NWEN 243 – 2023 T2 Project 2

Creating a server on AWS: Music Guru

Due:

- see submission system for authoritative dates and times.

Value 15% of your final Grade.

- You will need to submit screen shots and photographs of your working services - the **minimum** places are noted in RED in this document. If you only partially complete a section, choose appropriate screenshot to show how far you got. **Additional screenshots will always help.**
- You should provide contextual commentary on all screenshots you submit.

When people talk about "Servers" they mean two different things. They can mean:

1. the software that responds to client requests - like a WebSever, responds to requests from your browser (Client) for pages, or
2. A machine that hosts services (like in 1)

People use these terms interchangeably, and it can be a bit confusing. In this lab, I will use server to mean definition 1. For definition 2. I have so far used the term "instance" but that is a cloud term - that isn't always accurate elsewhere

Lab Outline

- Build on EC2 instances from lab 1
- Build a client / server pair and host your application - which we're calling 'Music Guru'.
- Client/Server will be the basis for learning about distribution and containers. So you really need to make sure you get the basic software working for both this and later labs. Time investment here will make all the difference.

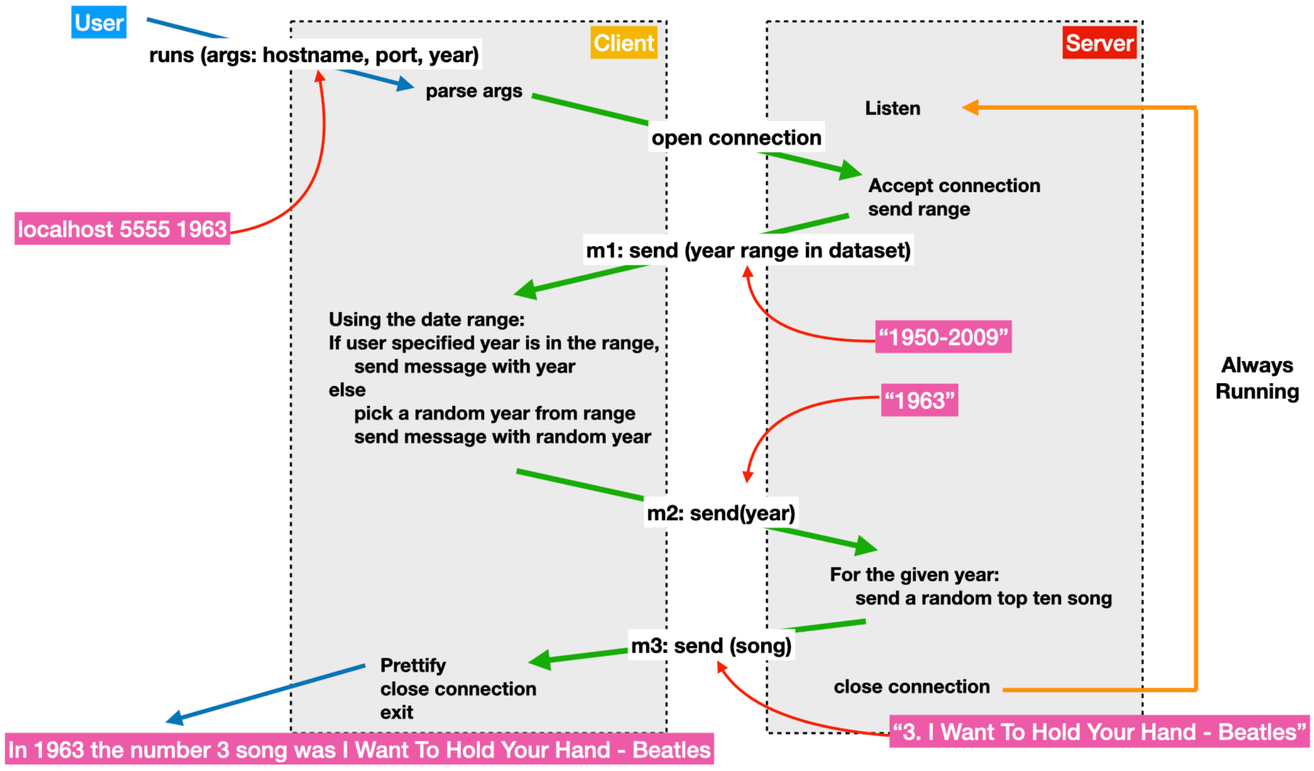
Part I: The “Music Guru” Service

The “Music Guru” is a service that knows all about the top 10 hits in a particular year.

