CS 310
Scalable Software Architectures
Fall 2023

Project #01 – Part 01 (v1.0)

Assignment: PhotoApp with AWS S3 and RDS
Submission: Gradescope (part 02)
Policy: individual work only, late work is accepted
Complete By: Part 01 and 02 by Friday October 6th @ 11:59pm CST
Late submissions: see syllabus for late policy… No submissions accepted after Sunday 10/8 @ 11:59pm

Pre-requisites: assignment "01 Getting Started"

Overview

We're going to build a cloud-native application **PhotoApp** that works with images, allowing users to upload, download, and manipulate their own images. Here in project 01 we'll lay the foundation by using AWS + Python to do the following:

1. Use S3 to store images
2. Use RDS (in particular MySQL) to keep track of users and their images
3. Use IAM to create users and policies for access

Note that the goal of this assignment is not to build something novel --- there are tons of apps that work with images. Instead, think of the images as *assets*, which could vary from law documents to large financial datasets, and think of this project as an architectural example of how to work with these assets.

Piazza, not email

There is simply too much email in the world. So please do not use email for communication with the course staff; in general all class-related emails will be ignored. The one exception are matters of a personal nature, in which case you can (and should) email the professor. Appropriate personal matters include family emergencies, classroom issues, and unresolved grading issues.

To provide another means of assistance, we will be using a forum-based web site called ***Piazza***. Piazza is your replacement for email. Piazza allows students and staff to help one another, reducing the *time-to-answer*. The staff will check Piazza throughout the day — you should get in the habit of doing the same. Before posting, please search Piazza as the answer to your question may already be online. When posting, please follow these guidelines:

1. *Look before you post* — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post a question. Posts are categorized to help you search, e.g. "HW".

2. Post publicly — only post privately when asked by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Be respectful. Anonymous posts are allowed, but note that such posts are *not* anonymous to the staff.
4. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means…".
5. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, so don't hesitate to take a screenshot and post; see http://www.take-a-screenshot.org/ .
6. Don't post large blocks of code / possible answerws — if you do, you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question --- in that case post only the fragment that denotes your question, and omit whatever details you can. If you must post a large block of code, then do so privately --- there's an option to create a private post ("visible to staff only").

You should have received an invitation to join Piazza (by email of course). If not, you can join via this link.

Academic Conduct Policy

Northwestern publishes a basic guide to academic integrity, which can be found here. In summary, here are NU's eight cardinal rules of academic integrity:

1. *Know your rights*
2. *Acknowledge your sources*
3. *Protect your work*
4. *Avoid suspicion*
5. *Do you own work*
6. *Never falsify a record or permit another person to do so*
7. *Never fabricate data, citations, or experimental results*
8. *Always tell the truth when discussing your work with your instructor*

School policies and more information can be found on NU's academic integrity website. With regards to CS 211, unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk and learn from the course staff, peers, and others, this interaction must be superficial with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The use of AI (ChatGPT, Co-pilot, etc.) is currently forbidden.

Examples of what is not allowed? Downloading work from a github repository and submitting it as your own, whether partial or complete. Downloading answers from StackOverflow and submitting them as your own. Emailing your work to another student, or receiving work from another student. Sharing your screen with another student so they can see your work (and potentially submit it as their own). Participating in a screen share and taking screenshots or photos of someone else's work, and then using that as a guide to submit your own work. Copying answers posted to Piazza, making a few simple changes, and then submitting as your own work. Allowing someone else to write / type the answer for you. Using AI (ChatGPT, Co-pilot, etc.) to generate code for you which you then submit as your own.

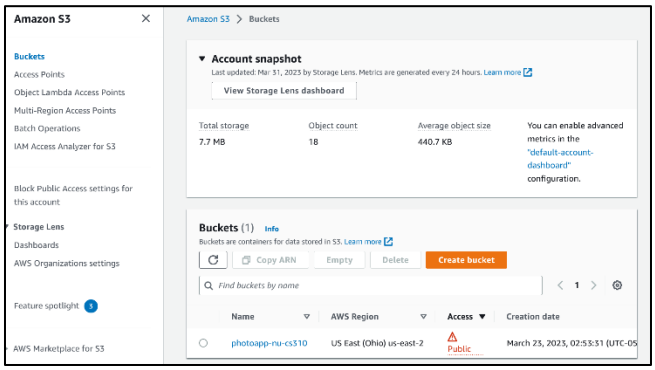
Okay, so what is allowed? Talking to the instructor or course staff, and getting insights --- but not direct answers --- to help you solve the assignment. Talking to other students about the assignment, using diagrams / natural language / pseudo-code to convey ideas. Searching the internet for guidance, and using that guidance to help you form your own solution. If you do receive help / guidance, it's always best to cite your source by name or URL just in case there is a question as to where the work came from.

Before you start…

Before you start, you'll need three things. First, you need an AWS account. If you haven't already, you can signup for a free account here.

Second, you will need a Python programming environment. You can use whatever IDE you prefer, popular choices include VS Code, PyCharm, and Spyder. If you prefer not to install anything on your local machine, you are welcome to use replit.com --- a Python replit will be provided for each class project. If you haven't already, you can join our replit team here.

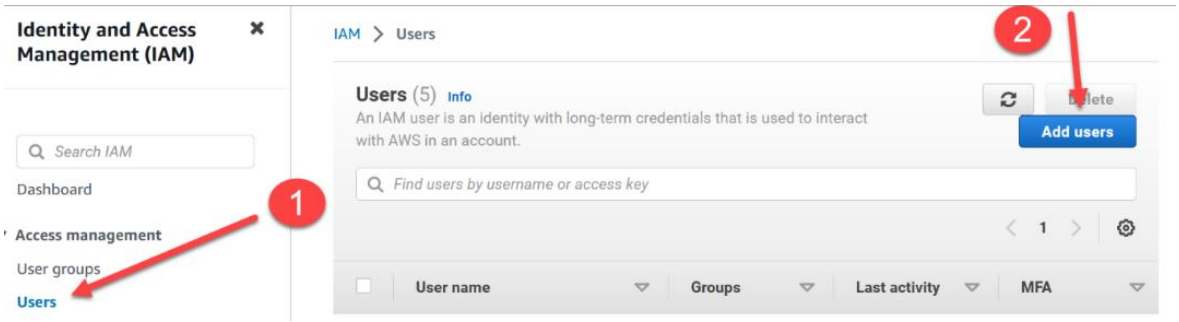
Step 1: Setup AWS CLI



There are various ways to work with AWS. The two most common are logging in to the web site, and using the **Management Console**, Amazon's web-based GUI ----------> This implies a visual interface with lots of navigation and button-clicking. Great for exploring and learning, but hard to remember what you did and hard to repeat over and over again. Once you learn your way around, most would agree that a **Command-Line Interface** (CLI) is preferred, especially when you can save those commands into files for repeated execution. For example, once the CLI is setup on your local machine, you can use a command window / terminal to make a new S3 bucket as follows:

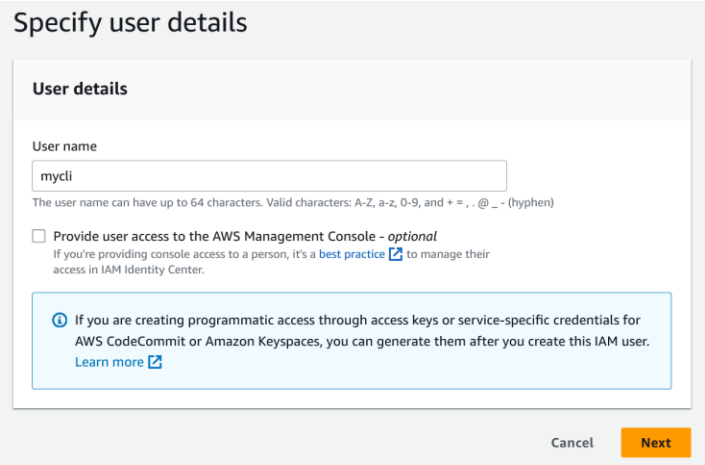
```
aws s3 mb s3://new-bucket-nu-cs310
```

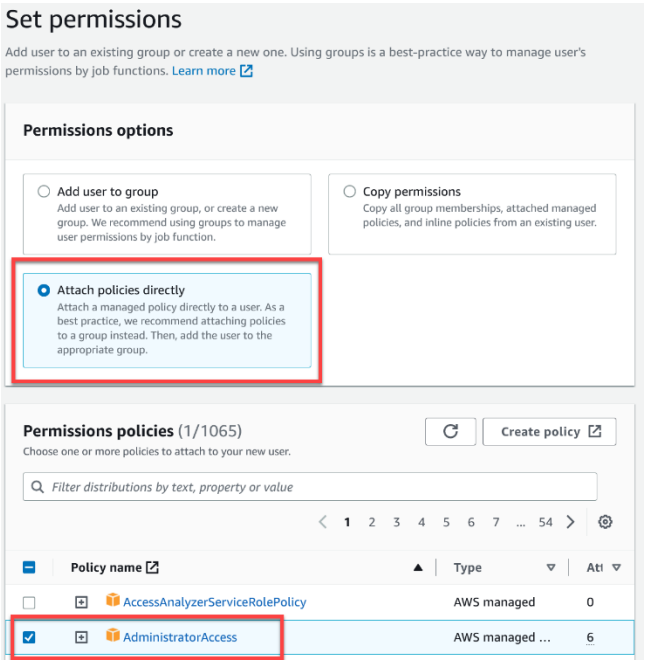
To setup your computer for CLI access, start by logging into AWS, opening the Management Console, and search for "IAM" to bring up the Identify and Account Management dashboard. Click on "Users" along the left the list of users that have access to your AWS account:



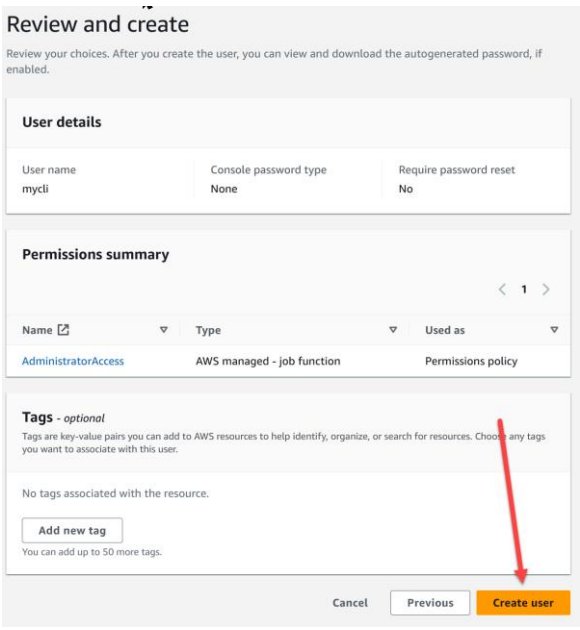
Click the "Add users" button, and then continue on the next page…

