NWEN 243 – 20234 T2 Project 2

Creating a server on AWS: The Magic 8 Ball

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 part I of this document. In part II, you're pretty much on your own - but you still need to compile evidence, so take them **as you go**, or you'll have to go through the steps again.
- If you only partially complete a section, choose appropriate screenshot to show how far you got. I provide less indication of screenshots as the lab goes on, as you should be able to understand from the early material what is called for. If in doubt, **Additional screenshots will always help.** It is your job to provide evidence of your work functioning and being configured correctly, not for us to infer.
- You should provide contextual commentary on all screenshots you submit.

Lab Outline

- Create a client / server pair - "The Magic 8 Ball".
    - The Client/Server you develop will be the basis for learning about distribution. We will look at remote hosting, regions and load balancing in this lab.
    - We'll also use it for some of Lab Project 3. So you **really** need to make sure you get the basic software working for both this and later labs.

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 use the term "instance" but that is a cloud term - that isn't always accurate elsewhere

Part I: The “Magic 8 Ball” Service

The “Magic 8 Ball” is a toy made by Mattel

- https://en.wikipedia.org/wiki/Magic_8_Ball

You can also watch an advertorial style demonstration here:

- https://www.youtube.com/watch?v=WSaS17CSS4c

General Specifications:

- You will need to write **both** the client and a server parts of “The Magic 8 Ball”.
  - You will use TCP and IPv4.
  - You have the option of writing your client and server in Java **or** Python, or a mix of the two (but be prepared for some string handling incompatibilities).
  - Library choices in **Java and Python** are limited to the standard **socket** and **socket class** respectively, absolutely **NO** libraries that abstract away the steps in making a socket, or hiding communication over it (buffered readers and writers are fine). 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. I will provide a reference server you can test your client against. You probably ought test your server code with some of your fellow class members clients.
  - You should ensure your code is reasonably robust, commented, and does basic error handling.
- This is the specification, with examples, for your code:

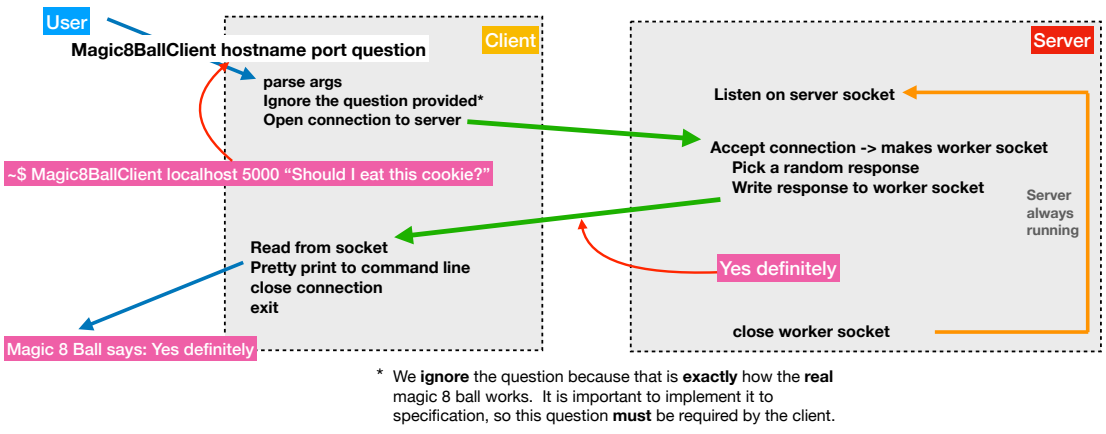


Figure 1: Code Specification

Functional Requirements:

- The server is always running (in the background) and can handle many clients in a row. Run code in the background in Unix with an '&'
- The client only runs once, and if the user wants another prediction, they need to run the client again. That is, there is no looping in the client.
- **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 will get penalised for not attributing, as that is plagiarism.

2. My Magic 8 Ball - Reference example.

- I will run my Magic 8 Ball - on embassy.ecs.vuw.ac.nz and port 32000 and not on AWS due to the - 4 hour time limit. This port and machine are visible outside university and should be available 24/7 unless someone crashes it.
- As I am providing a reference running server - you should write the client first to check you are conforming to the required functional standard. Then write your version of the server to match your 'validated' client.
- **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 Magic 8 Ball 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, 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, because those will change - potentially often.

Magic8BallServer: creates a TCP 'server' socket (you must choose a port number. You will then need to pass in the port number in a command line argument when you start your server. Be aware: port numbers below 1000 and sometimes 2000 are restricted on some machines. Ports below 16384 on ECS server machines (embassy, barrets, etc) are not accessible outside the university, ports higher than this are useable. The machine on the desk in your lab is not visible outside ECS **at all**, you can still contact and use outside services via NAT, but you can't put your server code on it and access it from outside, just in case you were curious.

Magic8BallClient: opens a TCP socket and connects to the server. It is somewhat similar to the example from Project 1, however we are now using arbitrary ports and our own 'protocol. The server name or IP, port and year arguments need to be given by the user as command line arguments. How the it works:

- The server will loop forever around an **accept** method call, which will sleep the server until a connection request is made from a client. Once the connection request is accepted, the host operating system will create a worker socket - that we will use to actually deal with the client. The next step is to choose a random Magic 8 Ball response and write it back to the client via the socket. Note, we're not needing anything from the client - a request text or anything, we're just sending the response anytime a connection is made, and then instantly closing the connection.
- 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 response out to the client using the worker socket, the client must read it, and then print it out on the client's command line.
- The server then closes the client connection and loops back around the moment you have sent the response - ready for the next new client connection (which will be on a different worker socket).
- When the client gets the response from the server:
    - Make it pretty as per the example diagram.
    - Print it out.
    - close the socket
- Running my Magic 8 Ball server and client looks like this:

```
~$java Magic8BallServer 32000 &
[1] 90407
~$Using port: 32000

~$java Magic8BallClient localhost 32000 "Should I eat this Cookie?"
Magic 8 Ball says: Most likely
```

- A quick note on jobs in the background - like the above. To see what jobs you have running in the current session (terminal) type jobs:

```
~$jobs
[1]  + running    java Magic8BallServer 32000
```

- You can kill this server by typing, kill %[job number]

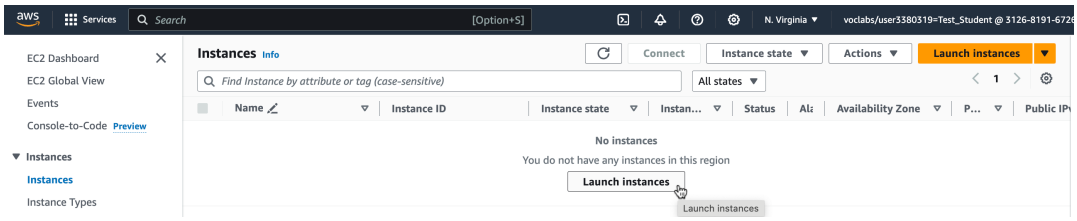
```
~$kill %1
~$
[1]  + exit 143   java Magic8BallServer 32000
~$jobs
~$
```

- As you can see above, this job is now killed. If your server complains that it can't use the port when you run it, it is possible you have one of these old ones hanging around - so you will probably need to do this while testing and refining your code. If the job is from an earlier session (you will need to kill it a different way - find the PID pass that to the kill command - which in the above example is 90407).
- A minimal functional implementation is just fine. You have a lot of leeway on how to write this client-server pair, **(VIP:) as long as you conform to the expected messaging and formatting between client and server, as given in figure 1. Compliance to this specification will be a major part of the marking.**
    - These are not long programs - in their most basic forms. Of course, you can always write more, parse things beautifully, store or manipulate the data with pizazz.
    - If you find yourself writing endless code for the socket side of things - please ask your tutor, something has probably gone off track. Using sockets is almost the same as using files - its normally pretty short and simple, adding more code isn't usually the right choice when stuff isn't working.