About the data: you can use any source of chart data and music genre you like, you can also use any non zero date range you like - the choice of dataset is not assessed. I simply grabbed a general list from a random website for the dates 1950-2010. You can either store this information in a file, that your code manipulates, or hard code it into your program.

1. Code Specification:

- You will need to write **both** client and a server parts of “Music Guru”.
- You have the option of writing your client server in Python[1] **or** Java.
- You will use TCP and IPv4.
- Library choices in **python and java** are limited to the standard **socket** and **socket class** respectively, but you may use any other libraries you choose for string manipulation etc.
- Your resulting code for your server should work with anyone else’s client and vice versa.
- You should ensure your code is reasonably robust and does error handling.
- This is the specification, with examples, for your code:


User
runs (args: hostname, port, year)
Client
Server
parse args
Listen
open connection
Accept connection
send range
localhost 5555 1963
m1: send (year range in dataset)
“1950-2009”
Using the date range:
If user specified year is in the range,
send message with year
else
pick a random year from range
send message with random year
“1963”
Always Running
m2: send(year)
For the given year:
send a random top ten song
m3: send (song)
Prettify
close connection
exit
close connection
“3. I Want To Hold Your Hand - Beatles”
In 1963 the number 3 song was I Want To Hold Your Hand - Beatles

- The server is always running (in the background) and can handle many clients in a row.
- The client only runs once, and if the user wants another song, they need to run the client again.
- **Note:** the server sends the date range in response to the connection, you don't need to have the client explicitly ask for it, but the client should immediately read it from the connection as a first step.
- You would usually write your client and server together, incrementally testing as you go. However, see section 2, for an alternative approach.

VERY IMPORTANT

If you use a tutorial, AI tool or someone else's code published online as a basis for your solution, YOU MUST reference it in your comments! You will not get penalised for correct attribution, indeed – you might well get penalised for not attributing, as that is plagiarism.

2. My Music Guru - Reference example.

- I will run my Music Guru - IP (to be confirmed on the project page) and port 5000 on an ECS server and not on AWS due to the - 4 hour time limit.
- Warning: you cannot connect to port 5000 from outside ECS on our machines. If you are not in a lab, you will need to ssh into, say, embassy and compile and run it there.
- As I am providing a reference running server - you could write the client first to check you are conforming to the required message and functional standard. Then write your version of the server to match your 'validated' client.
- Once you are done writing your client and server and testing them on the localhost - you can start to think about running your service on the AWS cloud.
- **Note**: The nice thing about TCP and sockets, is that it doesn't matter where the server is, what the language is, or whose machine it is running on (as long as it can get through the firewall - hey, guess what your AWS security group does!)

3. Writing your Music Guru Client-Server programs

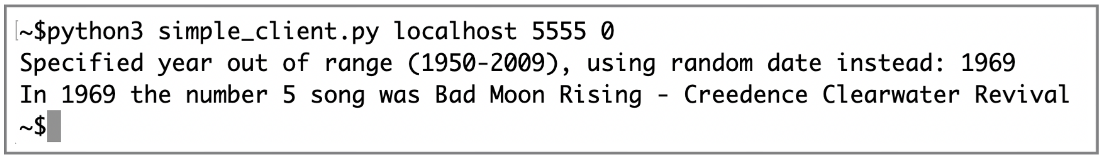
You should do your development on your local machine in your preferred IDE as is good practice - with local testing before deployment to the cloud. By local testing, I mean run your client and server on the same machine, and using localhost as the IP address. This means we can avoid errors introduced by networks when debugging, however, be sure you don't hardcode things like IP address / name or port number.

MusicGuruServer: creates a TCP 'server' socket (you must choose a port number, I usually use 5000), pass the port number in as a command line argument.

MusicGuruClient: opens a TCP socket and connects to the server. The server name or IP, port and year arguments are given by the user as command line arguments.

- The server will loop forever around an **accept** method call, which will pause the server until a connection request is made from a client. Once the request is accepted, send the year range of your dataset to the client via the socket. Do not hardcode the range in the client.
- The client need not send a request for the range, your server will need to send it automatically as the result of accepting a TCP connection.
- The client will now choose a year from the range and send that to the server. The way mine is set up, is I specify a year in the client command line argument. If it turns out this was within the range the server sent, I request that year. Otherwise my client generates a random year from the range. So for instance you can give the year as 0, and you'll just get a random top ten song from a random year - it is quite fun using it like that.
- When the server gets the year from the client:
    - it will then pick at random one of the top ten songs from that year and write it back into the socket - sending it to the client.
    - Something students have done incorrectly in past years (different project though) is to print the response on the server. To be extra clear, the server must write the song out to the client using the socket.
    - The server then closes the client connection and loops back around the moment you have sent the song - ready for the next new client connection.
- When the client gets the song from the server:
    - Make it pretty as per the example diagram.
    - Print it out.
    - close the socket

- My Music Guru client output looks like this:



```
~$python3 simple_client.py localhost 5555 0
Specified year out of range (1950-2009), using random date instead: 1969
In 1969 the number 5 song was Bad Moon Rising - Creedence Clearwater Revival
~$
```

- A minimal functional implementation is just fine. You have complete leeway on how to write this client-server pair, as long as you conform to the expected messaging and formatting between client and server.

