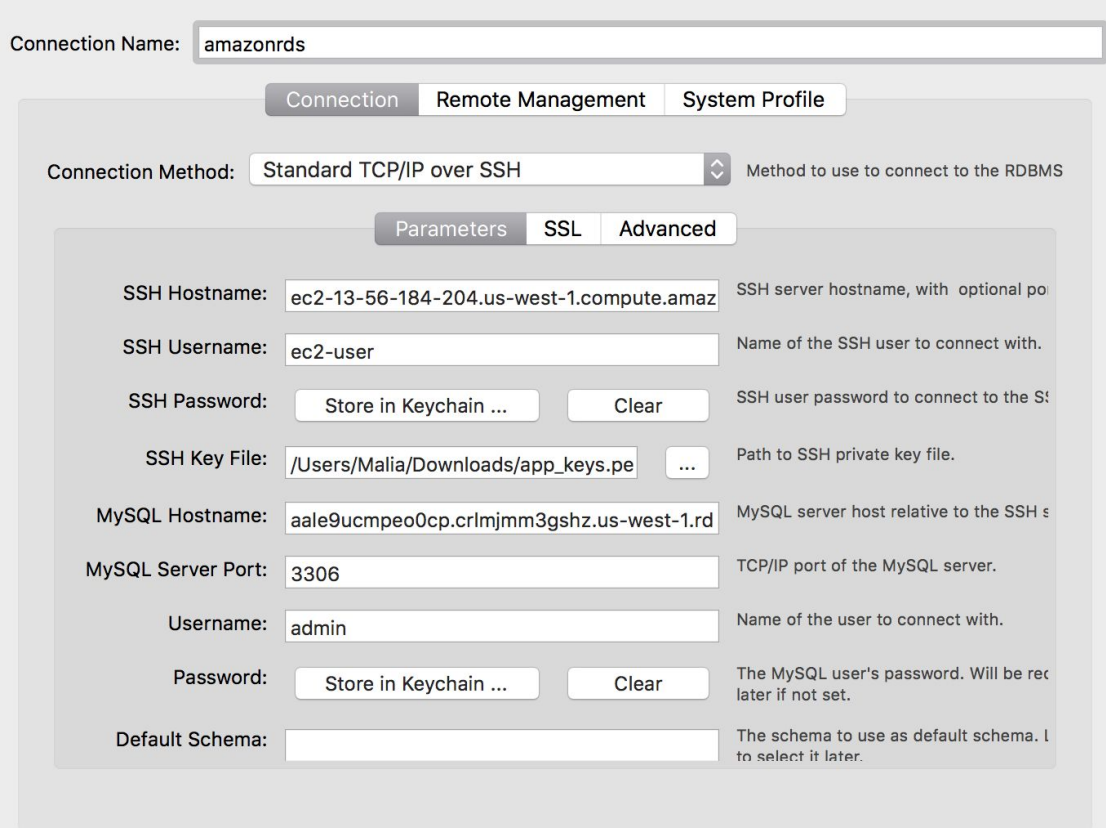
Jump Server:

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.



7. Test connections.
8. Jump Server is completed.

Web Application:

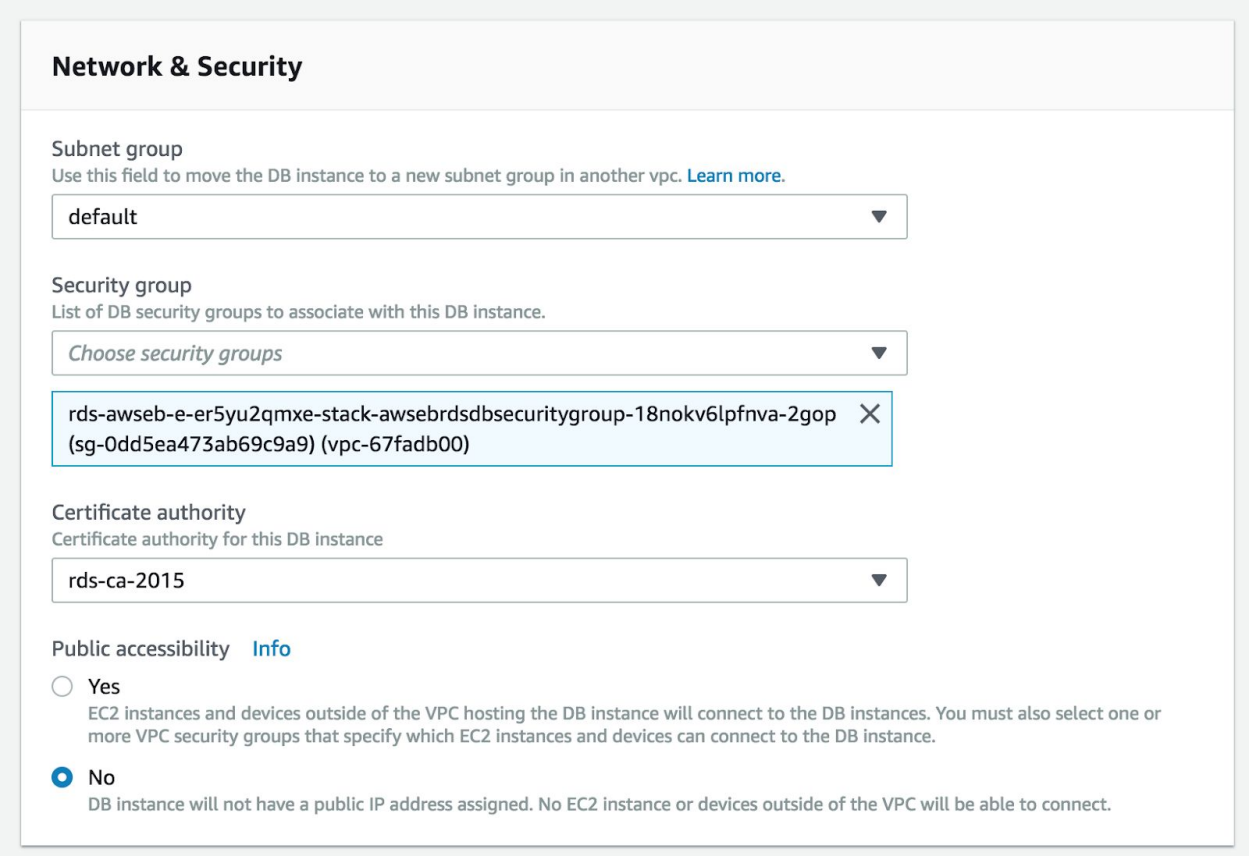
1. Now that most of the code for web app is ready, it's time to initialize the Elastic Beanstalk instance using EB CLI:
    1. Follow the instructions, adjusting for application needs.
    2. Set up SSH for instances.
2. Now that EB is initialized, connect RDS to EB.
    1. Add EB security group to RDS' acceptable inbound security groups.
    2. Within the EB environment configurations, create a variable with connection endpoint and authentication information for RDS.
3. Check if the EB is running
    1. Check to make sure everything runs fine locally.
    2. Using the jump server, export data in the local machine's database to the RDS.
    3. Using EB CLI, deploy application.
    4. Open EB instance link and see if the webapp is executing properly.
    5. 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.



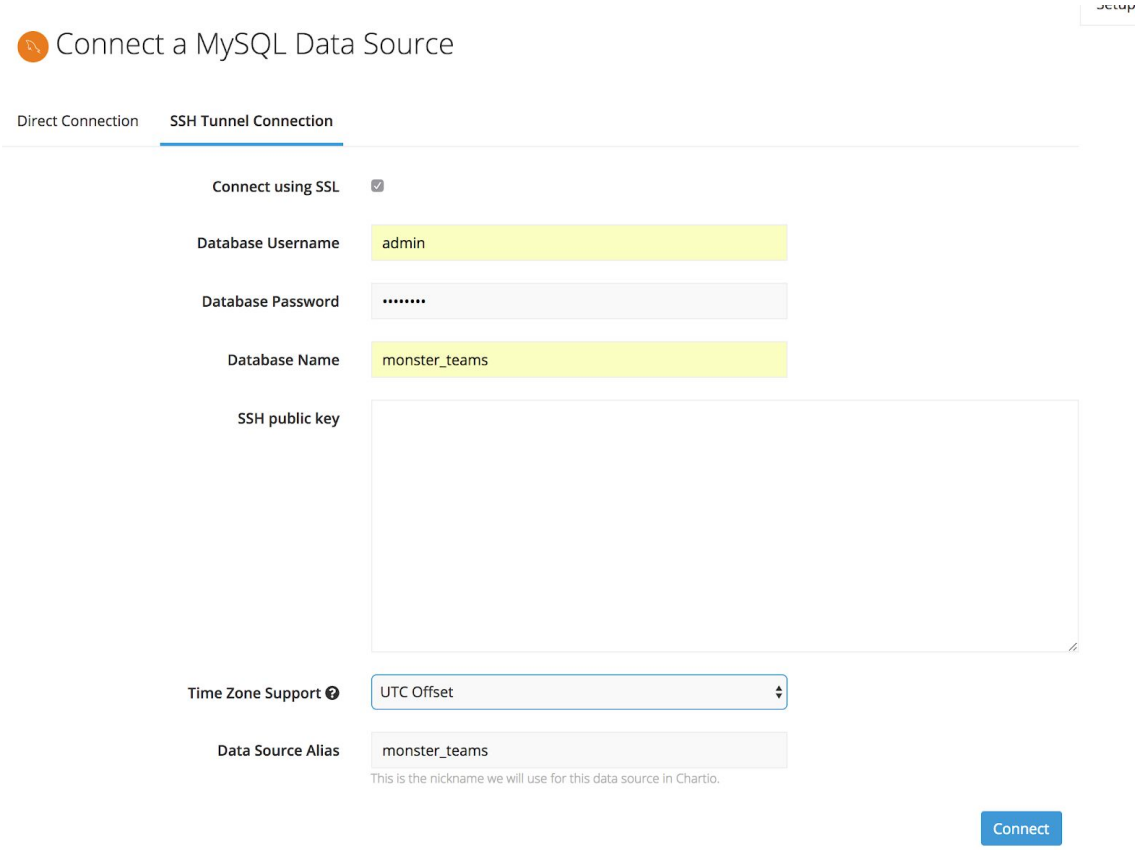
6. Test to make sure the servers still work. Once this is finished, RDS and Web Application steps are completed! Now all that is left is dealing with Chartio