Evidence to Submit: take a screen shot, with two terminal windows open on your local machine, one showing the execution of the Magic8Ball server, and the other showing the client. SAVE IT AS: 1.jpg

4. Create an EC2 Instance for your “Magic 8 Ball”:

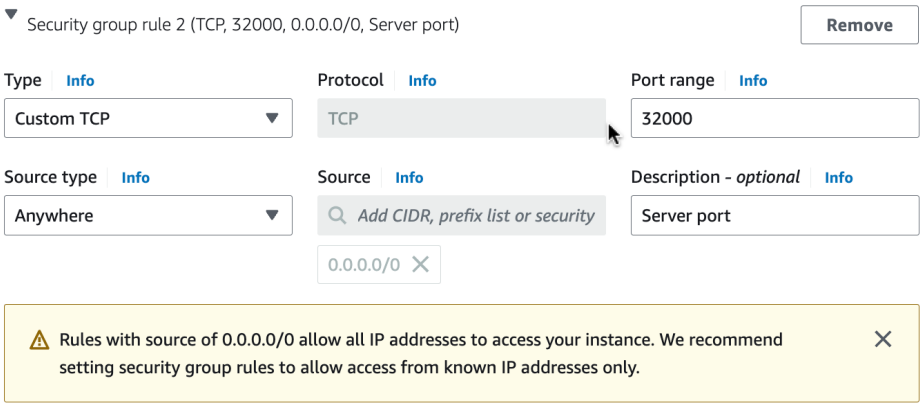
- You will need to create and configure an EC2 instance for hosting your Magic 8 Ball service.
    - Navigate to the AWS console, select EC2 from the console and then selected launch instances:



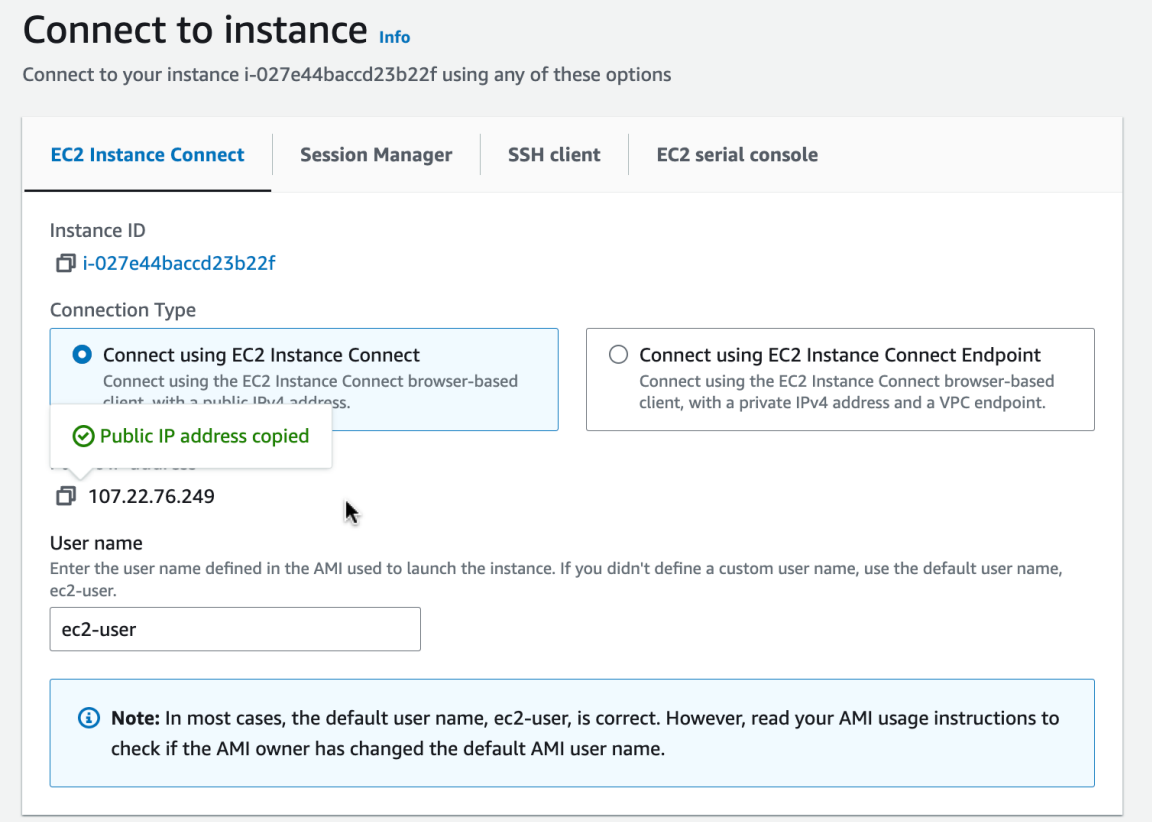
    - Fill in a name for your instance, then on the “Application and OS Images (Amazon Machine Image)” pane, choose the “Quick Start” tab and select **Amazon Linux 2 AMI (HVM)** as the operating system image to use.
    - On the next pane “Instance type” you must choose the t2.micro free tier as your instance (virtualised machine) on which the OS image will be run.
    - Select a private key pair, I recommend using the same one as Project 1. Just make sure you copy it into your current working directory for Project 2. If you make a new one - then remember to:

        chmod 400 myKeys.pem

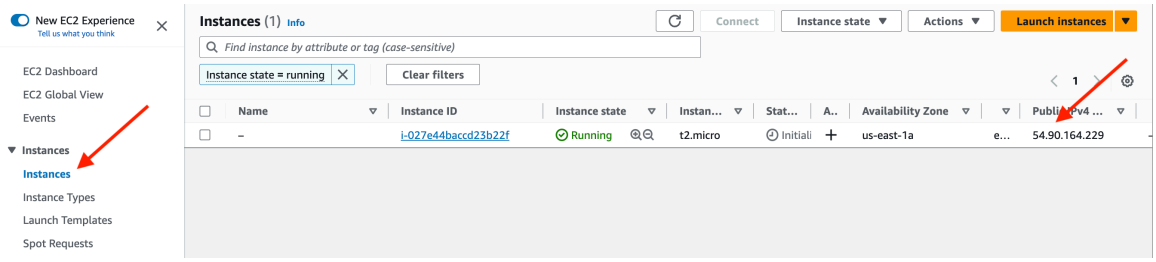
    - Under “network settings” You will need to add a custom TCP security rule.
    - Select “add security group rule” - which is at the bottom of the section. First thing, **do not delete or overwrite the default ssh rule - we need it.** You will need to make a new customTCP rule, with a port range that will be the port number you are using (mine is 3200) and ensure source is set to everywhere (0.0.0.0/0). You can ignore the nanny warning (in yellow).



- You can now launch your instance.
- When you wish to connect to your instance you can use the AWS Instance connect or SSH, as you saw in Project 1. I **prefer** people to use SSH as some important configuration issues/errors won't show up using AWS Instance connect - it is way too helpful.
- **Also, you will need to wait until the instance is provisioned, before you can connect to it - regardless of the method.**



- You can find the IP address of your instance in various places, after launch. You may need to refresh the instances screen in your browser to see the new instance.
- Or from the dashboard instances panel - I usually grab mine from here.



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, X.X.X.X being your ec2 instance IP address. Assuming it is provisioned and running.
    - *ssh* -i myKeys.pem ec2-user@X.X.X.X

6. Configure your Instance:

We now have a lovely blank unix machine 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 - or do you development work on the ec2 instance. That works too, although it isn't technically the best 'practice', dev and ops machines would normally be separate.

You might 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
- or **if you want to compile on the ec2 instance**
    - *sudo yum install* java-1.8.0-openjdk-devel

- 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, if you install the runtime, remember it is 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. But for this project, you should do what works best for you.

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 choice. 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 data files 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 Magic 8 Ball on your EC2 Instance:

- Now run your Magic8Ball 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 service running on AWS that you wrote!

Mine starting , using port 32000 and running in the background on my EC2 instance.

```
[ec2-user@ip-172-31-94-154 ~]$ java Magic8BallServer 32000 &
[1] 6877
[ec2-user@ip-172-31-94-154 ~]$ Using port: 32000

[ec2-user@ip-172-31-94-154 ~]$
[ec2-user@ip-172-31-94-154 ~]$ jobs
[1]+  Running                 java Magic8BallServer 32000 &
[ec2-user@ip-172-31-94-154 ~]$
```

Running my client on my local machine:

```
~$java Magic8BallClient 34.227.178.0 32000 "Can I haz chezburger"
Magic 8 Ball says: My sources say no
~$
```

Evidence to Submit: A screen shot, with two terminals open, one showing the execution of the Magic 8 Ball 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 Magic 8 Ball

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 'image' of our configured running service, so we can run as many copies as we need, at any particular time.