- **Note**: These are not long programs - in their most basic forms. Of course, you can always write more, parse things beautifully, store or manipulate song data files with pizazz. However, if you find yourself writing endless code for the socket side of things - please ask your tutor as using sockets is almost the same as using files and you may have gotten off track a bit.

Evidence to Submit: take a screen shot, with two terminal windows open on your local machine, one showing the execution of the Music Guru server, and the other showing the client. SAVE IT AS: 1.jpg

4. Create an EC2 Instance for your “Music Guru”:

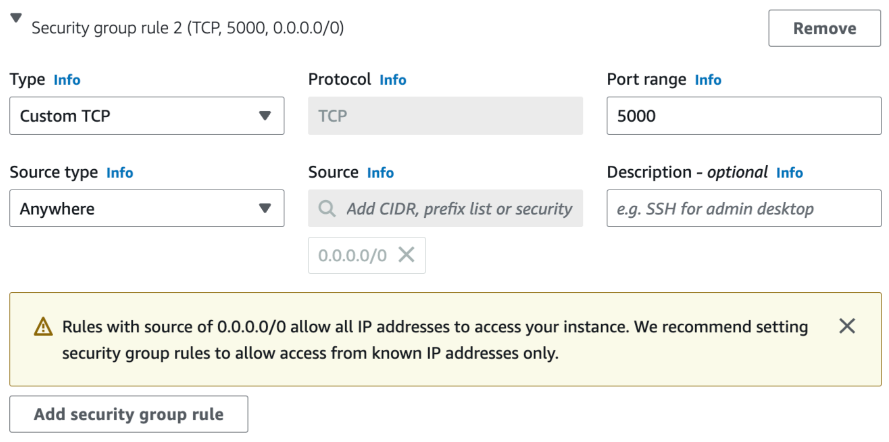
- You will need to create and configure an EC2 instance for hosting your Music Guru service. You will need:

    - a new EC2 instance - you must use **Amazon Linux 2 AMI (HVM)** free tier.

    - private key pair, remember to:

        chmod 400 myKeys.pem

    - Under “network settings” You will need to add a custom TCP security rule. Select “add security group rule” - which is at the bottom of the section. You



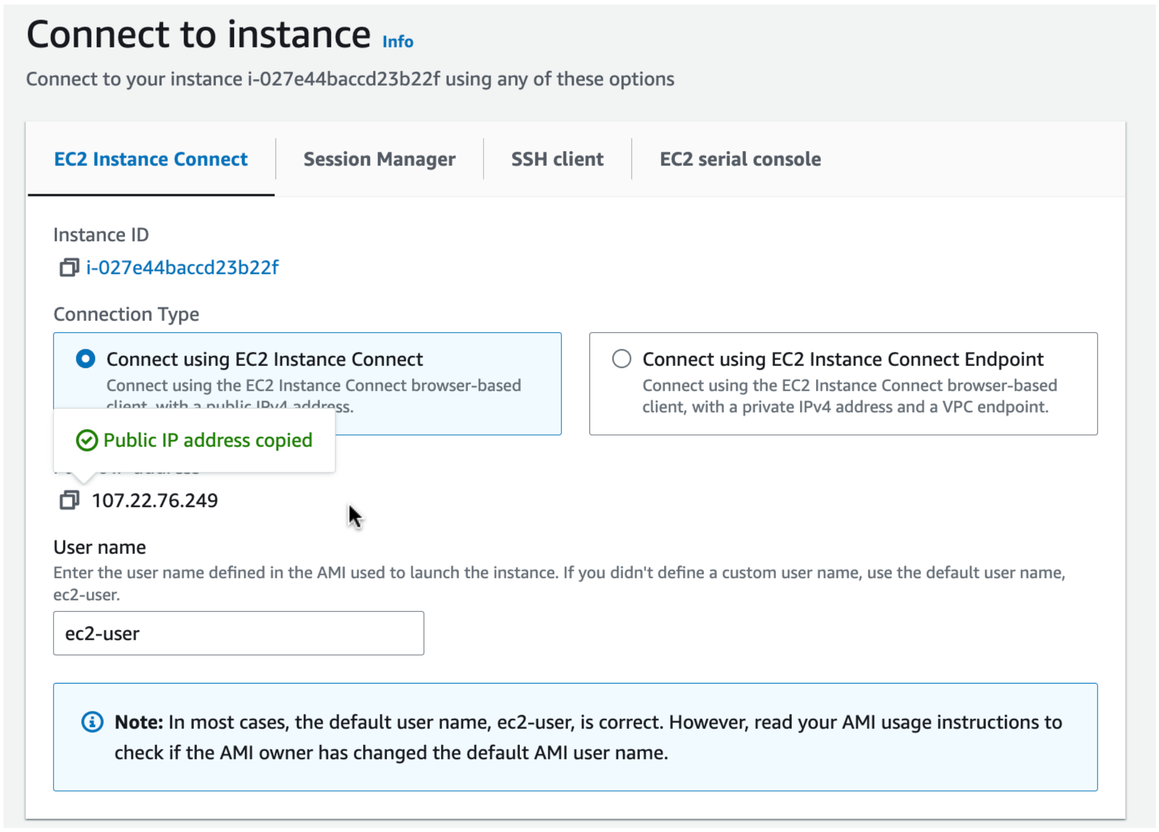
Security group rule 2 (TCP, 5000, 0.0.0.0/0)

