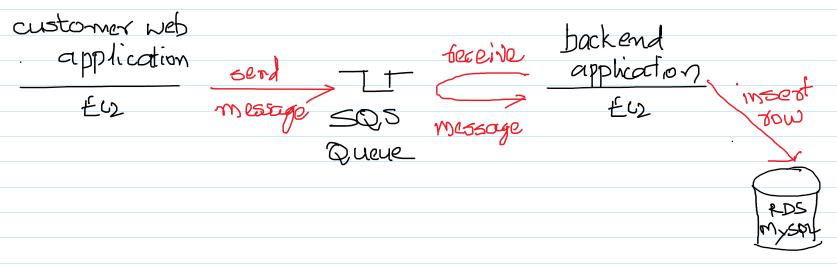
**Use Case**

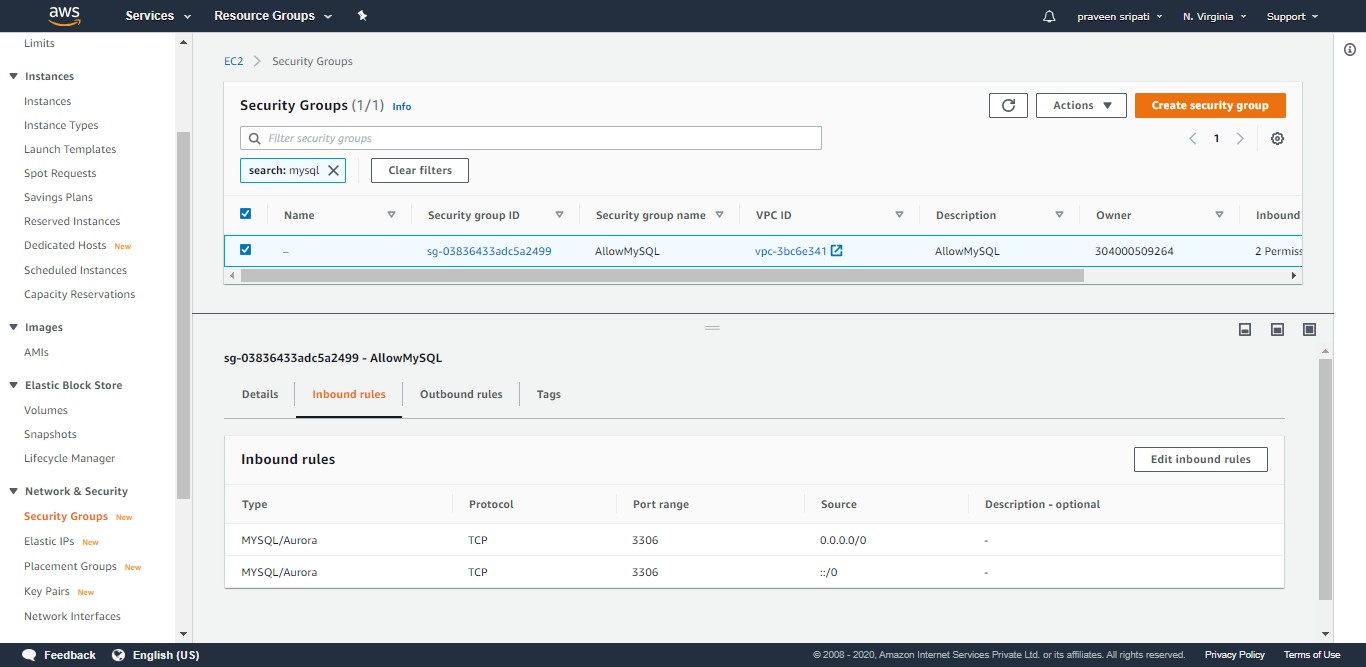
In the traditional way of building applications, the applications directly talk to each other. And if one of the applications goes down, it impacts the other application and some of the data might be lost. This is called tightly coupled architecture.

To make the applications more highly available, the applications don’t talk to each other directly, but through SQS Queue. In the below diagram, the Customer Web Applications interacts with the Backend Applications via SQS Queue. For some reason if the Backend Applications goes down, the Customer Web Application can still continue working with the messages being buffered in the SQS Queue. Once the Backend Application is up, it can start polling the messages from the SQS Queue and update the database. This way none of the messages are lost and applications are loosely coupled and not aware of the status of each other.