As discussed in the textbook, create a new user named “mycli” and setup their permissions as shown below:



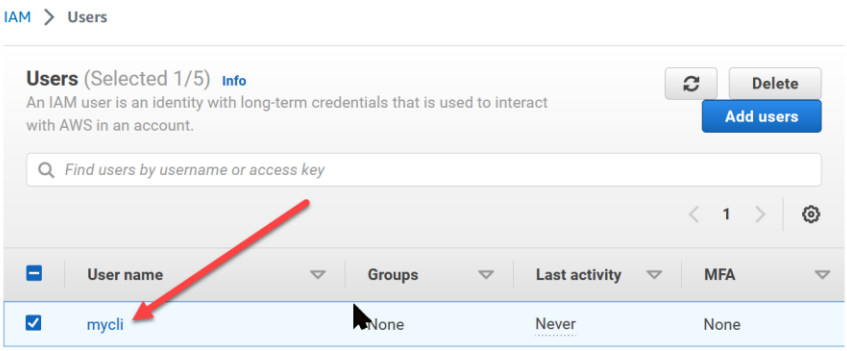


Now review and create the new user:

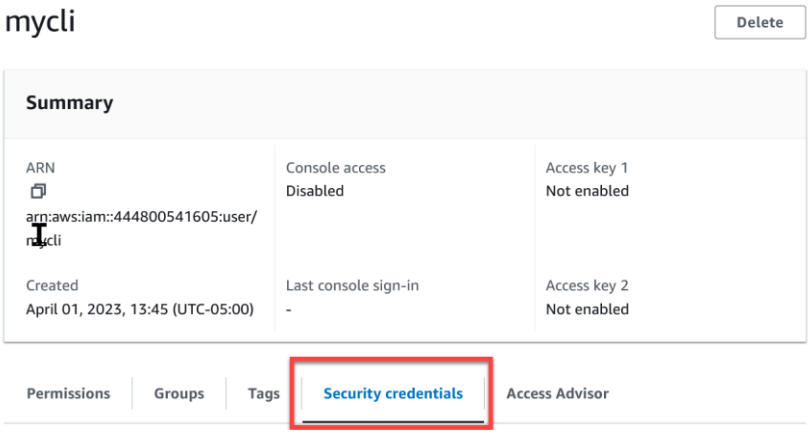


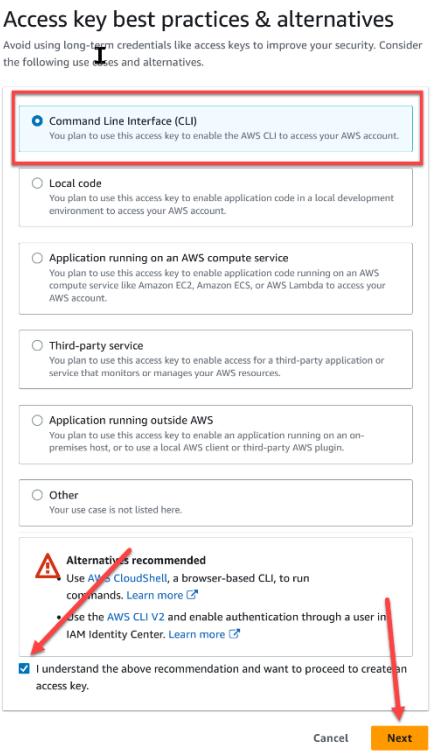
Continued on next page…

Once you create your "mycli" user, they will be listed as one of your users. Click on the name:

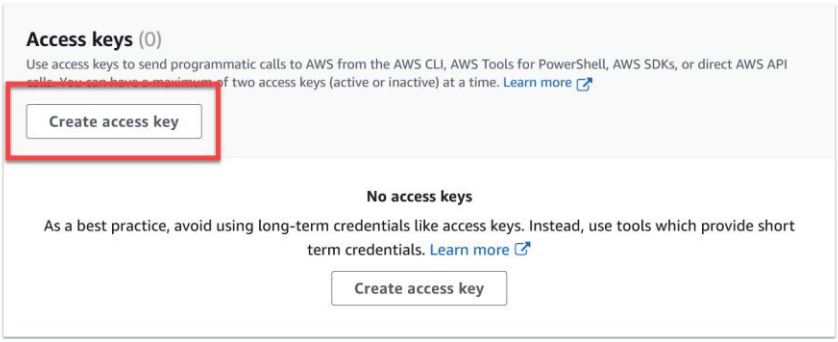


Now let's get the user's access and secret key (ignoring best practices for now):

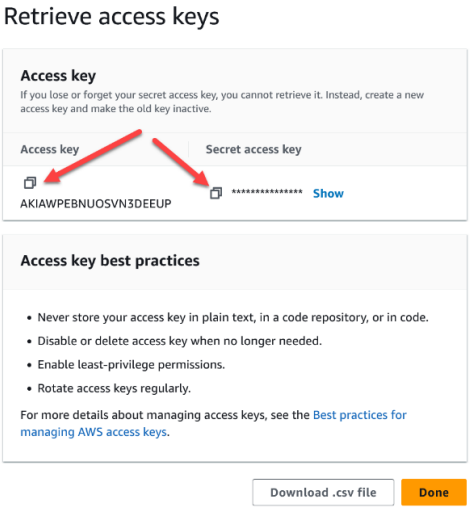




Scroll down to "Create access key", and create an access key for CLI access:



After creating the access key, you'll see this dialog --- stop and keep this dialog open.



Now let's install the AWS CLI SDK v2, here's the link:

https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html

Once installed, open a command window (Windows) or terminal (Mac). Configure your local machine by running "aws configure". Enter the access and secret keys --- note the small buttons on the GUI page to copy each of these to the clipboard. For the region enter "us-east-2", and "json" for the default format. Your local computer should now be setup for CLI access. Test using these commands:

```
aws --version

aws s3 mb s3://new-bucket-YOUR-NAME-RANDOM-NUMBER
```

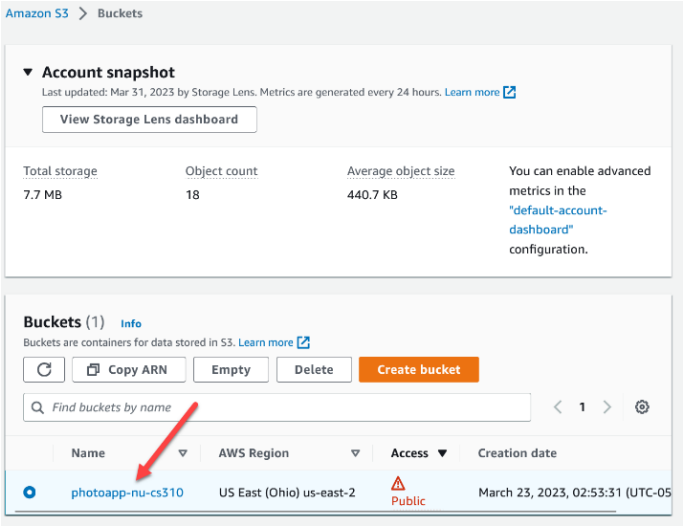
The second command creates a bucket; recall that bucket names must be unique in each region, so you need to personalize the bucket name. To confirm that the bucket was created, use the Management Console (search for "S3"), or better yet, use the CLI to list the contents of your S3:

```
aws s3 ls
```

Step 2: Create photoapp bucket

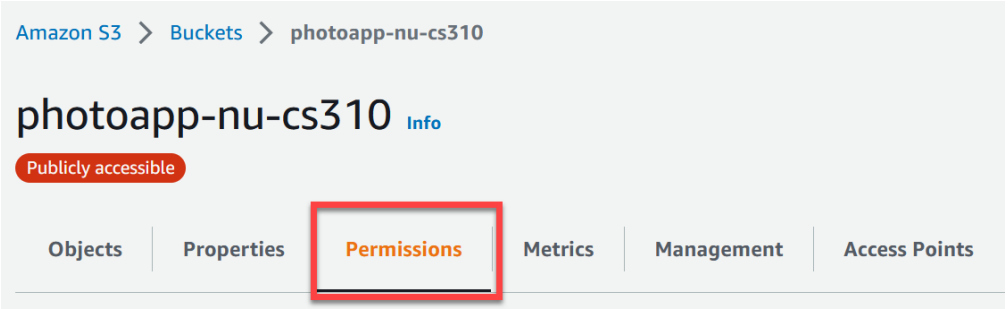
Step 2 is to create a new S3 bucket for your **PhotoApp** application. Using the Management Console (search for "S3"), or use the CLI. The name of your bucket should start with "photoapp" then follow with your name and if necessary a unique identifier of some sort. Example: our staff bucket is "photoapp-nu-cs310".

For testing purposes, we want your bucket to be publicly readable. After creating your bucket, in the Management Console find your bucket and click on the name:

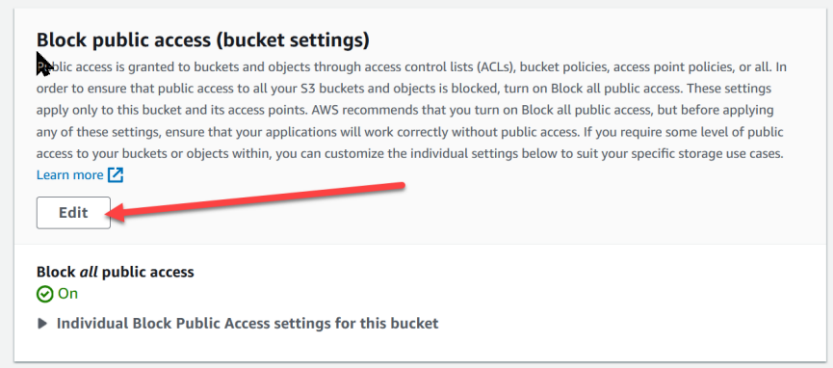


<< next page >>

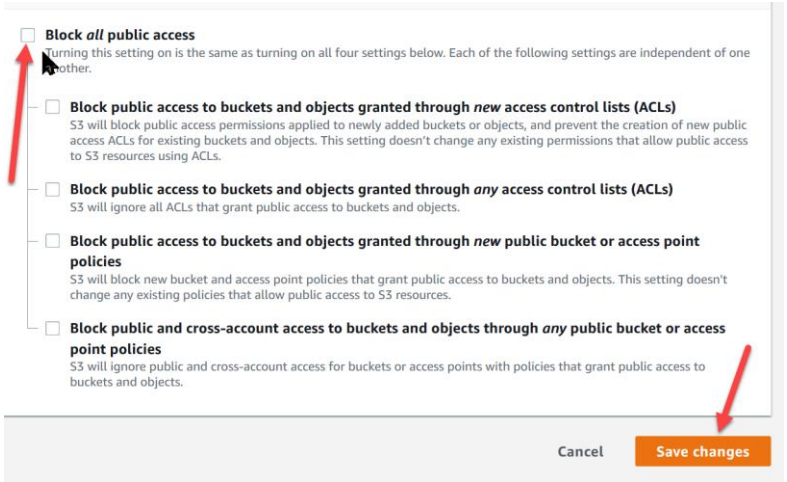
Switch to the Permissions tab:



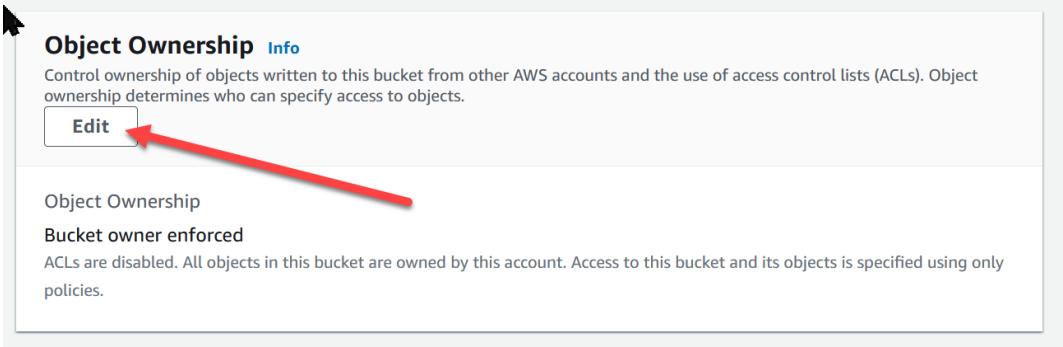
Edit the public access settings:

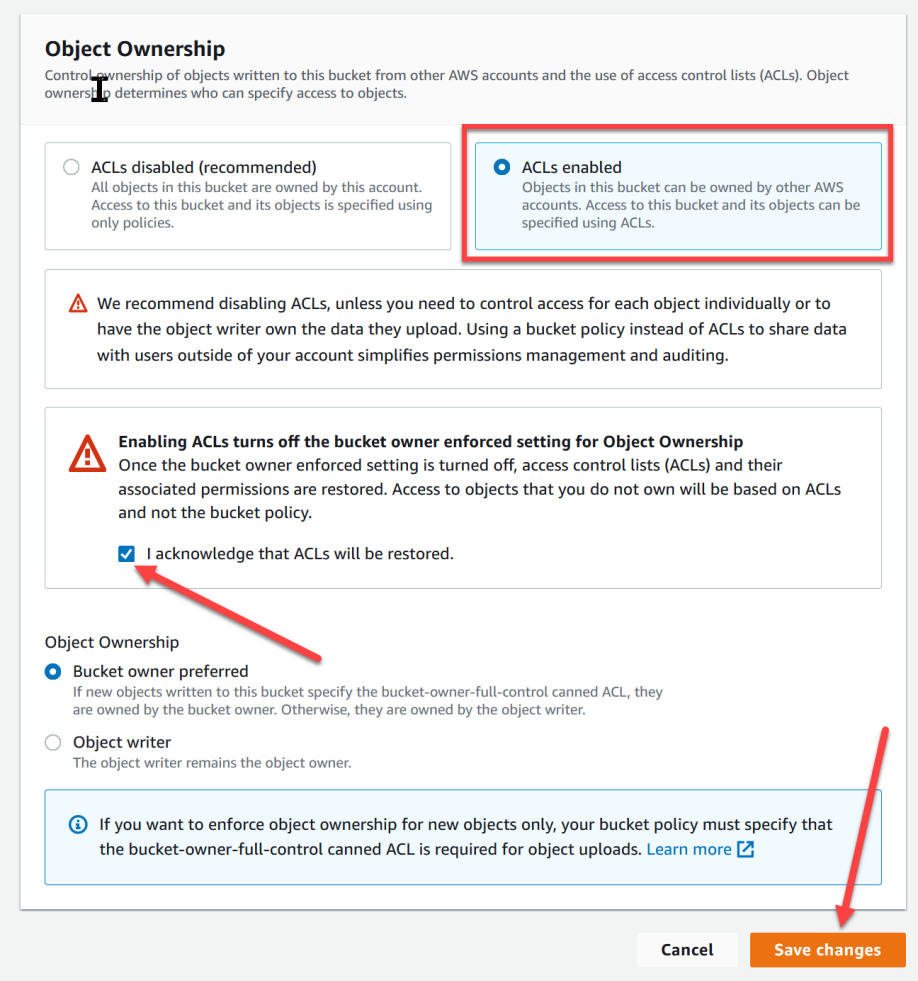


<u>Uncheck</u> the option to block all public access, thereby ALLOWING public access, and Save changes:



Now we need to enable ACLs, or access control lists. ACLs are how permissions work in most operating systems, where access is controlled by 3 aspects: List, Read, and Write. We want to enable ACLs for the bucket, so do the following and save changes:

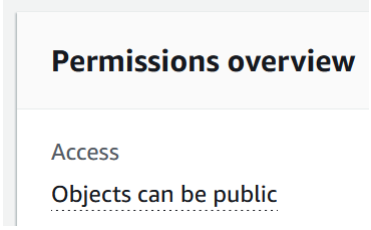


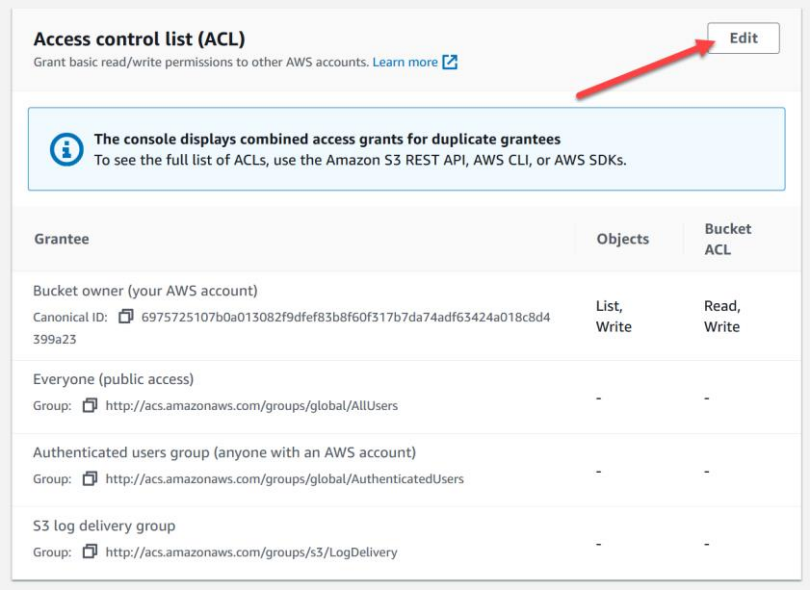


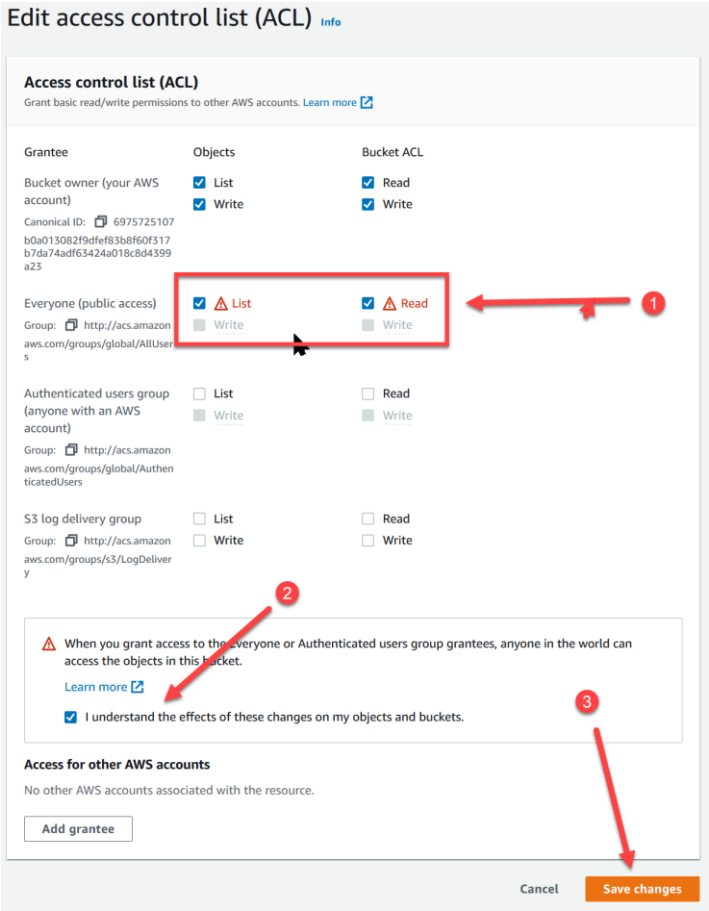
<< continued on next page >>

At this point the bucket is now public, and its objects *can* be public.

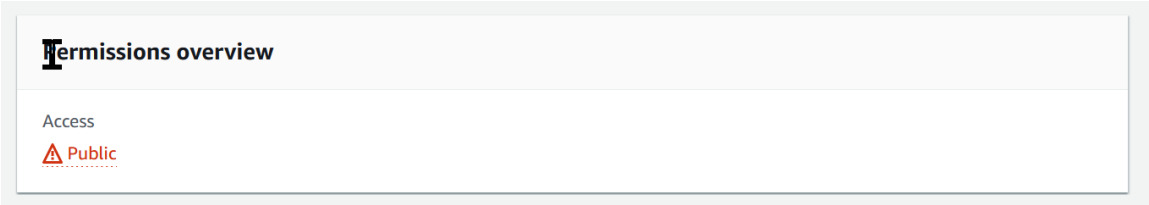
The last step is to actually make objects public. We need to configure the bucket's ACL to allow public access:







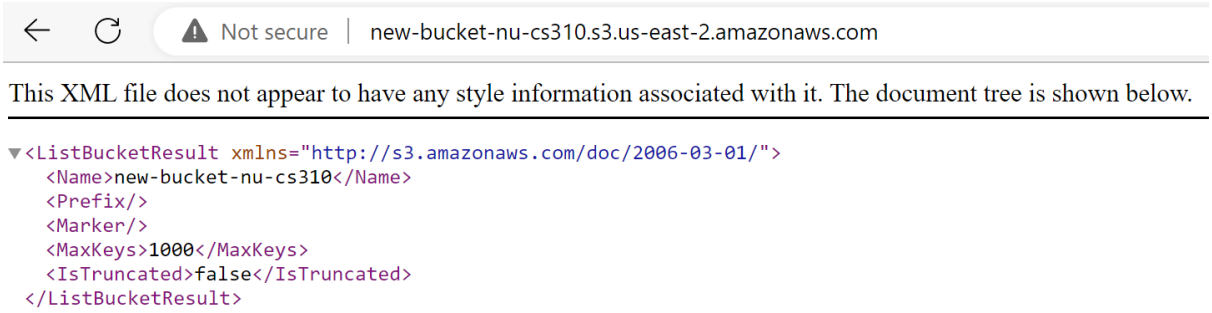
At this point, the bucket "Permission overview" should say that all access is now Public:



If all is well, you should now be able to open a browser and view the contents of your new bucket:

http://**new-bucket-nu-cs310**.s3.us-east-2.amazonaws.com/

Replace "new-bucket-nu-cs310" with the name of your bucket, and you should see something like this:

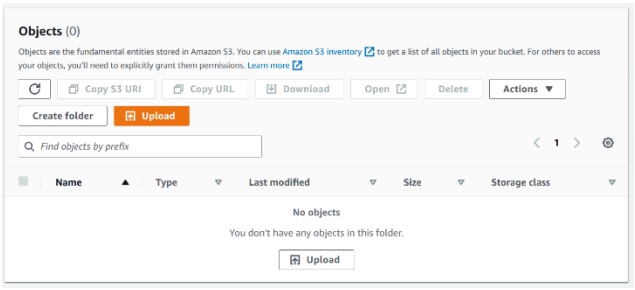


If it's not working, delete the bucket you created and start again? Did you create in the correct region? Are you configured for the correct region if using the CLI?