Type: Custom TCP
Protocol: TCP
Port range: 5000
Source type: Anywhere
Source: 0.0.0.0/0
Description - optional: e.g. SSH for admin desktop

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

will need to make a customTCP rule, with a port range that will be the port number you are using (mine is 5000) and ensure source is set to everywhere (0.0.0.0/0). **Don't delete or overwrite the default ssh rule, and you do not need a HTTP rule for this instance. You can ignore the nanny warning…**

- You can now launch your instance. You can find the IP address of your instance in various places, after launch. Including from the screen you saw in project 1.



Connect to instance Info

Connect to your instance i-027e44baccd23b22f using any of these options

EC2 Instance Connect | Session Manager | SSH client | EC2 serial console

Instance ID
i-027e44baccd23b22f

Connection Type
[✓ Connect using EC2 Instance Connect]
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

Connect using EC2 Instance Connect Endpoint
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IP address copied

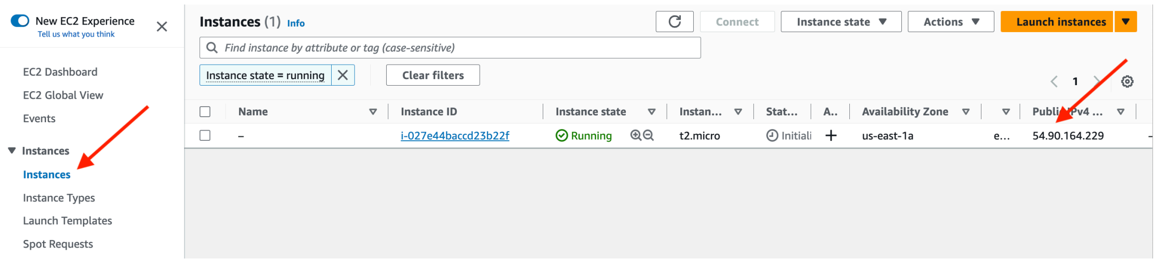
107.22.76.249

User name
Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, ec2-user.

ec2-user

Note: In most cases, the default user name, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

- Or from the dashboard instances panel.



Instances (1) Info

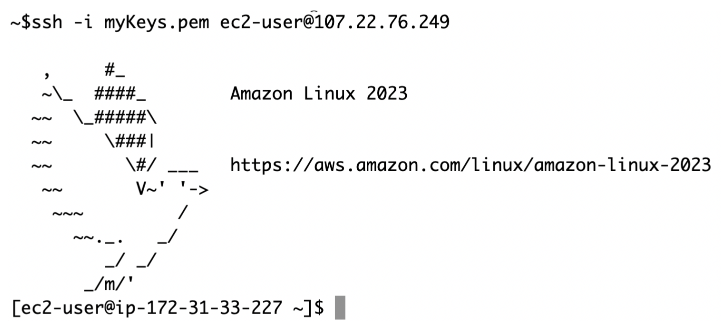
Find instance by attribute or tag (case-sensitive)