8.1 MODIFY YOUR CODE

We're going to need to make a small modification for testing and assessment purposes. As 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** we would not want the user to see this.

Modify your Magic 8 Ball **server** code to return its IP address as a post-fix (at the end) when you return a prediction. 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 the 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.

My output now looks like this:

```
~$java Magic8BallClient 34.227.178.0 32000 "Can I haz chezburger"
Magic 8 Ball says: Outlook good (172.31.94.154)
~$
```

8.2 ENSURING OUR SERVER STARTS WHEN THE INSTANCE IS INSTANTIATED.

Before we can make an AMI image from our instance - we need to ensure our Magic8Ball server automatically starts when we start or reboot the host instance.

Otherwise **nothing** will happen when we started up a new instance from this image, we'd have to go in and start The Magic 8 Ball server manually - and that would be boring and unhelpful.

- We want Magic8Ball 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[1] editor (or you can do this directly on the instance if you want using cat > filename and *control* D to finish - which I prefer) write…
        - for java:

          ```
          #!/bin/sh

          cd /home/ec2-user/

          java <your server name> <your port> &
          ```

        - or for python:

          ```
          #!/bin/sh

          cd /home/ec2-user/

          python3 <your server name> <your port> &
          ```

- 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

---

[1] be careful, some editors use hidden control characters which may cause your instance to crash.

- change the permissions of your shell script so it is executable, like this:
    - chmod a+x run.sh (this is critical) so you should check the permissions are **-rwxr-xr-x**

```
[ec2-user@ip-172-31-94-154 ~]$ ls -la run.sh
-rwxrwxr-x 1 ec2-user ec2-user 59 Aug  2 01:31 run.sh
[ec2-user@ip-172-31-94-154 ~]$
```

- Kill your existing server and **test** the shell script (sh run.sh). You will need to use **ps** (ps -aux | grep java) 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:

    - 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
```

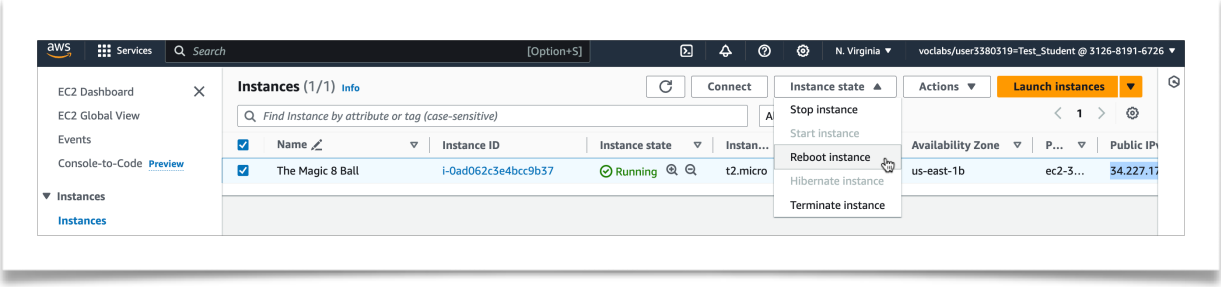
- Check the entry is right with:

```
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 Magic8Ball instance
    - Then, using the instance state dropdown menu, select reboot.



    - wait while it restarts (this can take a while)

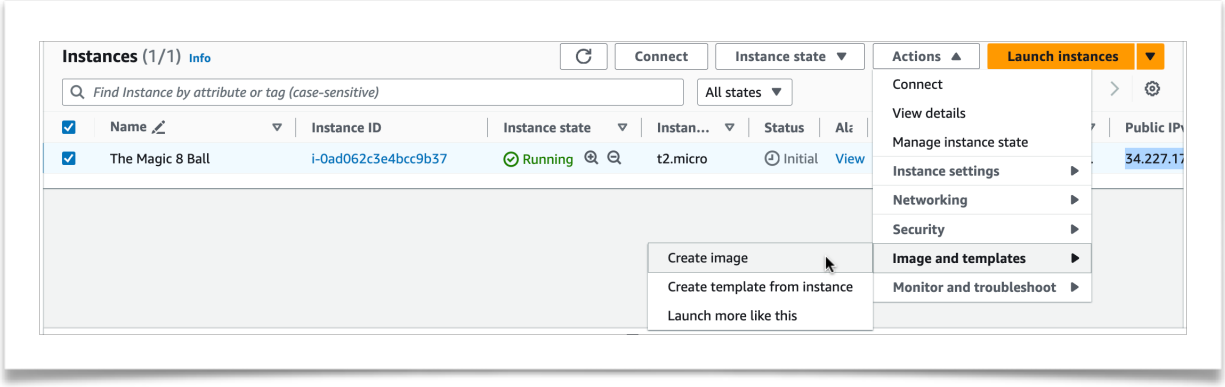
- To test:
  - run your client to see it still connects and works (i.e., the Magic8BallServer restarted), or,
  - You could also (optional) ssh into your instance and confirm your server is running with ps -auxww | grep java or ps -auxww | grep python

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

8.3 MAKING AN AMAZON MACHINE IMAGE (AMI)

Now we need to turn this running Magic8BallServer 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 choose Instances
- With your **running** instance selected in the "instances" page (checks 2/2)
- Go to actions -> images and templates -> create image.

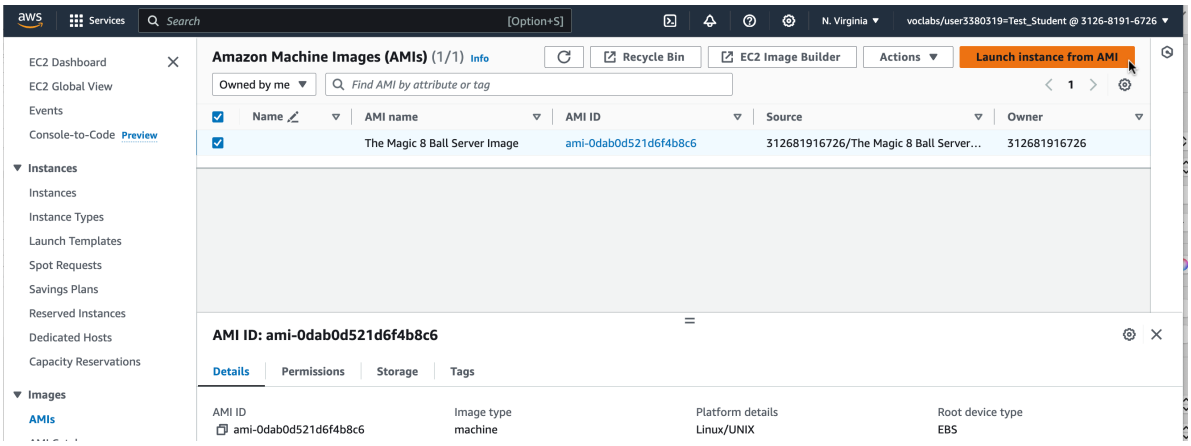


- Name your image
- Leave everything else as default, and
- Click "create image", it can take a little while sometimes, so you may need to be patient.

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, select the image and click "Launch instance from AMI".



- When you get the next page to create the instance you will need your key **and**,
- security group. You have 2 options (option 1 is better).
    1. you can select the same security group that the original instance was using from the drop down, or
    2. you can manually edit the security group to allow the port number - just like you did before, which will create a new security group, which is messy.