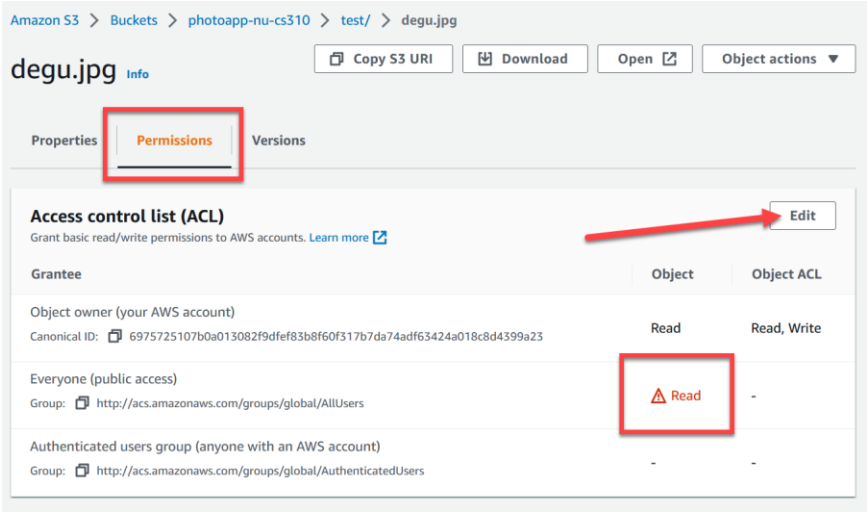
Step 3: Manually upload some photos

Step 3 is to manually upload some photos to the bucket. We're doing this to make sure things are setup properly, then in a later step we'll upload programmatically using Python. But one step at a time…

Using the Management Console (or CLI), create a new folder in your bucket named "test". Note that AWS will automatically add the "/" as part of the folder name. Click on the folder name to navigate into the folder. Use the Upload button to upload one or more images to your test folder, don't worry about the file names, it can be whatever you want, though upload JPG images since we know those will display correctly in replit.



It turns out that uploads do *not* inherit bucket permissions, so if we want the newly-upload images to be publicly readable, we must explicitly grant that permission. For each image you uploaded, click on the image in the Management Console, change to the Permissions tab, click the ACL edit button, and change the permissions to allow Read access for Everyone:



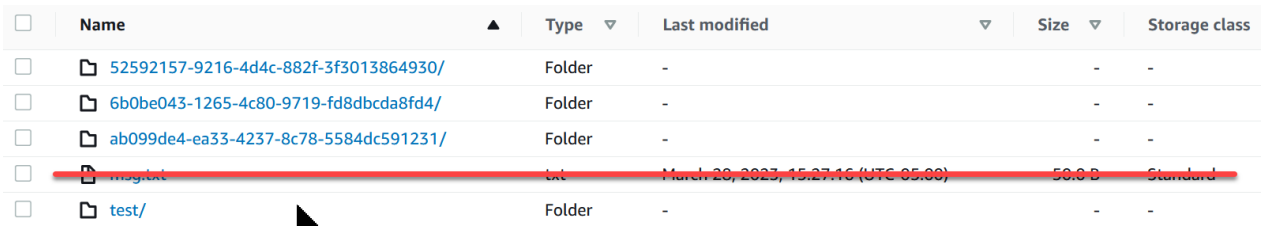
Go ahead and test that everyone has read access to your image(s). Browse to your bucket like before, adding "test/filename.jpg" to the URL:

http://new-bucket-nu-cs310.s3.us-east-2.amazonaws.com/test/degu.jpg

If all is well, your image should appear in the browser. Now we want to repeat this 3 more times… First, create three folders with random, unique names using the UUID python module. Startup any python environment, replit, or an online python compiler such as https://www.online-python.com/, and run the following code each time you need a unique name:

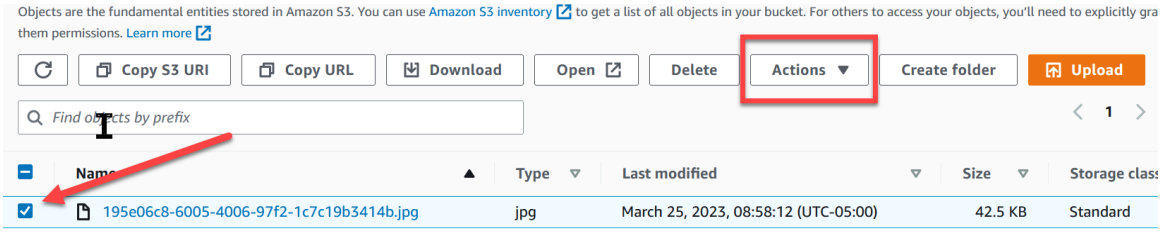
```
import uuid

print(uuid.uuid4())
```

Create 3 folders in your bucket. Something like this:



<< continued on next page >>

Navigate into each folder, and upload at least two image (JPG) files. The filename should also be randomly generated using the UUID module; note you can upload a file with its existing name and then rename the object once it's in the bucket (select the object and then under "Actions" select "Rename object"). Don't forget to also change the permissions on the object so it can be publicly accessed --- a short-cut is to select the object and then under "Actions" select "Make public using ACL".



At this point you should have FOUR folders (including "test"), and inside each folder at least two image (JPG) files. A final test is to browse to your root-level bucket URL, and you should receive an XML document that lists every folder and every object key:

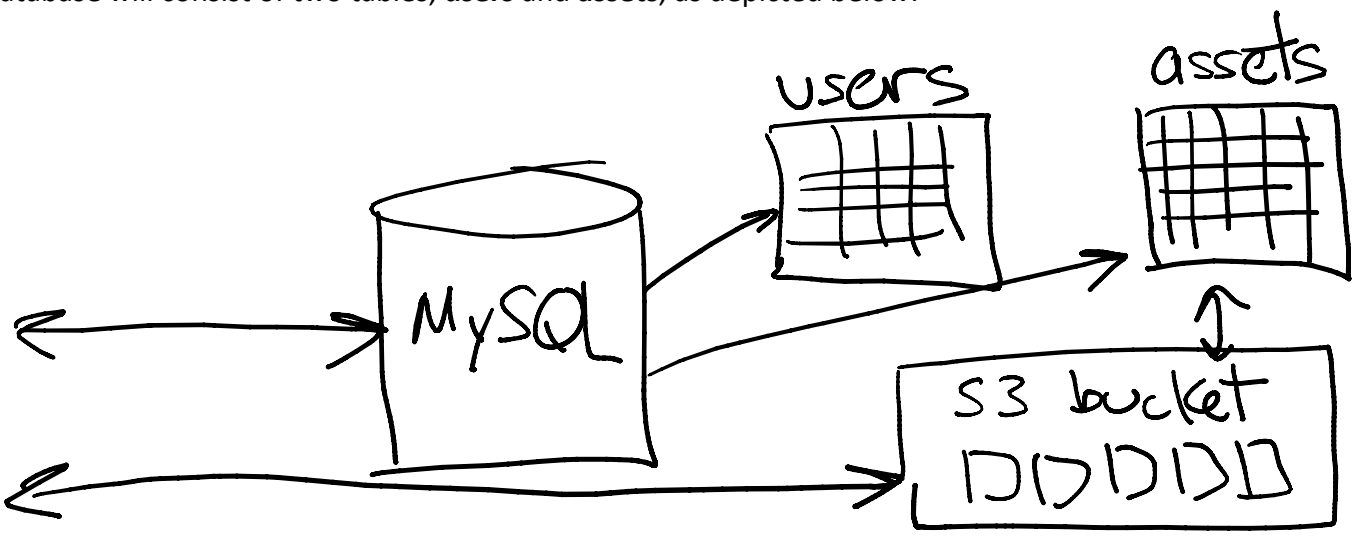
http://**photoapp-nu-cs310**.s3.us-east-2.amazonaws.com/

Something like this:

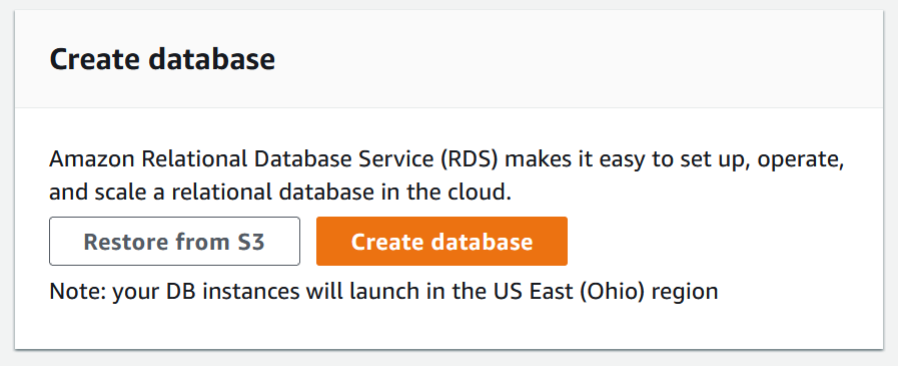

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>photoapp-nu-cs310</Name>
  <Prefix/>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>52592157-9216-4d4c-882f-3f3013864930/</Key>
    <LastModified>2023-03-25T13:37:47.000Z</LastModified>
    <ETag>"d41d8cd98f00b204e9800998ecf8427e"</ETag>
    <Size>0</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <Contents>
    <Key>52592157-9216-4d4c-882f-3f3013864930/195e06c8-6005-4006-97f2-1c7c19b3414b.jpg</Key>
    <LastModified>2023-03-25T13:50:13.000Z</LastModified>
    <ETag>"e9ec7990943728e66fde155981493e84"</ETag>
    <Size>43534</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <Contents>
    <Key>52592157-9216-4d4c-882f-3f3013864930/564055dc-f109-4df8-bdf1-27a629d4a0c6.jpg</Key>
    <LastModified>2023-03-25T13:57:13.000Z</LastModified>
    <ETag>"9cac1fd5cd2f5feadc5f8b38f1735473"</ETag>
    <Size>5438</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <Contents>
    <Key>52592157-9216-4d4c-882f-3f3013864930/a68cab2e-7d23-4af3-bbb1-f067befc6712.jpg</Key>
    <LastModified>2023-03-25T13:56:36.000Z</LastModified>
    <ETag>"e73d5e337b65182f8f9a5f2f68a63804"</ETag>
    <Size>60222</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <Contents>
    <Key>6b0be043-1265-4c80-9719-fd8dbcda8fd4/</Key>
    <LastModified>2023-03-25T13:33:42.000Z</LastModified>
    <ETag>"d41d8cd98f00b204e9800998ecf8427e"</ETag>
    <Size>0</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
```

Step 4: Setup AWS RDS and MySQL database server

Our application needs a database to keep track of users, and for each user to keep track of their images in S3. The database will consist of two tables, **users** and **assets**, as depicted below:

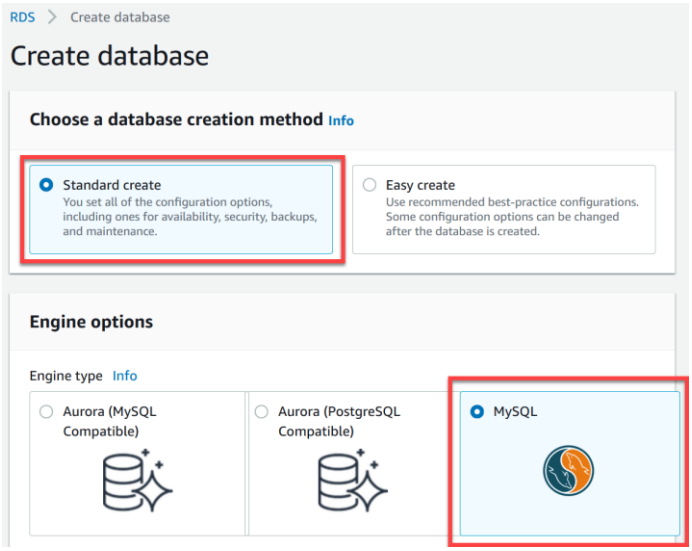


We're going to use **MySQL** as our relational database, which is hosted as part of AWS RDS. The first step is to configure RDS to run MySQL and host our database. Be ready, it's a tedious process. Open the Management Console and search for "RDS". Scroll down to where you see "Create database", and click the big button:

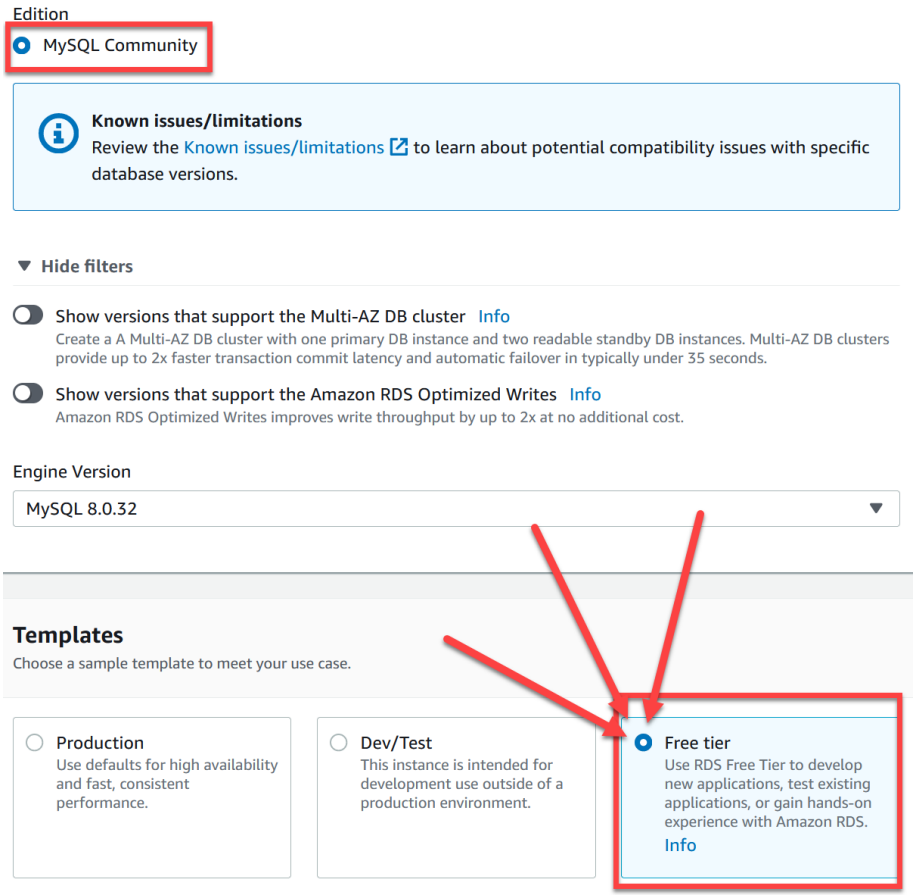


<< next page >>

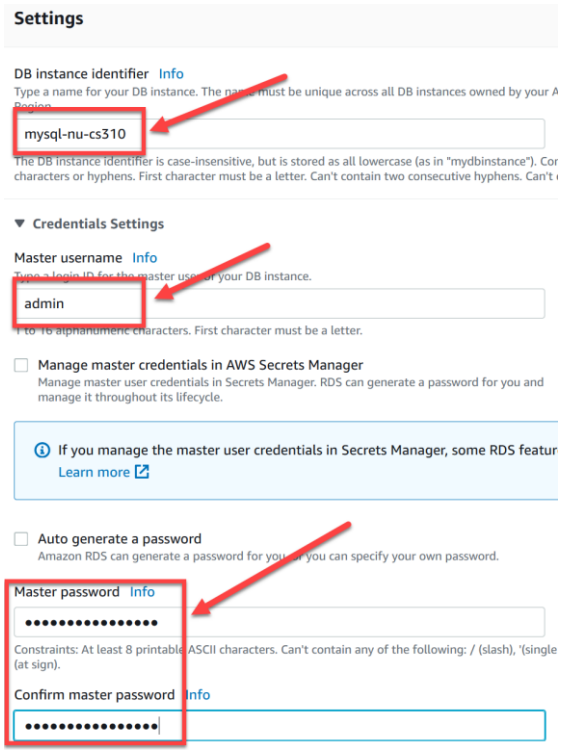
On the top-half of the page select “Standard create” and “MySQL”:



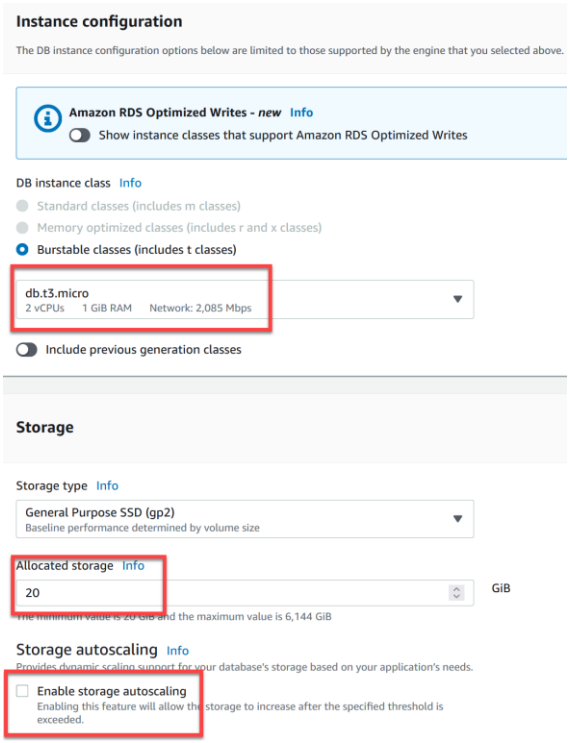
As you scroll down, be sure to select “Free tier”:



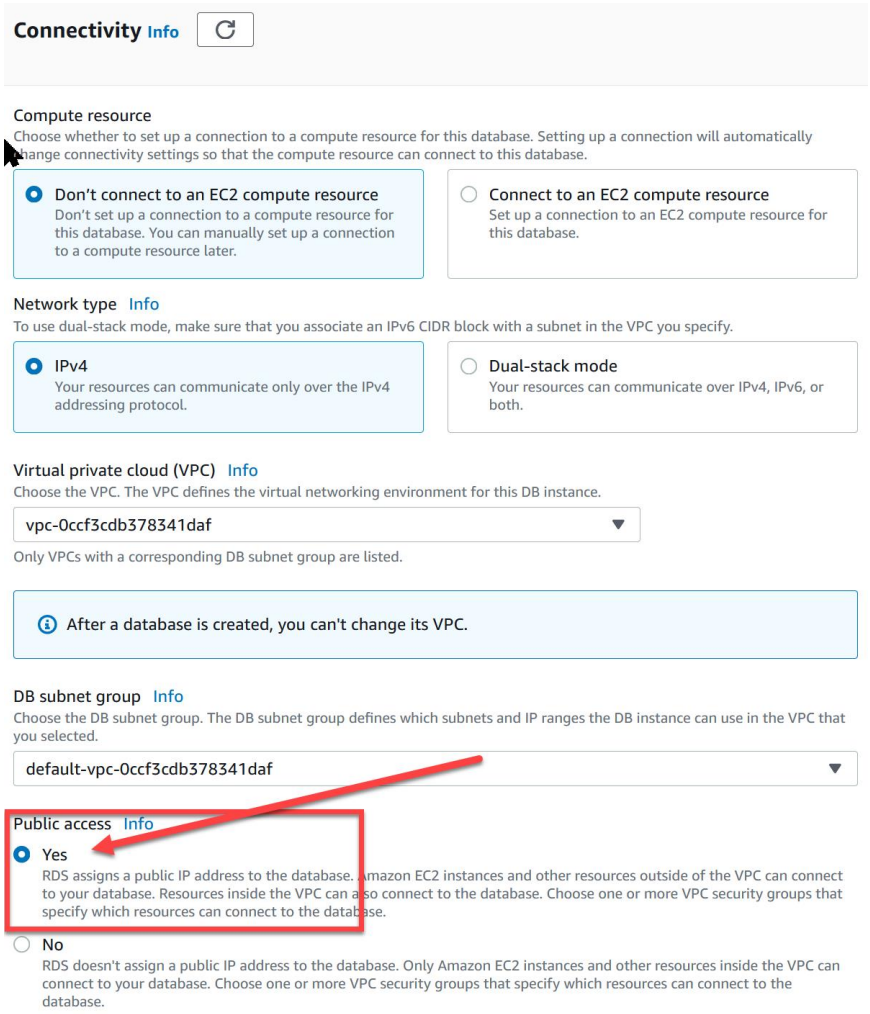
Scrolling down, set the database instance identifier to the name of your MySQL server --- some variation of MySQL is fine --- and provide a password for the server's admin account:



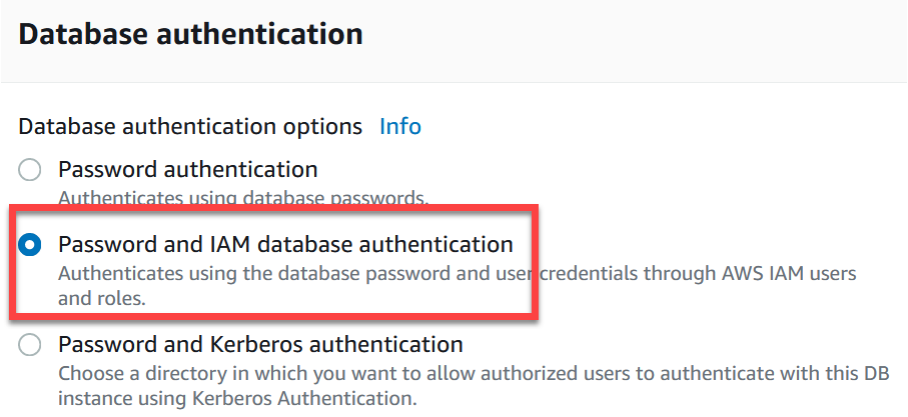
Reduce the allocated storage to the minimum, and disable storage autoscaling:



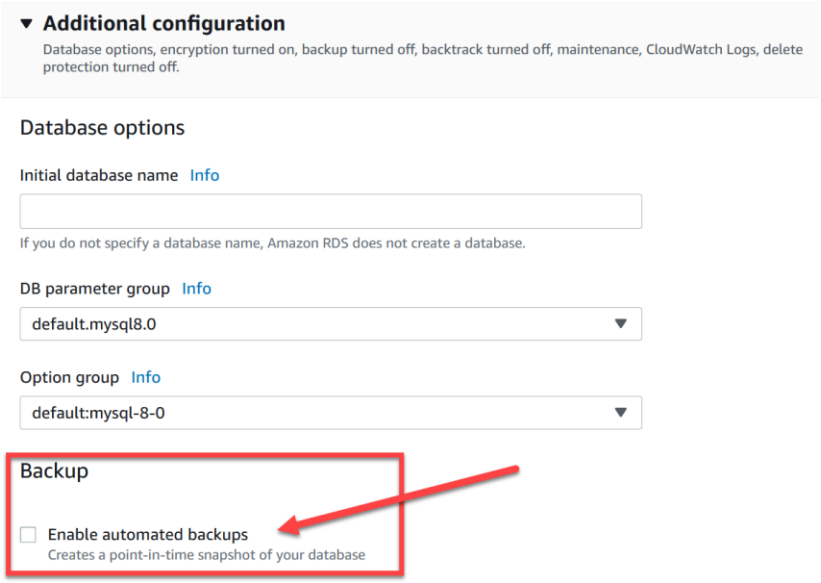
Almost done… Next, enable the database server so it's publicly accessible --- the default values for the other settings should be correct:



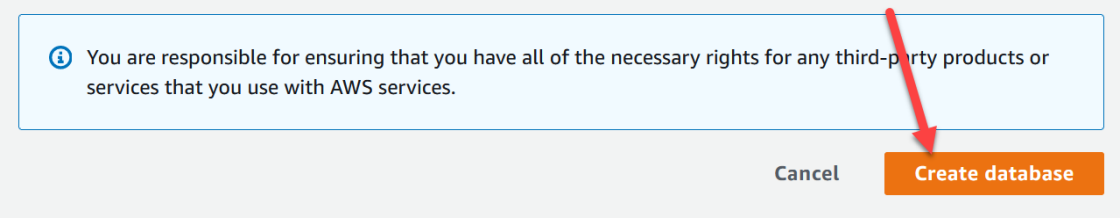
Scrolling down… Change "Database authentication" to allow both "Password and IAM":



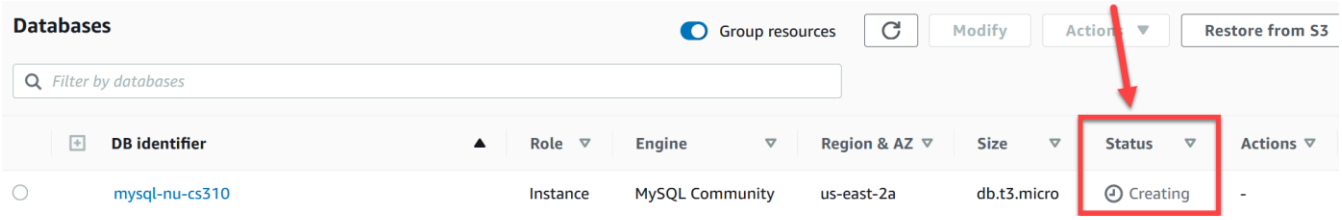
To save $, you want to <u>uncheck</u> automated backups, under "Additional configuration":



Finally, at the very bottom, click "Create database", and take a break. AWS will take at least 5 minutes to create your new database…