Instance state = running

	Name	Instance ID	Instance state	Instan…	Stat…	A..	Availability Zone		Publi IPv4 …
	–	i-027e44baccd23b22f	Running	t2.micro	Initiali		us-east-1a	e…	54.90.164.229

5. Connect to your EC2 Instance via ssh.

- We're going to do things a little differently from lab 1 - you'll see why later in Project 3.
- Use SSH on your local machine and the key-pairs and IP address to connect to your new ec2 instance.
    - ssh -i myKeys.pem ec2-user@X.X.X.X



```
~$ssh -i myKeys.pem ec2-user@107.22.76.249

   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-33-227 ~]$ 
```

6. Configure your Instance:

Depending on your language of choice, you will need to either configure Java on your ec2 instance, or sit back and **relax** if you chose to write in python3 as it is already installed.

6.1 JAVA

Java is not installed by default on your instance. So we need to do that now. You can install either java 11 or java 8. Ensure the version is compatible with your local development machine.

You'll likely be told some packages are out of date - do the sudo yum update it requests. Then (as sudo - Super User Do)…

- java 8 or java 11
    - sudo yum install java-1.8.0-openjdk OR
    - sudo amazon-linux-extras install java-openjdk11
- Check it all installed ok and the version is right with
    - java -version

- If you find yourself needing to switch version, try:
    - sudo alternatives --config java

  note, this is a runtime not a full developer suite, so you cannot compile your java with javac on your instance. This is actually the right way to do things, you should develop on your local machine, using whatever development environment you like, and then upload the JAR or class files as needed to the EC2 instance.

6.2 UPLOADING YOUR CODE

To upload your code to your instance, you will need to use scp to copy your code from your development machine to the EC2 instance.

I recommend you upload a small test program first - in your language of chocie. Like HelloWorld. Just so you can make sure the install etc went well.

- scp -i <your keys>.pem <your files> ec2-user@X.X.X.X**:**
- Don't forget the ':' at the end - it is critical.
- Also don't forget any song data file you might have
- Now do an ' ls' in your EC2 instance, and you will see your uploaded file.
- Run your program, it will work if everything is correct.

7. Run your Music Guru on your EC2 Instance:

- Now run your MusicGuru Server on the instance and connect to it using the client running on your development machine (NOT on the instance). Make sure you use the public IP address not the private one(s).
- **Make sure** your client, server and security rule all use the same port number. I can't emphasise strongly enough how many student hours are lost to this simple mistake!
- If everything worked - you now have a running AWS service that you wrote!

Evidence to Submit: a screen shot, with two terminals open, one showing the execution of the Music Guru server on the EC2 instance, and the other showing the client window on your dev/local machine, ensure the IP and ports are clear in both. SAVE IT AS: 2.jpg

8. Preparing to 'scale out' your Music Guru

At this point you have your service running on an AWS host. But there were a lot of manual steps involved in the creation of the instance, configuration, uploading your code, and even running the server itself. That is not very convenient - and worse, it is impossible to build a scaleable application like that, which after-all, is what the cloud does best.

What we want to do is make a 'template' of our running service, so we can run as many copies as we need, at any particular time. After all, Music Guru, might be the next AirBnB or Netflix!

8.1 MODIFY YOUR CODE

Our Music Guru is a complete solution. However, we're going to need to make a small modification for testing and assessment purposes. We're going to potentially have many copies running at the same time, we'll need to also print out which server a response came from. Normally the user would not see this.

Modify your Music Guru server code to return its IP address as a post-fix (at the end) when you return a song. Depending on how we manage multiple servers, the client may not even be aware of which specific server it is talking to, and that is the reason the server needs to provide the IP address.

You might find the following java code fragment useful:

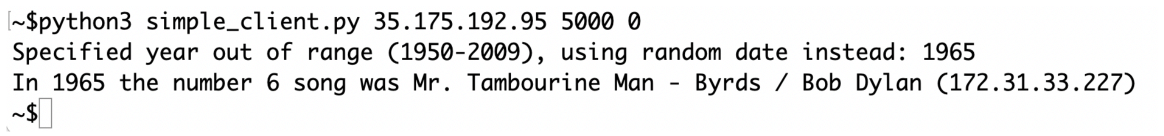
```
InetAddress.getLocalHost().getHostAddress()
```

or for Python:

```
socket.gethostbyname(socket.gethostname())

note: on some systems this won't work with localhost.  But it
will be fine on the ec2 server (yes I'm looking at you Apple)
```

When your server is running this will return a private IP address - **NOT** the public IP address we potentially used in the client to contact the server. This is fine for our needs as we just want to distinguish the server not use it as an IP.