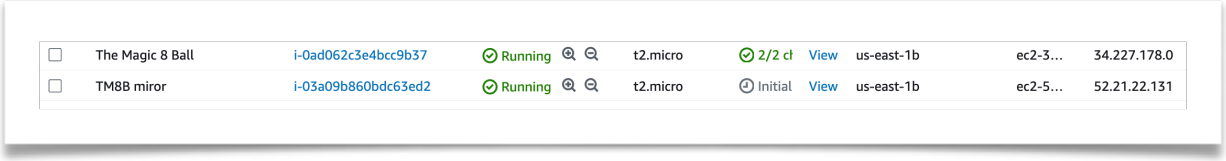
  This is because these settings are AWS configurations, and nothing to do with the VM itself - which is what you created.
- Click "Launch"
- Now, go back to the normal instances page, and you will see your new instance that was instantiated from the AMI.
- Copy the public IP address and use your client 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. Take a screenshot of both original and AMI instances being tested with your client (ensure IPs are clear) and save these as: 5.jpg, I'd expect something like this:

```
~$java Magic8BallClient 34.227.178.0 32000 "Can I haz chezburger"
Magic 8 Ball says: Yes (172.31.94.154)
~$java Magic8BallClient 52.21.22.131 32000 "Can I haz chezburger"
Magic 8 Ball says: Without a doubt (172.31.82.154)
```

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 - that is the first image in section 8.4.

Save these screenshots as 6.jpg and 7.jpg respectively - something like the image below would be 6.jpg.



- You can now make as many running Magic8Ball instances as you need, but,
    - they all have different IP addresses, so
    - how do we tell the client these IP addresses, and/or
    - hide from the client which server it is using?
- We'll address these issues next!

Part II Making it Scale - load balancing for replication

Up until now we've only looked at a deploying a single instance of your Magic8Ball service - remember the raison d'être for cloud - is elasticity and scalability, so that is what we will explore in this lab - in particular, automated horizontal scaling of VMs with the AWS load balancing service.

What we need to do use your Amazon Machine Image (AMI) of your Magic8Ball server, so that we can create any number of essentially identical duplicates - we can then configure scaling using policies and observe what happens using the cloud watch monitoring data.

This is also make our application for resilient, as if one server fails, or a network becomes unavailable, we can simply use one of the other replicas, and best of all, the user should not even notice.

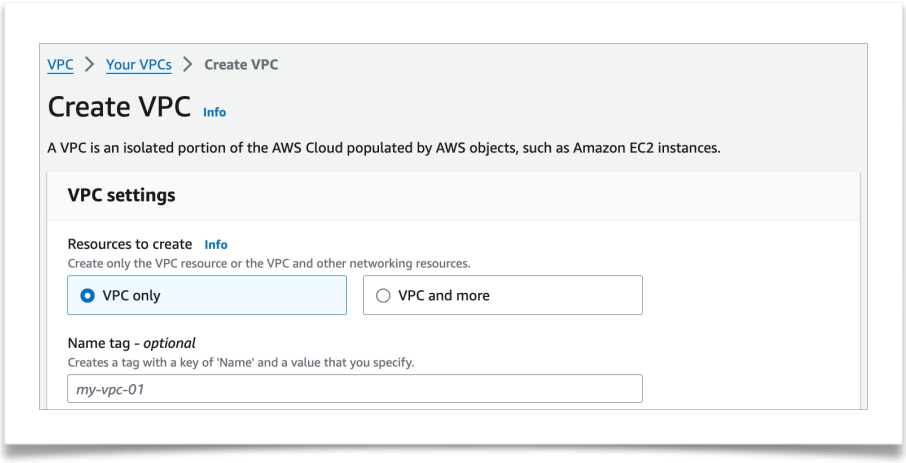
WARNING

You will get charged by AWS for the load balancer. Shut it down when you're done collecting your evidence or you could run out of budget later.

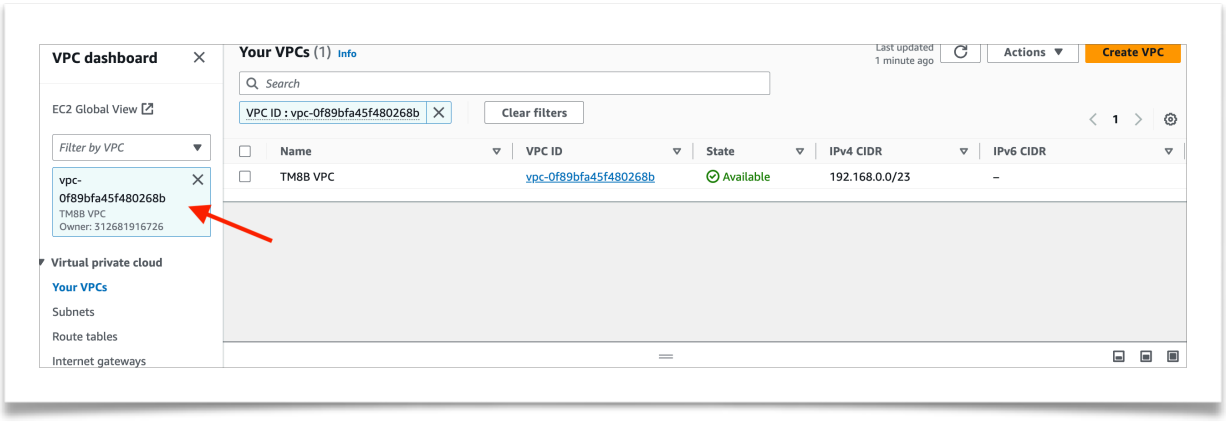
We Need a VPC to Span Availability Zones:

What we want to do is create a Virtual Private Cloud (VPC) for our load balancing as the **default** one AWS set up for us does not do what we need. This is essentially our own little TCP/IP network - virtualised on the AWS cloud.

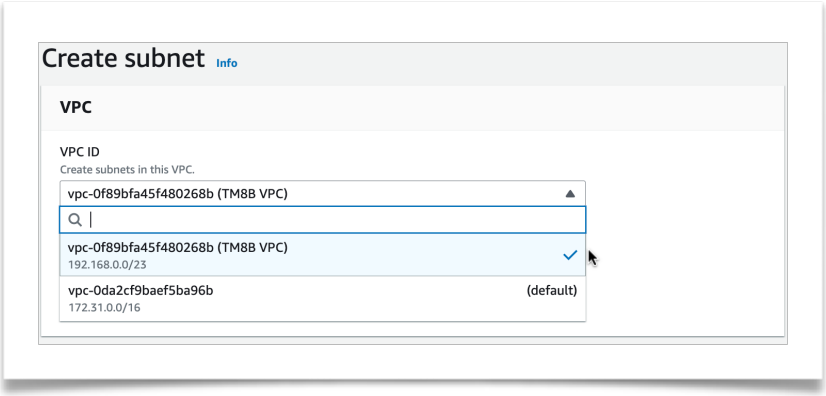
- You will need to use the search box, and search for “VPC”
- Select "Your VPCs" and "create VPC” - ignore the “VPC and more option”.
- Select VPC-only, and choose a name for it.



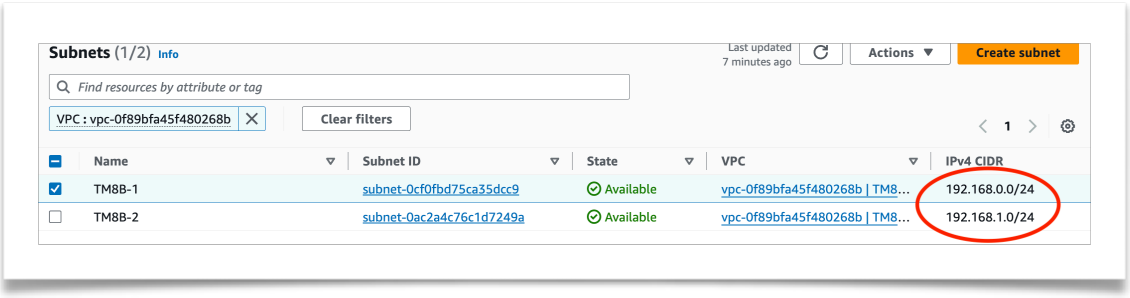
- Select the button, “IPv4 CIDR manual input”
- In the "IPv4 CIDR” I used: 192.168.0.0/23 You can choose whatever class you want here.
- Then select create.
- On the left bar, choose filter by VPC so you only see things associated with the new VPC

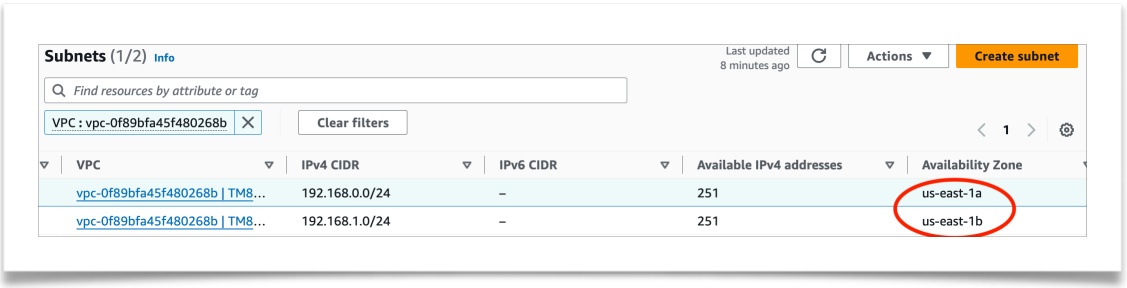


- Select “subnets” from the left menu, and then choose “create”.
- You will need to choose the VPC you just created from the drop-down. Don’t choose the default or everything will go terribly.