**AWS Services:** SQS, EC2, IAM and RDS

-- Make sure to create a Security Group with the port 3306 allowed in the “Inbound rules”.



-- Go to the RDS Management Console and click on “Create Database”.

A screenshot of a social media post

Description automatically generated

-- Select MySQL as the Database.

A screenshot of a social media post

Description automatically generated

-- Select “Free tier” for the templates. This will make sure all the settings for the RDS are for the AWS Free Tier (<https://aws.amazon.com/free/>). Specify the “DB instance identifier” as CustomerDB.

A screenshot of a social media post

Description automatically generated

-- Specify the username and the password.

A screenshot of a social media post

Description automatically generated

-- Uncheck “Storage autoscaling”.

A screenshot of a social media post

Description automatically generated

-- Under Connectivity, for “Publicly accessible” as Yes.

A screenshot of a social media post

Description automatically generated

-- Select the AllowMySQL Security Group created earlier.

A screenshot of a social media post

Description automatically generated

-- For the “Initial database name” specify “customer\_db”. Disable the Backup.

A screenshot of a social media post

Description automatically generated

-- Finally click on “Create Database”.

A screenshot of a social media post

Description automatically generated

-- The database would be initially in a Creating Status for a few minutes.

A screenshot of a social media post

Description automatically generated

-- After a few minutes the database should be in available status and the endpoint would be populated.

A screenshot of a cell phone

Description automatically generated

-- Download the HeidiSQL (<https://www.heidisql.com/download.php>) by clicking on “Installer, 32/64 bit combined” and install it as any other application.

A screenshot of a cell phone

Description automatically generated

-- Launch HeidiSQL once the installation is complete and enter the RDS/MySQL endpoinp/user/password. And finally click on Open.

A screenshot of a cell phone

Description automatically generated

-- Click on Yes when prompted. This will store the connection details.

A screenshot of a cell phone

Description automatically generated

-- In the HeidiSQL make sure to select customer\_db in the top left pane, go to the Query tab, copy the below DDL statement and click on Execute to create a table.

*CREATE TABLE customers (  
 name VARCHAR(30) NOT NULL,  
 address VARCHAR(30) NOT NULL  
);*

A screenshot of a social media post

Description automatically generated

-- Go to the SQS Management Console and click on “Create Queue”.

A screenshot of a cell phone

Description automatically generated

-- Enter the Queue name as “CustomerQueue”, go with the default options and click on “Create queue”.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

-- Note down the Queue URL and the ARN.

A screenshot of a social media post

Description automatically generated

-- Go to the IAM Management Console and go to Policies. Click on “Create Policy”.

A screenshot of a cell phone

Description automatically generated

-- Click on the JSON tab and paste the below JSON. Make sure to replace the Queue ARN with the one got from the previous step. Click on “Review Policy”.

*{  
 "Version": "2012-10-17",  
 "Statement": [{  
 "Sid":"SQSSendMessage",  
 "Effect": "Allow",  
 "Action": "sqs:SendMessage",  
 "Resource": "****arn:aws:sqs:us-east-1:304000509264:CustomerQueue****"  
 }]  
}*A screenshot of a social media post

Description automatically generated

-- Enter the policy name as “SQS\_SendMessage\_Policy” and click on “Create policy”.

A screenshot of a cell phone

Description automatically generated  
  
-- Click on Roles and click on “Create Role”.

A screenshot of a computer

Description automatically generated

-- Select EC2 as the service and click on “Next Permissions”.

A screenshot of a cell phone

Description automatically generated

-- Select the Policy create in the previous step and click on “Next Tags”.

A screenshot of a cell phone

Description automatically generated

-- Tags are optional. Click on “Next Review”.

A screenshot of a social media post

Description automatically generated

-- Enter the Role name as SQS\_SendMessage\_Role and click on “Create Role”.

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

-- Create another Policy called “SQS\_ReceiveDeleteMessage\_Policy” with the below JSON. Make sure to replace the SQS Queue ARN. Create a role called “SQS\_ReceiveDeleteMessage\_Role” and attached the Policy to it.