Mine output now looks something like this:



```
~$python3 simple_client.py 35.175.192.95 5000 0
Specified year out of range (1950-2009), using random date instead: 1965
In 1965 the number 6 song was Mr. Tambourine Man - Byrds / Bob Dylan (172.31.33.227)
~$
```

8.2 ENSURING OUR SERVER STARTS WHEN THE INSTANCE IS INSTANTIATED.

Before we can make an AMI template out of our instance - we need to ensure our MusicGuru automatically starts when we start or reboot the host instance.

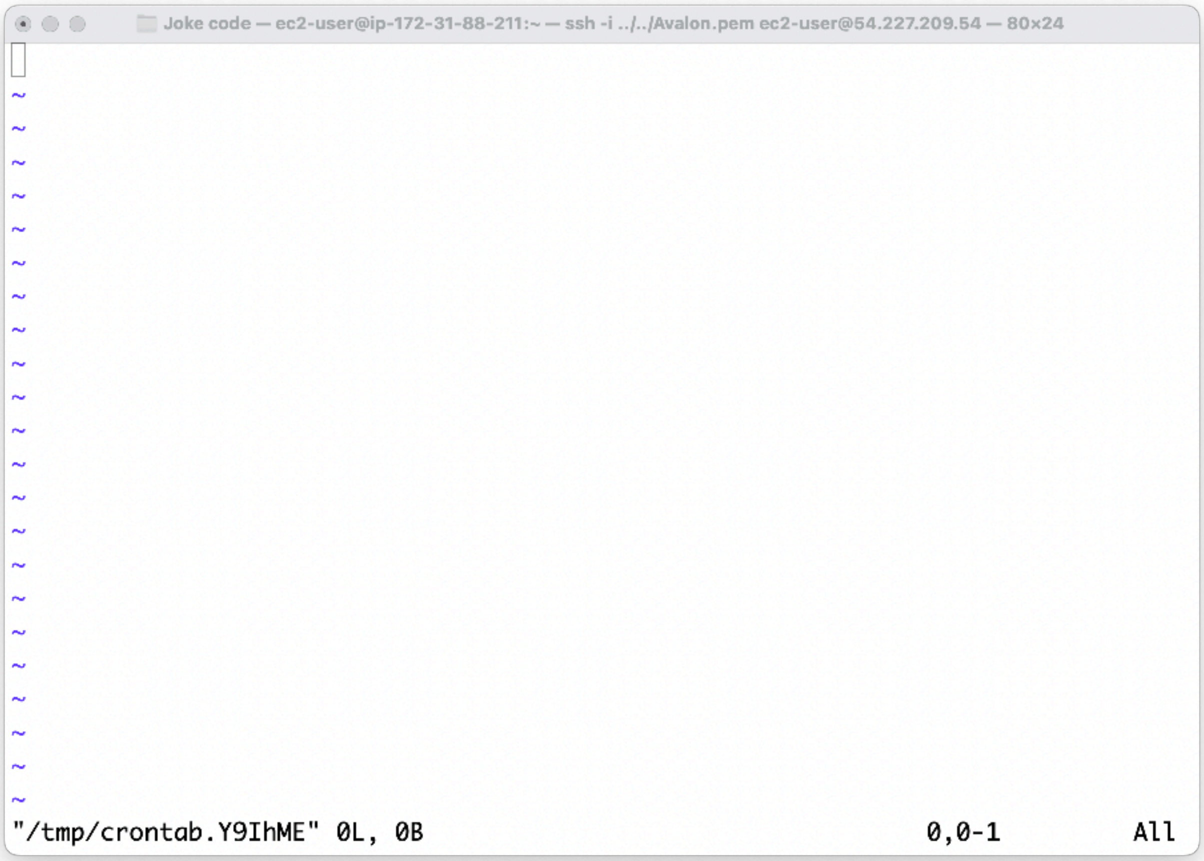
Otherwise **nothing** will happen when we made new instances from this template - and that would be boring and unhelpful.