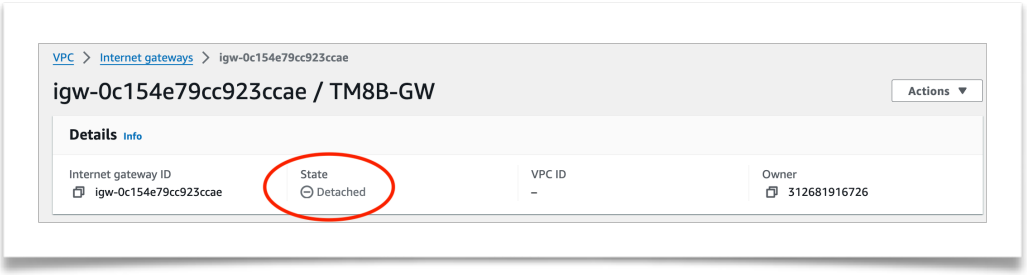
- Follow the wizard prompts to Create 2 public subsets - I called mine TM8B-1 and TM8B-2.
  - You need to make sure they divide the IP range of your VPC in 2. So if you used 192.168.0.0/23 for your VPC, then the 2 subnets that will divide this exactly in half will be 192.168.0.0/24 and 192.168.1.0/24 - which will each have 251 available IP addresses (overheads).
  - You also need to choose each subnet a different availability zone. I selected us-east-1a and us-east-1b, simply because they were the first two on the list - it doesn’t matter, as long as they’re different. See below:



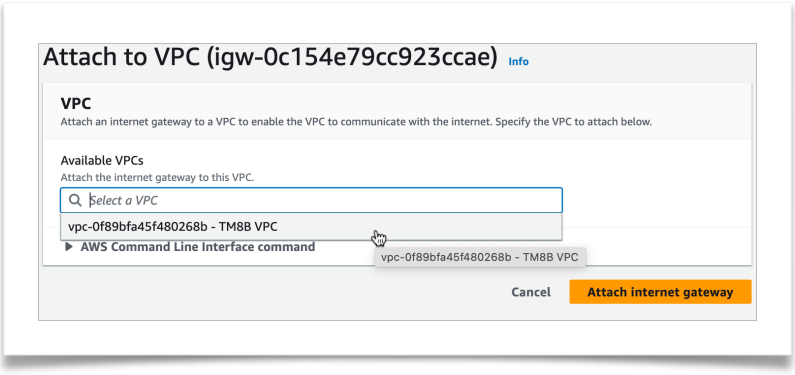


- We now have a VPC – divided into 2 subnets, each in a different availability zone. We need to link our public sub networks to the big bad outside world. We do this by attaching an “Internet Gateway” to the public VPC.

- Click the “internet gateways” on the left sidebar, create and name your gateway

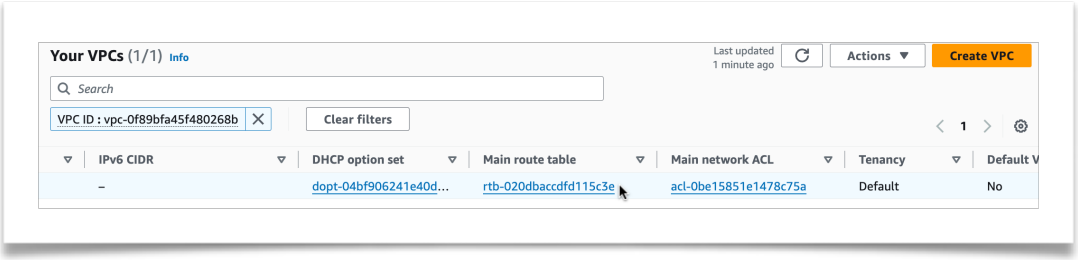


- Notice how the gateway says it is detached. We need to attach it to our VPC network. Select Actions-> “attach to VPC”

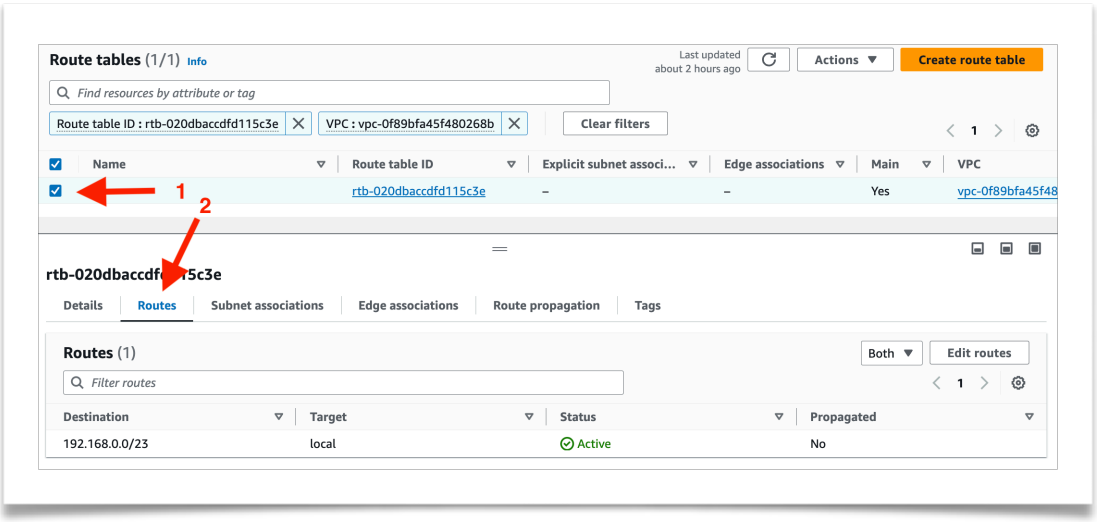


- In the drop down choose your VPC and attach.
- Next we need to configure routing, so that our network routes to and from the internet using our new gateway. All the default routing table knows how to do, is to route local traffic. It does not know about how to route external traffic. We will add a route to do this.

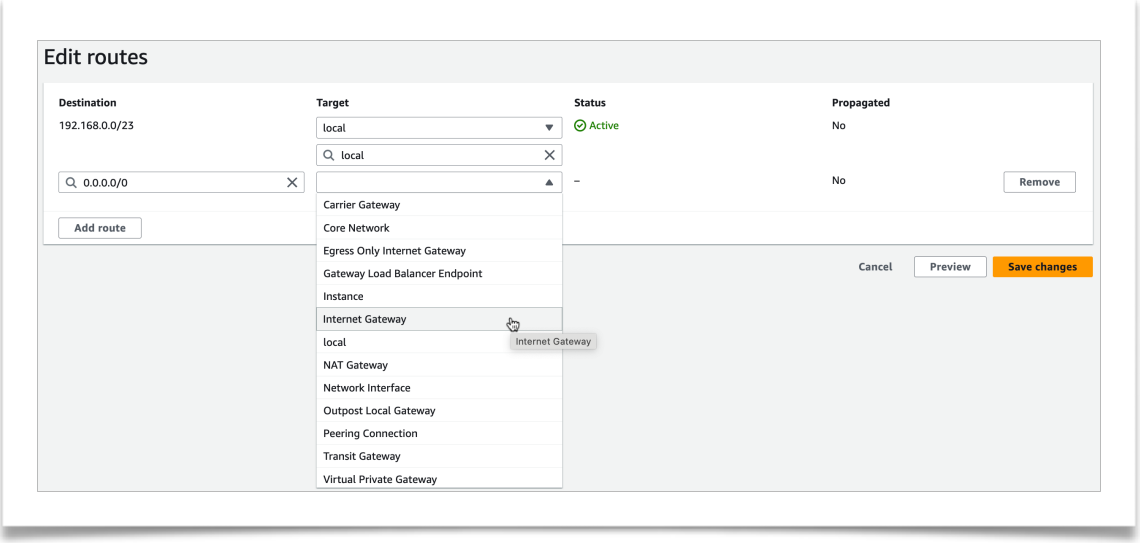
- Go back to the VPCs page, scroll the VPC entry to the right and click on the link to the main route table - you may need to refresh the page to show the new default route table show up.