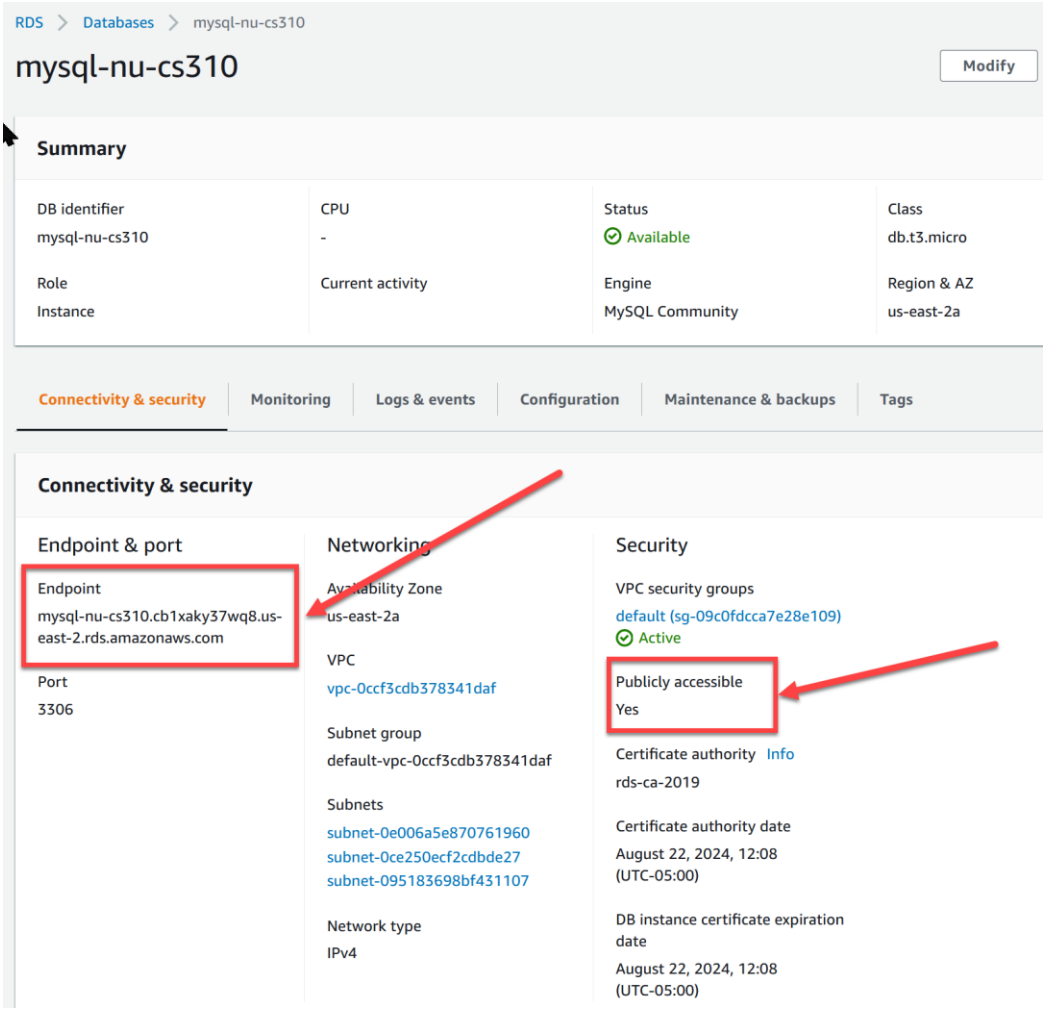


In the Management console, monitor the "Status" column to see when the creation process has completed. You may need to enlarge your browser window (or shrink the page):



<< continued on next page >>

Your database server is ready when its status changes to "Available". When available, click on the database instance ID so you can view its properties, in particular its endpoint and to confirm it's publicly accessible:

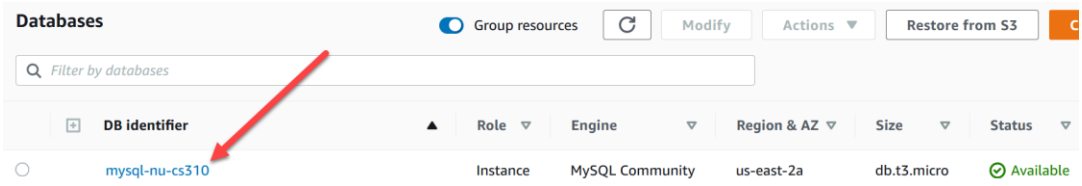


The endpoint is the hostname for your database server --- this is the machine you'll connect to when connecting to your database. If the database is not "Publicly accessible", you will be unable to connect --- in this case delete the database instance and start over…

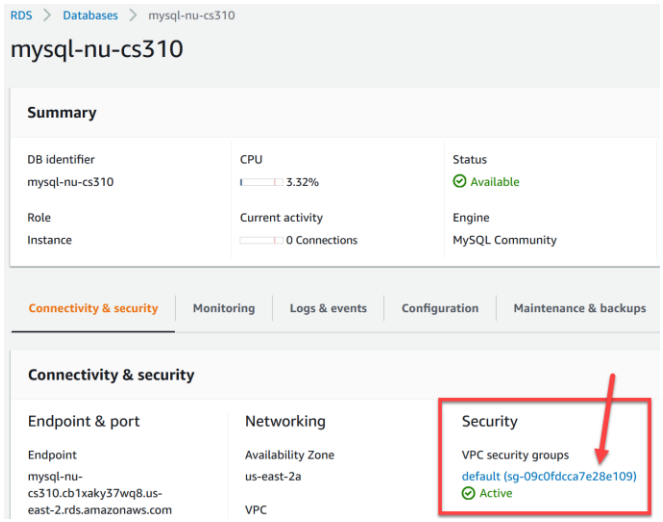
Are we done with the setup of the database server? Unfortunately, no. There's one more step, critically important because it turns out the database server is still not accessible. Read on…

By default AWS is blocking incoming network traffic to the server for security reasons. Normally you could ping the machine to make sure it's up and running --- but ping traffic is blocked. It's our job to decide how much to "open up" our database server to the internet. For simplicity, we're going to open the server to the entire world so it can be accessed by any machine.

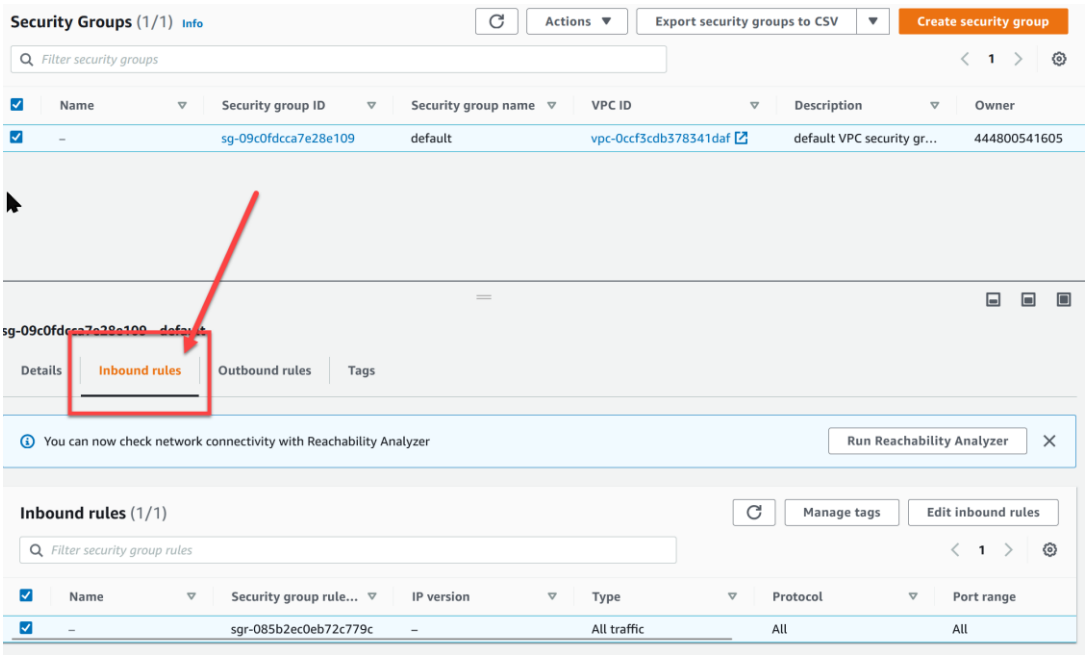
First, confirm your database is available (search for “RDS”, and click “databases” along the left-side):



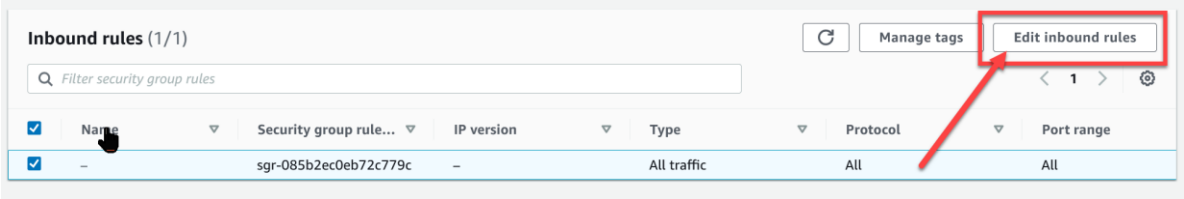
Click on the database server identifier, and you’ll see it’s properies. Focus on Security:



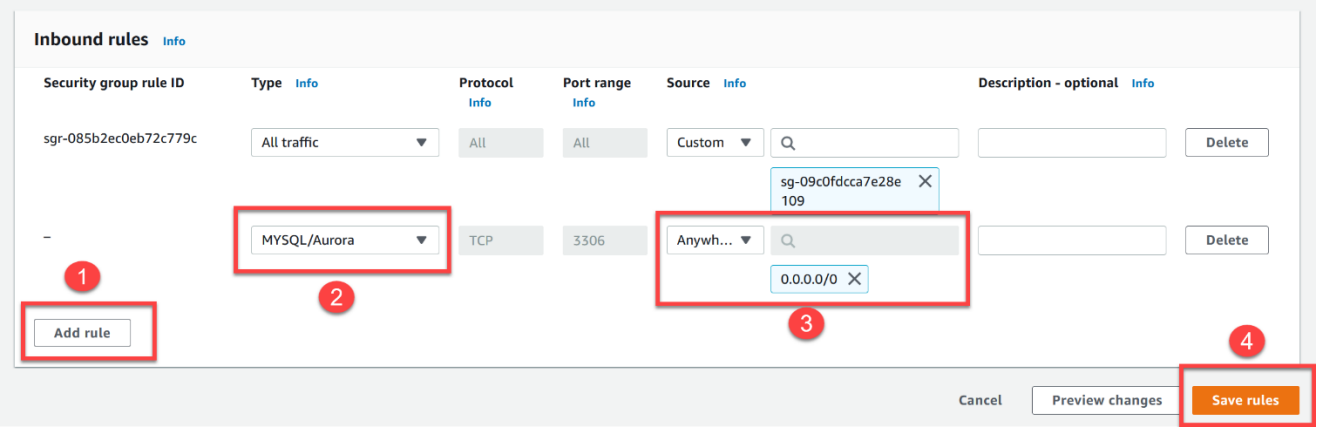
Click on the VPC (Virtual Private Cloud) security group, and click the tab for “Inbound rules”:



Interestingly, even though it lists "All traffic", it's not what we think. Click "Edit inbound rules" so we can add another rule to open up the server to all IP addresses:



In the dialog that opens, click "Add rule", change Type to "MySQL/Aurora", and change Source to "Anywhere-IPv4". Then click "Save Rules":



Done with RDS and MySQL server setup!

The good news is you only have to do this once, the AWS RDS MySQL database server can host any number of databases. To confirm everything is working properly, the simplest approach is to login to replit.com, open the team, and then open the project "Test-MySQL-in-AWS". It's a small python program that will connect to your database server, and if successful, output the current date and time GMT. You'll need to copy-paste your database server endpoint and master user password (and master user name if you changed it from "admin"):

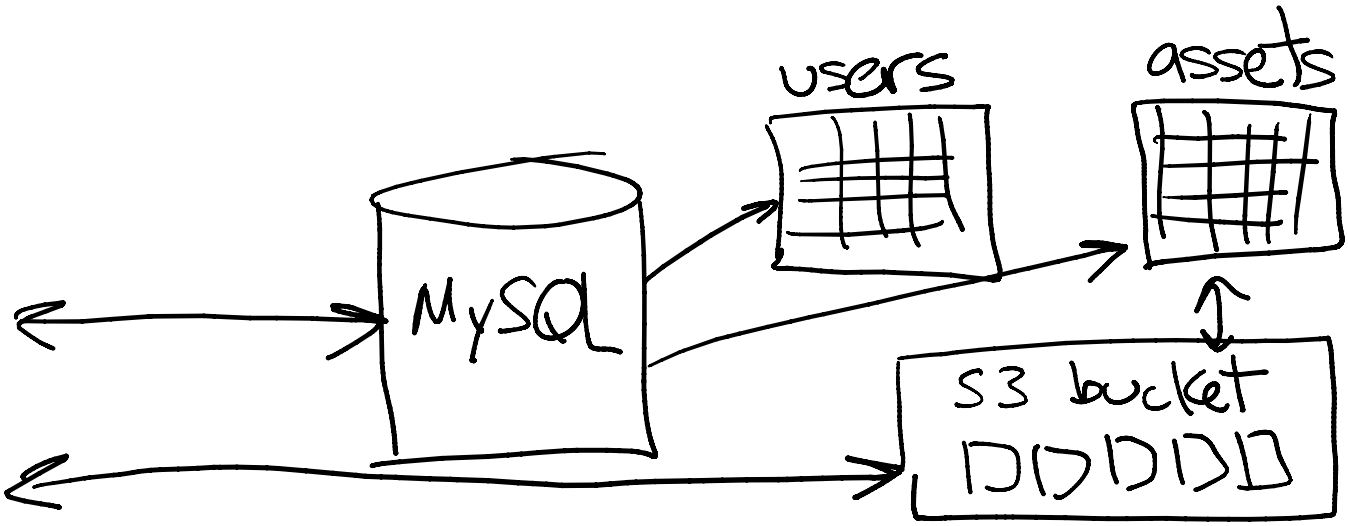
```
ENDPOINT="TODO"
USER="admin"
PASSWORD="TODO"
```

Click "Run" and if all is well, you'll see date and time. If you plan to take a break from working on the assignment, select your database instance and under "Actions", select "Stop temporarily" --- this will pause the server, free the assigned VM instance, and save $. However, you'll need to manually restart when you want to resume working. Using AWS CLI, where the DB_INSTANCE_NAME is something like *mysql-nu-cs310*:

```
aws rds stop-db-instance --db-instance-identifier DB_INSTANCE_NAME
…
aws rds start-db-instance --db-instance-identifier DB_INSTANCE_NAME
```

Step 5: Create photoapp database

Now that MySQL is setup in AWS RDS, the next step is to create our photoapp database with two tables **users** and **assets**, to keep track of users, and for each user to keep track of their images in S3:



The **users** table will keep track of each user's email, last name, first name, and the name of their bucket folder. Every user is identified by a unique user id; ids will start at 80001 and increase by 1. In DB design terms, userid is the *primary key*. Example: the CS310 photoapp database consists of 3 rows in the users table:

userid	email	lastname	firstname	bucketfolder
80001	pooja.sarkar@company.com	sarkar	pooja	6b0be043-1265-4c80-9719-fd8dbcda8fd4
80002	e_ricci@email.com	ricci	emanuele	ab099de4-ea33-4237-8c78-5584dc591231
80003	li_chen@domain.com	chen	li	52592157-9216-4d4c-882f-3f3013864930

The **assets** table keeps track of each image in the bucket: it's original name ("assetname"), it's key name ("bucketkey"), and the user who's to which this image belongs to ("userid"). Each asset has a unique asset id; ids start at 1001 and increase by 1. In DB design terms, assetid is the *primary key*, and userid is a *foreign key* to the users table. Example: the CS310 photoapp database contains 3 images per user, for a total of 9 rows in the assets table:

assetid	userid	assetname	bucketkey
1001	80001	A3-mac-2016.JPG	6b0be043-1265-4c80-9719-fd8dbcda8fd4/af986381-55ac-4bf2-85b3-ff4a29047226.jpg
1002	80001	A3-verve-Megan-on-bow.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/62ec70d5-ba0c-4822-9017-3864b1d1fb47.jpg
1003	80001	NOOD-A3-water-photo-2016.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/2c20b346-6f2b-4bca-b770-772bd0aa9ad3.jpg
1004	80002	ajax.jpg	ab099de4-ea33-4237-8c78-5584dc591231/a4365e93-5c41-4997-8414-389cf874509e.jpg
1005	80002	Darwin.jpg	ab099de4-ea33-4237-8c78-5584dc591231/fdcc04a8-4a73-494a-978f-41e0dba9eeb3.jpg
1006	80002	Degu.JPG	ab099de4-ea33-4237-8c78-5584dc591231/58732c6b-2e1c-4503-944b-c63f7abb546c.jpg
1007	80003	All-Day-Joke.jpg	52592157-9216-4d4c-882f-3f3013864930/a68cab2e-7d23-4af3-bbb1-f067befc6712.jpg
1008	80003	NU-purple-background.png	52592157-9216-4d4c-882f-3f3013864930/564055dc-f109-4df8-bdf1-27a629d4a0c6.jpg
1009	80003	NU-white-background.jpg	52592157-9216-4d4c-882f-3f3013864930/195e06c8-6005-4006-97f2-1c7c19b3414b.jpg

NOTE: because MySQL is multi-platform and potentially case-sensitive, the convention is to use lowercase for all table and column names. We follow that convention in our database design.

The database and tables are created by executing SQL. Unfortunately, there's no tool within the AWS Management Console for executing the necessary SQL to create the database. You'll need to install a local tool on your laptop to interact with the MySQL instance running in AWS. I would recommend either of the following:

1. **MySQL Workbench**: free, runs on any platform, very popular. Download link is https://dev.mysql.com/downloads/workbench/ . [M1 Monterey users; M1 Big Sur users.]
2. **Visual Studio code**: if you're already using VS Code, there are a number of extensions you can install to connect to MySQL servers and execute SQL. *MySQL Management Tool* (Jun Han) works fine; there are many others.

You'll see demos of how to use these tools in class, but the general idea is to open a connection to your MySQL database instance, and then open a query window and execute the SQL. To open a connection, you'll need to provide your endpoint (e.g. mysql-nu-cs310.cb1xaky37wq8.us-east-2.rds.amazonaws.com), master user name (e.g. admin), and master user password. If you're asked for the port number, use 3306; if asked for a security certificate, it's optional so you can ignore. Once connected, you want to open a query window and execute SQL. As a test, execute the following SQL query to display the current date and time in GMT:

```
select now();
```

You're ready to create to the photoapp database. Execute the following SQL:

```
CREATE DATABASE photoapp;
```

Now we need to create the tables. Here's the SQL, but it's also available online in the file create-tables.sql so you can more easily copy-paste:

```
USE photoapp;

DROP TABLE IF EXISTS assets;
DROP TABLE IF EXISTS users;

CREATE TABLE users
(
    userid       int not null AUTO_INCREMENT,
    email        varchar(128) not null,
    lastname     varchar(64) not null,
    firstname    varchar(64) not null,
    bucketfolder varchar(48) not null,  -- random, unique name (UUID)
    PRIMARY KEY (userid),
    UNIQUE      (email),
    UNIQUE      (bucketfolder)
);

ALTER TABLE users AUTO_INCREMENT = 80001;  -- starting value

CREATE TABLE assets
(
    assetid      int not null AUTO_INCREMENT,
    userid       int not null,
```

```
    assetname    varchar(128) not null,  -- original name from user
    bucketkey    varchar(128) not null,  -- random, unique name in bucket
    PRIMARY KEY (assetid),
    FOREIGN KEY (userid) REFERENCES users(userid),
    UNIQUE      (bucketkey)
);

ALTER TABLE assets AUTO_INCREMENT = 1001;  -- starting value
```

The last step is to execute SQL to populate your database with M users (one user row for each folder you created), and N assets (one asset row for each image you uploaded). Given the earlier instructions, you should have at least 3 users (M >= 3) and 6 assets (N >= 6) in your database. How do you insert row into the database? Use an SQL INSERT action query. Here are examples of inserting into the users and assets table from our CS310 database (this SQL is available in insert-rows.sql). Note that the userid in the users table (and the assetid in the assets table) are automatically generated to ensure they are unique, and so those values are omitted when inserting into the users table (and assets table):

```
USE photoapp;

INSERT INTO
  users(email, lastname, firstname, bucketfolder)
  values('pooja.sarkar@company.com', 'sarkar', 'pooja',
         '6b0be043-1265-4c80-9719-fd8dbcda8fd4');

INSERT INTO
  assets(userid, assetname, bucketkey)
  values(80001,
         'A3-mac-2016.JPG',
         '6b0be043-1265-4c80-9719-fd8dbcda8fd4/af986381-55ac-4bf2-85b3-ff4a29047226.jpg');
```

Create fictitious user names and emails, then associate each user with one of the folders in your S3 bucket. Likewise, associate the image within each bucket with the appropriate user (based on which user owns which folder). Example: if we insert “pooja” first, she’ll have the userid 80001. We associate her with one of the folders, and then associate the image(s) in that folder with her. If this is the first asset inserted, it will have assetid 1001. Repeating this for each folder and each image in the S3 bucket, we end up with the tables populated as shown earlier.

As you insert and populate your database, you can check your work using SELECT * as shown:

```
USE photoapp;

SELECT * FROM users;
```

userid	email	lastname	firstname	bucketfolder
80001	pooja.sarkar@company.com	sarkar	pooja	6b0be043-1265-4c80-9719-fd8dbcda8fd4
80002	e_ricci@email.com	ricci	emanuele	ab099de4-ea33-4237-8c78-5584dc591231
80003	li_chen@domain.com	chen	li	52592157-9216-4d4c-882f-3f3013864930

```
USE photoapp;

SELECT * FROM assets;
```

assetid	userid	assetname	bucketkey
1001	80001	A3-mac-2016.JPG	6b0be043-1265-4c80-9719-fd8dbcda8fd4/af986381-55ac-4bf2-85b3-ff4a29047226.jpg
1002	80001	A3-verve-Megan-on-bow.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/62ec70d5-ba0c-4822-9017-3864b1d1fb47.jpg
1003	80001	NOOD-A3-water-photo-2016.jpg	6b0be043-1265-4c80-9719-fd8dbcda8fd4/2c20b346-6f2b-4bca-b770-772bd0aa9ad3.jpg
1004	80002	ajax.jpg	ab099de4-ea33-4237-8c78-5584dc591231/a4365e93-5c41-4997-8414-389cf874509e.jpg
1005	80002	Darwin.jpg	ab099de4-ea33-4237-8c78-5584dc591231/fdcc04a8-4a73-494a-978f-41e0dba9eeb3.jpg
1006	80002	Degu.JPG	ab099de4-ea33-4237-8c78-5584dc591231/58732c6b-2e1c-4503-944b-c63f7abb546c.jpg
1007	80003	All-Day-Joke.jpg	52592157-9216-4d4c-882f-3f3013864930/a68cab2e-7d23-4af3-bbb1-f067befc6712.jpg
1008	80003	NU-purple-background.png	52592157-9216-4d4c-882f-3f3013864930/564055dc-f109-4df8-bdf1-27a629d4a0c6.jpg
1009	80003	NU-white-background.jpg	52592157-9216-4d4c-882f-3f3013864930/195e06c8-6005-4006-97f2-1c7c19b3414b.jpg

If you make a mistake, not to worry. Simply re-run the first SQL code and rebuild the database (create-tables.sql); this deletes and recreates an empty database. Now re-insert the data, correcting any errors you made earlier. You can repeat this as many times as necessary.

Step 6: Controlling access to your database

Right now you are connecting to your MySQL server using the adminstrator account. This is okay when creating a database, but certainly not a best practice for general-purpose read/write access. When we start programming against the database using Python, we want user accounts with more limited permissions.

User accounts and permissions are defined in two ways: (1) through MySQL, and (2) through AWS IAM. We're going to use both methods, but to start we're going to focus on #1 so you see how to create accounts using MySQL --- this way you'll know how to do this in the future if you happen to be working with MySQL outside of AWS. Here's the SQL to create two user accounts, photoapp-read-only and photoapp-read-write. The former can execute SELECT queries to retrieve data, and the latter can execute both SELECT and ACTION queries to both retrieve and modify data (this is available online in the file add-users.sql). Do *not* change the user names as shown, and do *not* change the passwords, we need these as below for grading purposes:

```
USE photoapp;

DROP USER IF EXISTS 'photoapp-read-only';
DROP USER IF EXISTS 'photoapp-read-write';

CREATE USER 'photoapp-read-only' IDENTIFIED BY 'abc123!!';
CREATE USER 'photoapp-read-write' IDENTIFIED BY 'def456!!';

GRANT SELECT, SHOW VIEW ON photoapp.*
      TO 'photoapp-read-only';
GRANT SELECT, SHOW VIEW, INSERT, UPDATE, DELETE ON photoapp.*
      TO 'photoapp-read-write';

FLUSH PRIVILEGES;
```

Test and make sure the user accounts were created successfully. The best way to test is to create another connection to your photoapp database --- using MySQL Workbench or VS Code (or whatever tool you are using to execute SQL) --- and connect using the user name **photoapp-read-only** and the password **abc123!!**. You should be able to successfully connect, and execute this SQL:

```
USE photoapp;

SELECT * FROM users;
SELECT * FROM assets;
```

Repeat, creating another connection with the username **photoapp-read-write** and the password **def456!!**. Congratulations, and well done!

Step 7: Pause your RDS server?

At this point you may want to take a break, and in that case you should temporarily stop your database server instance to save $. Using the AWS Management Console, select your database instance and under "Actions", select "Stop temporarily" --- this will pause the server, free the assigned VM instance, and save $. However, you'll need to manually restart when you want to resume working. Alternatively, you can use the AWS CLI, where the DB_INSTANCE_NAME is something like *mysql-nu-cs310*:

```
aws rds stop-db-instance --db-instance-identifier DB_INSTANCE_NAME
…
aws rds start-db-instance --db-instance-identifier DB_INSTANCE_NAME
```

Next steps?

This is NOT the end of Project 01. But the handout is 25 pages at this point, so I'm going to break the handout into parts 01 and 02. This is the end of part 01, but there's a part 02 where we'll program in Python to tie all this together and build the foundation of our PhotoApp application. Watch for the release of part 02 on Canvas and Piazza. The due date remains the same: Friday October 6th, 11:59pm. Final submission details will appear in part 02.