- We want MusicGuru to run every time the instance is started or rebooted.
- The safest way to do this is to get the operating system to do it. We'll be using **cron**, this gets a little deep, hold on tight!:
    - A cron job is one that is scheduled at a specific time and run. The time we're going to choose is "reboot time"
    - But first we need to make a shell script to run our java program (for reasons to do with java).
    - In your favourite editor (or you can do this directly on the instance if you want using cat > filename and control D to finish) write…
        - for java:

          ```
          #!/bin/sh

          cd /home/ec2-user/

          java <your server name> 5000 &
          ```
        - or for python:

          ```
          #!/bin/sh

          cd /home/ec2-user/

          python3 <your server name> 5000 &
          ```
- I saved mine to a file called **run.sh** - you'll need this later.
- using scp upload this to your EC2 instance (unless you made it directly on the instance - see there are advantages)
- Now ssh into your instance (if you are not already logged in)
- and change the permissions of your shell script so it is executable,
    - chmod a+x run.sh (this is critical) check: permissions are **-rwxr-xr-x.**

- Kill your existing server and test the shell script (sh run.sh). You will need to use **ps** to see if running the shell script worked (it will not show up for the jobs command - for good reasons), or just try running your client.
- Now, here it gets a little messy...
    - in your instance we now have to set up the cron job.
    - we're going to do this as root (that is what sudo does - Super User do)
    - sudo crontab -e
    - This will throw you into an edit screen using 'vi' which is an editor of purest evil! It looks like this:


Joke code — ec2-user@ip-172-31-88-211:~ — ssh -i ../../Avalon.pem ec2-user@54.227.209.54 — 80×24

```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/tmp/crontab.Y9IhME" 0L, 0B                                  0,0-1         All
```

- Press 'i' - for insert mode to start editing, then type or paste:
    - `@reboot sh /home/ec2-user/run.sh`
- Then press the ESC key, followed by :wq to save and exit. Yuk! I am so sorry! ***Don't forget that sneaky colon!***

- You should get a response like this:

  ```
  no crontab for root - using an empty one

  crontab: installing new crontab
  ```

- Check the entry is right with:

  ```
  sudo crontab -l

          [ec2-user@ip-172-31-88-211 ~]$ sudo crontab -l

          @reboot sh /home/ec2-user/run.sh
  ```

THIS IS A GOOD POINT TO TAKE A SCREENSHOT of your terminal screen showing the crontab configuration. SAVE IT AS: 3.jpg

- exit ssh.

- Go to your EC2 instance panel
  - reboot the instance - the IP address will not change on a reboot.
    - Do this by selecting the check box for the MusicGuru instance
    - Then, using the instance state dropdown menu, select reboot.
    - wait while it restarts (this can take a while)
  - run your client to see it still connects and works (i.e., the QuoteServer restarted)
  - You could also (optional) ssh into your instance and confirm your server is running with ps -auxww | grep java or ps -auxww | grep python

THIS IS A GOOD POINT TO TAKE AN EVIDENTIAL SCREENSHOT. SAVE IT AS: 4.jpg

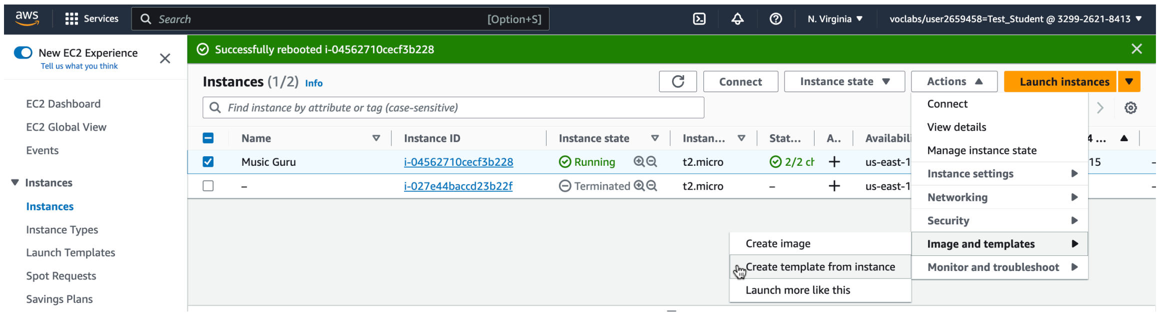
Done? Phew! That was horrible. WooHoo! Now we need to turn this instance into an Amazon Machine Image.

8.3 MAKING AN AMAZON MACHINE IMAGE (AMI)

Done? Phew! That can be tricky. WooHoo!

Now we need to turn this instance into an Amazon Machine Image. This is a 'snapshot' if you like, of a virtual machine configuration.

- Go back to the Amazon EC2 panel and choosing Instances
- With your running instance selected in the "instances" page (checks 2/2):
- Go to actions -> images and templates -> create image from instance (not a template).