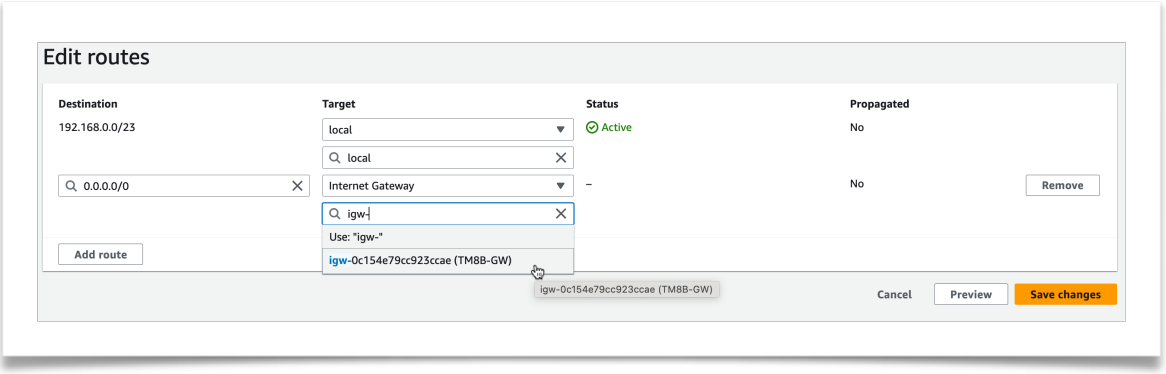


- Select the route table (arrow 1), select the routes tab that is revealed (arrow 2), and you will see the existing route for local (within the subnet).

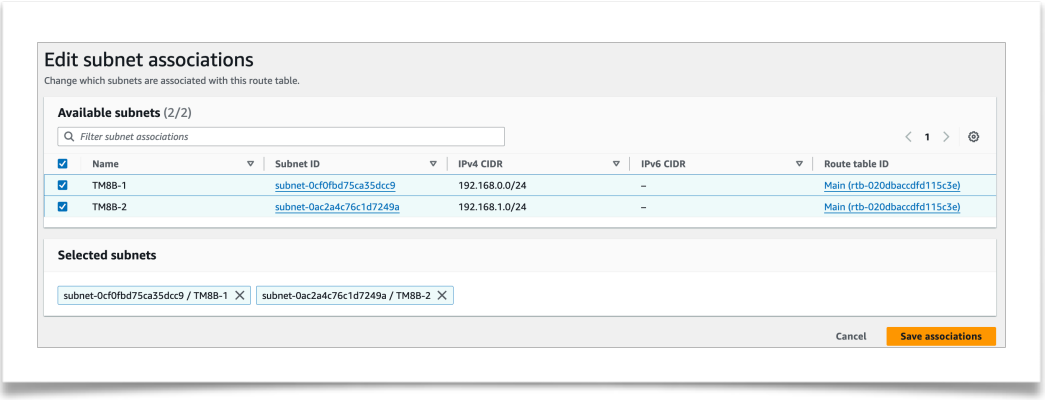


- Click Edit Routes and add a route for 0.0.0.0/0 (the outside) with “Internet Gateway” as the target so we can route network traffic externally, choosing the Gateway you made (and named) earlier.





- The last step is to associate this new routing table with the subnets created earlier, mine were TM8B-1 and TM8B-2. On the routing table - look for the subnet associations tab.
- Edit the subnet associations to associate the routing table with your two subnets and save.

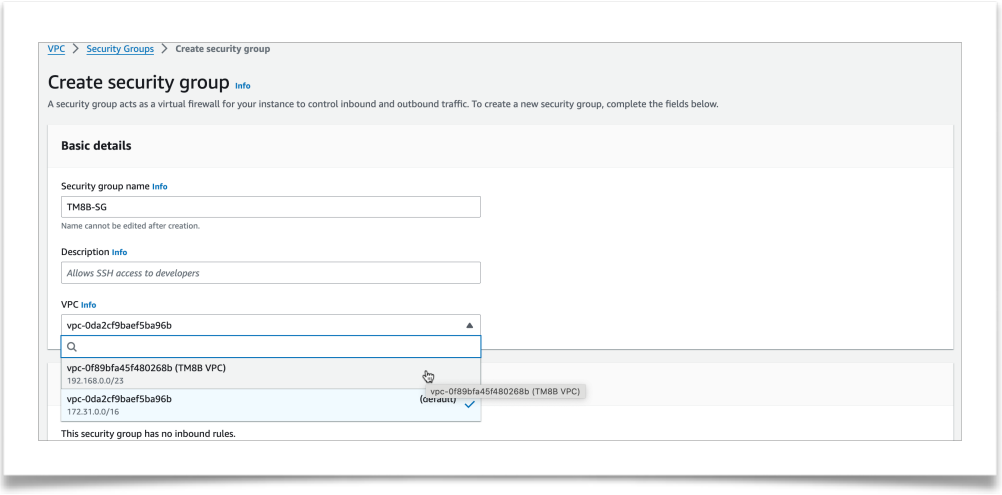


- We now have a VPC with 2 subnets in different availability zones that we can route internet traffic to.
- Now when we create instances, we can place them in either zone - we need that for our load balancer.
- Double Check your work by looking for the associated subnets in the “subnet Associations” tab.

Manually Create a Security Group

- We will need a security group later. This will control the access to the ports of the instances we create - that is, list the ports that are 'open' to incoming traffic.

- Choose security group from under "Security" from the left menu, the name it, add a description, and replace the default VPC with the one you just made.