Chartio:

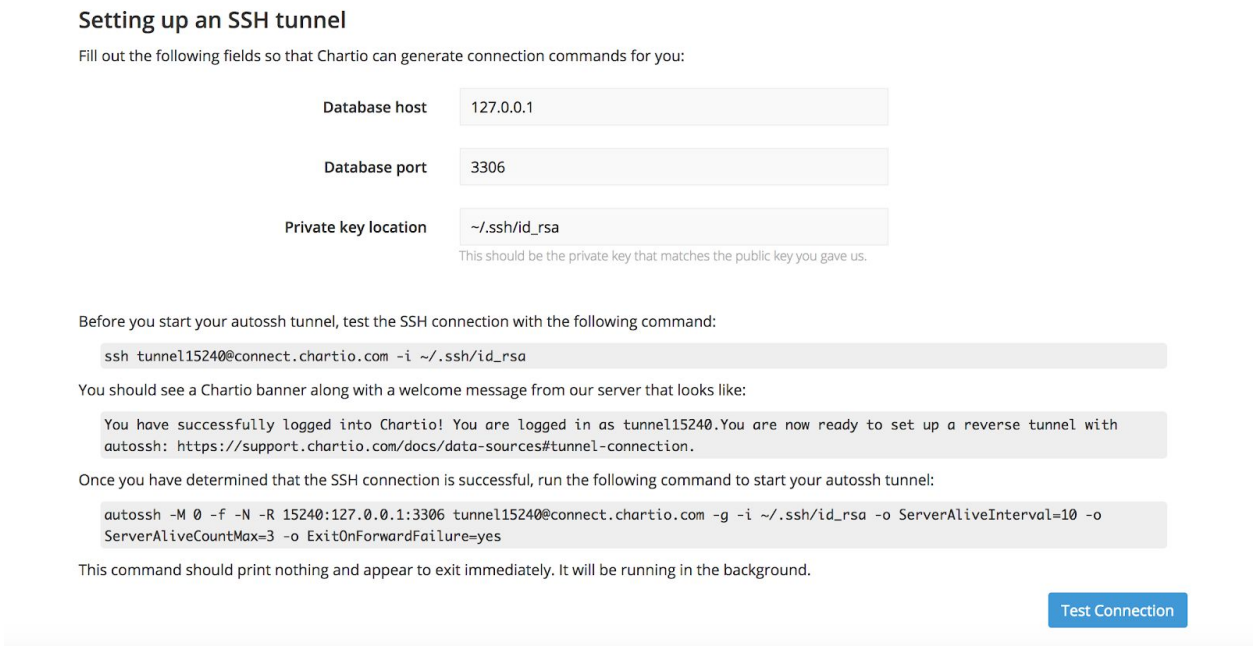
1. Create a new EC2 instance using the same strategy as we used for the Jump Server.
2. Modify RDS security group to allow inbound traffic from Chartio EC2 instance.

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.



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.



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