*{  
 "Version": "2012-10-17",  
 "Statement": [{  
 "Sid":"SQSReceiveDeleteMessage",  
 "Effect": "Allow",  
 "Action": ["sqs:ReceiveMessage" , "sqs:DeleteMessage"],  
 "Resource": "****arn:aws:sqs:us-east-1:304000509264:CustomerQueue****"  
 }]  
}*

-- At the end two Policies and two Roles should be created as shown below.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

-- Create two EC2 Instances and name them as "CustomerWebApplication" and "BackendApplication" with the below details. Make sure to select the KeyPair.

- Ubuntu 18.04  
 - t2.micro  
 - SecurityGroup with 22/SSH inbound

A screenshot of a social media post

Description automatically generated

-- Login to the EC2 instances via Putty. Execute the below commands on both the EC2 instances. See the exception.

*#become root  
sudo su  
  
#get the list of softwares  
apt-get update  
  
#install python  
apt-get install -y python2.7 python-pip  
  
#install Python AWS SDK and MySQL drivers  
pip install boto3  
pip install mysql-connector-python* ***(only on the "BackendApplication" EC2 Instance)***

*exit  
mkdir .aws  
echo -e "[default]\nregion=us-east-1" > .aws/config*

-- Attach the SQS\_SendMessage\_Role Role to the "CustomerWebApplication" EC2 Instance.

Attach the SQS\_ReceiveDeleteMessage\_Role Role to the "BackendApplication" EC2 Instance.

-- Create a file send\_message.py with the below code on the "CustomerWebApplication" EC2 instance. Make sure to change the queue\_url. We are trying to mimic a web application where the customers can create an account and the customer details are put into a Queue.

*import sys  
import boto3  
  
sqs = boto3.client('sqs')  
queue\_url = '****https://sqs.us-east-1.amazonaws.com/304000509264/CustomerQueue****'  
  
response = sqs.send\_message(  
 QueueUrl=queue\_url,  
 MessageBody=(sys.argv[1])  
)  
  
print(response['MessageId'])*

-- Execute the below commands to put the messages in the Queue.

*python send\_message.py Praveen,Hyderabad  
python send\_message.py Prajval,Delhi  
python send\_message.py Kusuma,Mumbai  
python send\_message.py Prathibha,Bangalore*

A screenshot of a cell phone

Description automatically generated

-- Create a file get\_message\_write\_to\_rds.py with the below code on the "BackendApplication" EC2 instance. Make sure to change the queue\_url and the database details. The below program reads the message from Queue and inserts a row in the RDS/MySQL Database.

*import time  
import boto3  
import mysql.connector  
  
queue\_url = '****https://sqs.us-east-1.amazonaws.com/304000509264/CustomerQueue****'  
  
#Specify the database details  
host = '****customerdb.cmeeo0ikklen.us-east-1.rds.amazonaws.com****'  
user = '****praveen****'  
password = '****praveen123'*** *database='****customer\_db****'  
  
#Create a SQS Client  
sqs = boto3.client('sqs')  
  
#Connect to the RDS MySQL Instance  
mydb = mysql.connector.connect(host=host, user=user, password=password, database=database)  
mycursor = mydb.cursor()  
  
# Receive message from SQS queue  
response = sqs.receive\_message(QueueUrl=queue\_url)  
message = response['Messages'][0]  
  
# Delete received message from queue  
receipt\_handle = message['ReceiptHandle']  
  
sqs.delete\_message(  
 QueueUrl=queue\_url,  
 ReceiptHandle=receipt\_handle  
)  
  
print('Received and deleted message: %s' % message["Body"])  
  
#Get the customer name and address from the message  
customerDetails = message["Body"]  
customerDetailsList = customerDetails.split(',')  
name = customerDetailsList[0]  
address = customerDetailsList[1]  
  
#Write the record to the database  
val = (name, address)  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
  
mycursor.execute(sql, val)  
mydb.commit()  
print("Record inserted in the DB")*

-- Wait for a few minutes and execute the “python get\_message\_write\_to\_rds.py” command to get the message from the Queue and insert a row the database.

Here we are trying to mimic the backend application being down. Notice that the messages between the web application are not lost inspite of waiting for a few minutes or the backend application being down. This is how Highly Available applications are built.

A screenshot of a cell phone

Description automatically generated

-- Go back to the HeidiSQL and execute the “select \* from customers” query to fetch the customers details.

A screenshot of a social media post

Description automatically generated