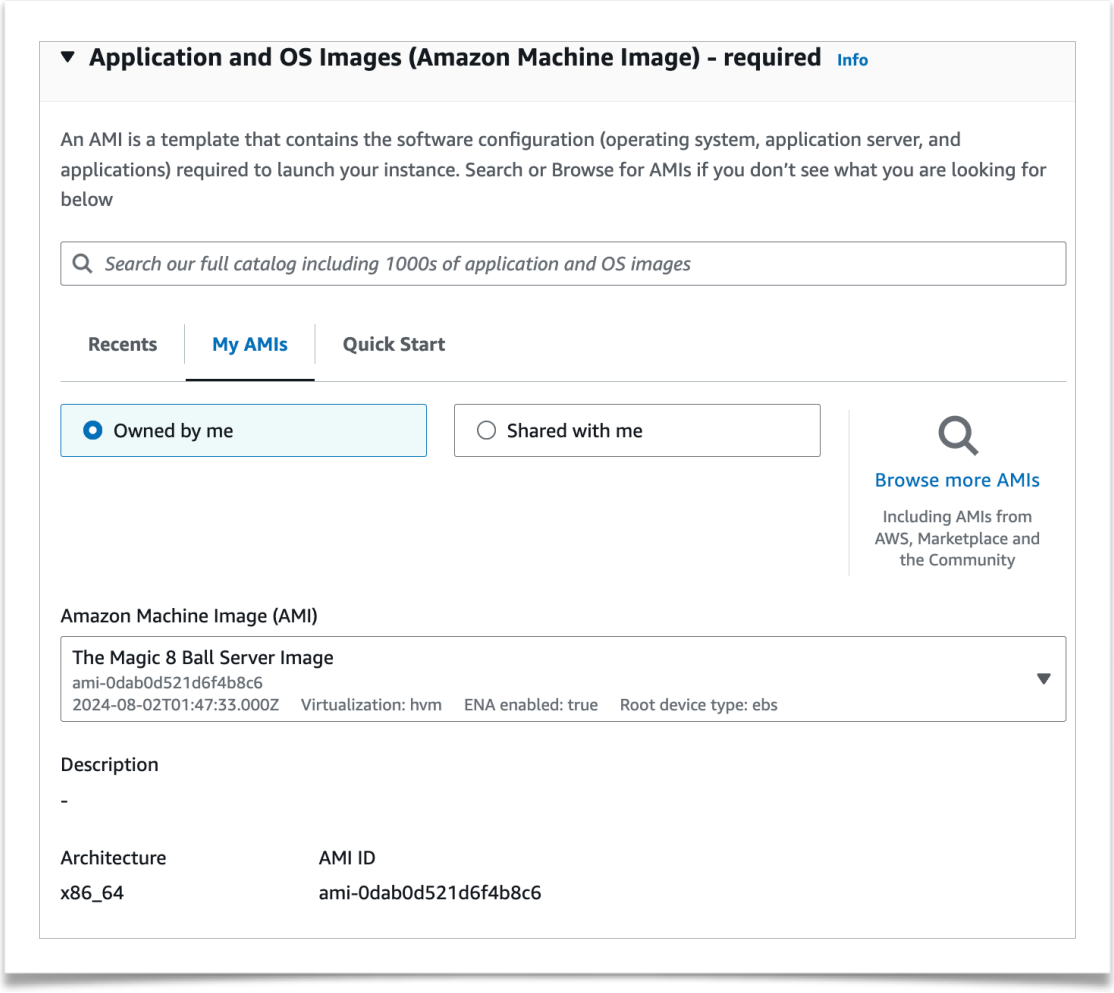


- Add 2 **inbound** rules one for SSH and one for TCP (customTCP) with your server port (recall mine is 32000). Ensure that for both of these, the traffic can come from anywhere - set this with the "source" dropdown and set it to "anywhere-IPv4"

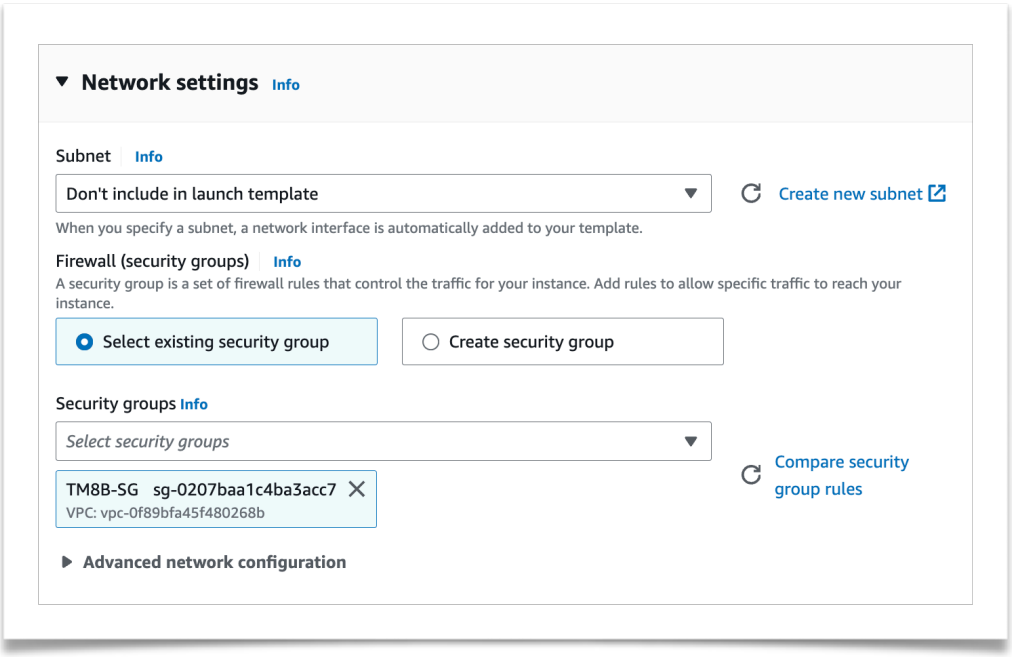
- Click Create.

Creating a Launch Template

- We are going to convert our AMI to a launch template that AWS can use to create instances on demand.
- Go back to the EC2 dashboard (you can type ec2 into the search bar) and select "Launch Templates" from the left bar of the EC2 dashboard under “instances”
- Click “Create Launch Template”
- Name your template
- Check the box - “Provide guidance to help me set up a template that I can use with EC2 Auto Scaling”
- scroll down to the “Application and OS Images (Amazon Machine Image)” panel
- Select the “My AMIs” tab and select the AMI you created in the first part of this lab.



- Next, scroll down to “Instance Type” and select the free-tier t2.micro.
- Then - select your key pair - the one you used to create the AMI.
- Under network settings - select “**Select existing security group”** and choose the one we just made.



- Next, select “Advanced network configuration” to get the drop down
  1. Add network interface
  2. Enable “Auto assign public IP” (it is on the right side).
  3. Delete on termination - choose yes (it is on the left side, lower down).

- You can leave everything else on default settings.
- Click “create launch template”

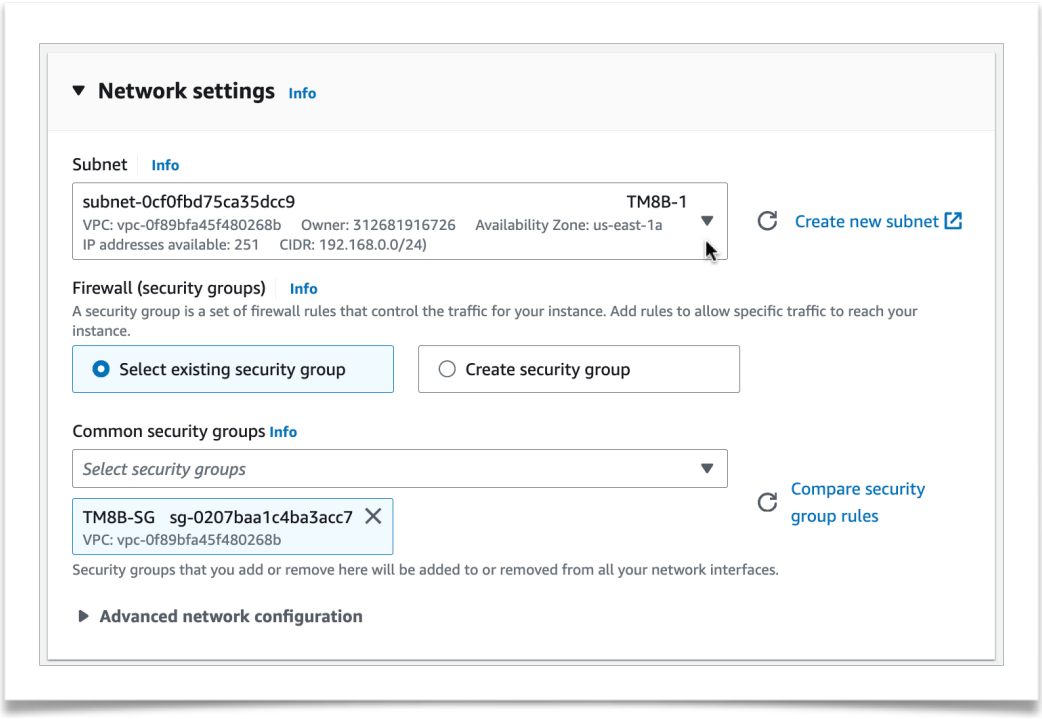
- Assuming everything went OK, and you got a success message:

    - You now have a launch template that AWS can use to create instances on demand.
    - Each instance will use your AMI - which means your Server code will start up automatically on boot.
    - You have also defined the VPC and Security group that will be associated with this template.

Some Testing

- You can now test your Launch template is working, by selecting Launch templates, then your template. Action - launch instance.