Successfully rebooted i-04562710cecf3b228

Instances (1/2) Info

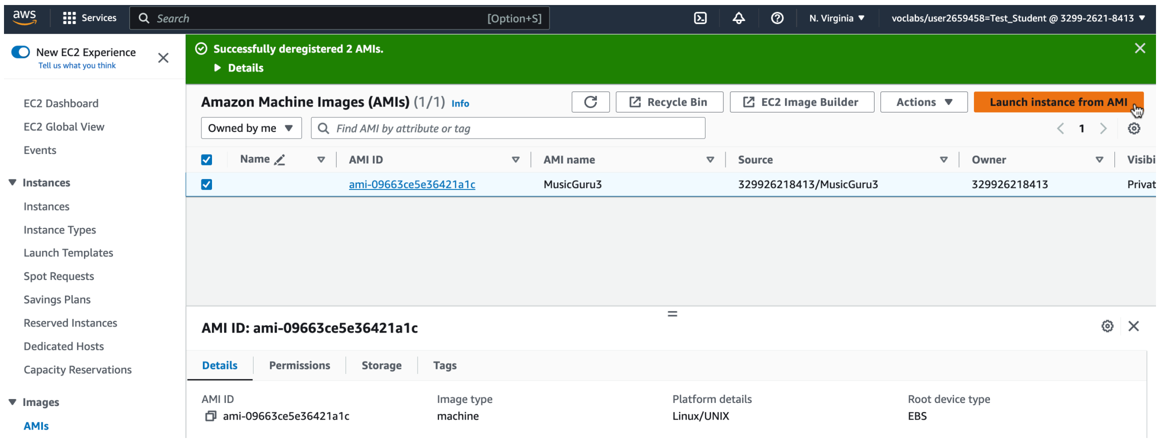
Name	Instance ID	Instance state	Instan...	Stat...	A..	Availabil
Music Guru	i-04562710cecf3b228	Running	t2.micro	2/2 ch	+	us-east-1
–	i-027e44baccd23b22f	Terminated	t2.micro	–	+	us-east-1

- Name your image
- Leave everything else as default, and
- Click "create image"

8.4 TESTING YOUR NEW AMI

Now you can launch as many instances as you like from your AMI:

From the EC2 sidebar, under images, select AMIs and launch a new copy of the AMI



Successfully deregistered 2 AMIs.
Details

Amazon Machine Images (AMIs) (1/1) Info

Name	AMI ID	AMI name	Source	Owner	Visibi
	ami-09663ce5e36421a1c	MusicGuru3	329926218413/MusicGuru3	329926218413	Privat

AMI ID: ami-09663ce5e36421a1c

AMI ID	Image type	Platform details	Root device type
ami-09663ce5e36421a1c	machine	Linux/UNIX	EBS

- When you create the instance you will need to supply the keys **and**,
- security group. You have 2 options.
    1. you can manually edit the security group to allow the port number - just like you did before, or
    2. you can look at the security group that the original instance was using, and select that from the drop down.

  This is because these settings are AWS configurations, and nothing to do with the VM itself - which is what you created.
- Now, go back to the normal instances page, and you will see your new instance that was instantiated from the AMI.
- Copy the public IP address and use your client to check everything is working (you can also ssh into it and check things there if you want). **Be patient, it can take a while to spin up a new VM.**

THIS IS A GOOD POINT TO TAKE AN EVIDENTIAL SCREENSHOT. Get both server and client screen shots, ensure IPs are clear as well as the terminal etc.

SAVE IT AS: 5.jpg

Also, take a screenshot of the AWS console showing both the original manually created instance and the new instance running alongside each other. Also take a screenshot of the images->AMI tab - so we can see your AMI. Save these screenshots as 6.jpg and 7.jpg respectively.

- You can now make as many running MusicGuru instances as you need, but,
    - they all have different IP addresses, and
    - how do we tell the client these IP addresses, and/or
    - hide from the client which server it is using?
- We'll address these problems and more in Project 3.

Submission:

Don’t forget to attribute any code sources in your code comments.

- Submit your server and client source code, with everything needed to compile or execute it:
    - your .java or .py source files - and ensure you have named as per section 3:
        - MusicGuruClient.java or MusicGuruClient.py
        - MusicGuruServer.java or MusicGuruServer.py
    - Do not submit .jar or class files.
- Submit at least the minimum 7 screen shots - I encourage you to make more in order to more fully document your own progress.
- Submit the context/discussion/commentary on each screenshot (all in 1 file please, call it context.pdf)
- If we have any questions about the submission, you may be asked to complete a demo.
- Finally: do not terminate your instance - you may be able to used it for the next lab. Choose stop instead. Be aware – once you restart your instance after stopping, you will be assigned a different IP address.
- Your uploaded files will still be present unless you terminate, and if you do terminate - then it is back to square one for you!