- Choose in turn each of your two subnets under network settings, and launch instances in them.



- You should now be able to view them in your instances list, harvest the IP addresses from the instances list and connect to them using both your client and ssh.

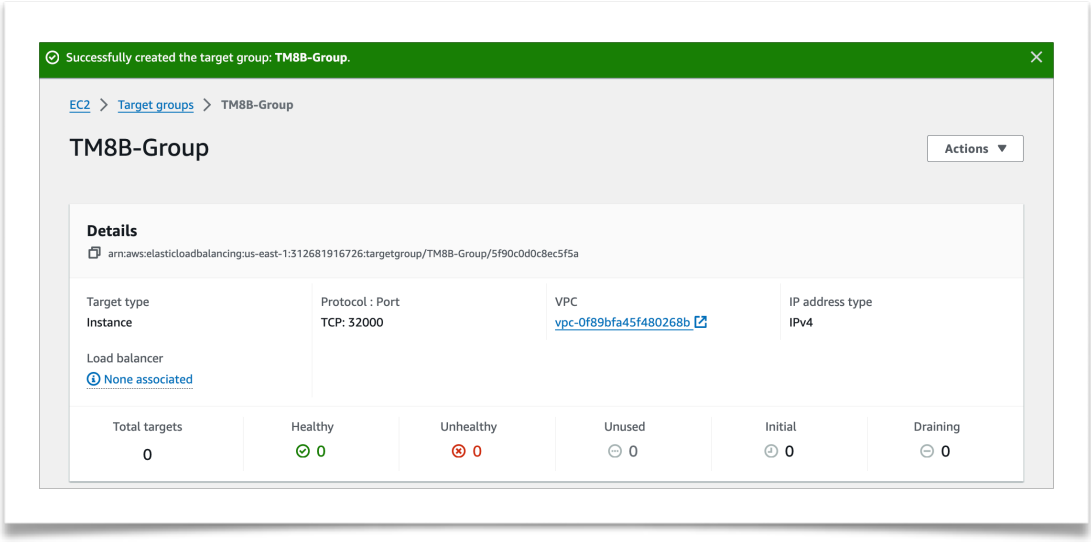
- Make sure that **both** ssh and your client work, so we properly test that both ports 22 and 32000 (in my case) both work. If you don't test both, we'll run into really tricky problems to diagnose later.

- Once we're sure we can create working instances from the templates, we can create a load balancer that will create new instances on demand, and direct client traffic to any of the instances, all transparently to the user!

- After you're done testing, **terminate** the instances **created** by the template (using instance state -> terminate), these will be the ones with no name. Otherwise things will get messy and you will likely lose track of what is what.

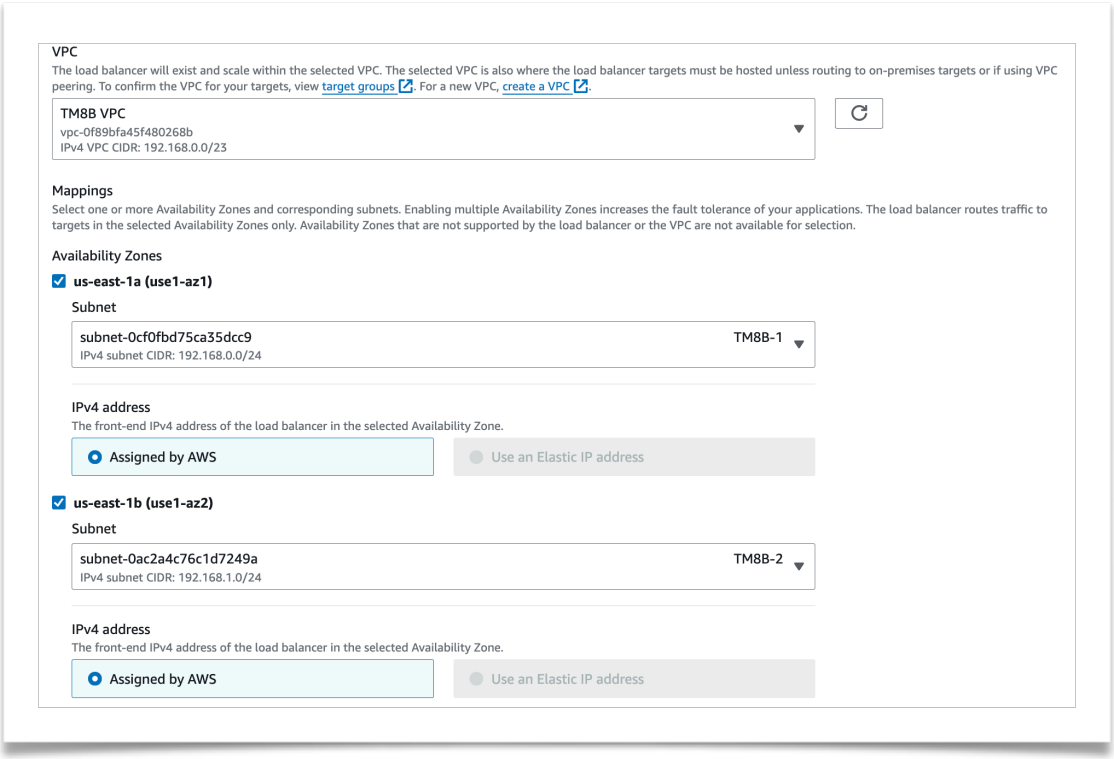
Creating a Load Balancer (and a Target Group)

- What we're going to do in this section is create a load balancer that can balance traffic across multiple EC2 instances and availability zones. There are a couple of ways to do this, including directly in the Autoscaling group wizard - but we're going to do it manually, so as to make it explicit, and less black box magic.
- Go back to the EC2 menu, and choose "Target Groups" under "Load Balancing". What a Target group does is define where to send the traffic that comes into the load balancer. This allows you to separate, say, traffic coming from mobile apps from other traffic. We will only use **one** Target Group.
- Choose "Create Target Group"
- Leave the default as "instances", name your group and change the protocol from HTTP to TCP and change the port from 80 - to whatever your server port is (mine is 32000).
- Next, select your VPC from the drop down.
- Change the health check protocol to TCP - you don't need to change the health check port, as it will default to the traffic port you specified above. However, if you were using a different port for the health checks (like we did last year - as we used a different application), you'd need to update it under the advanced options.
- Select next
- We don't have any suitable instances yet - so skip the next step and **create** the target group. You should get something that looks like this:

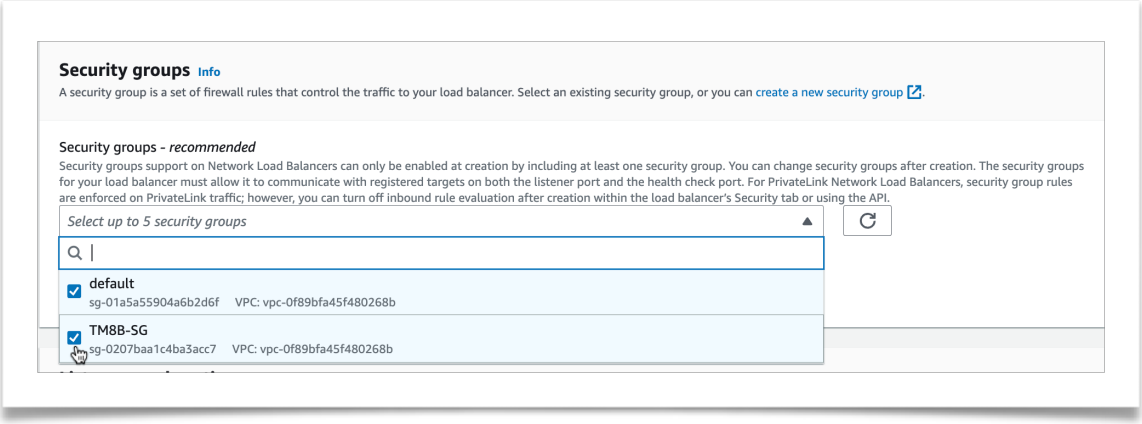


- We don't have a load balancer associated with this, so we need to create one.

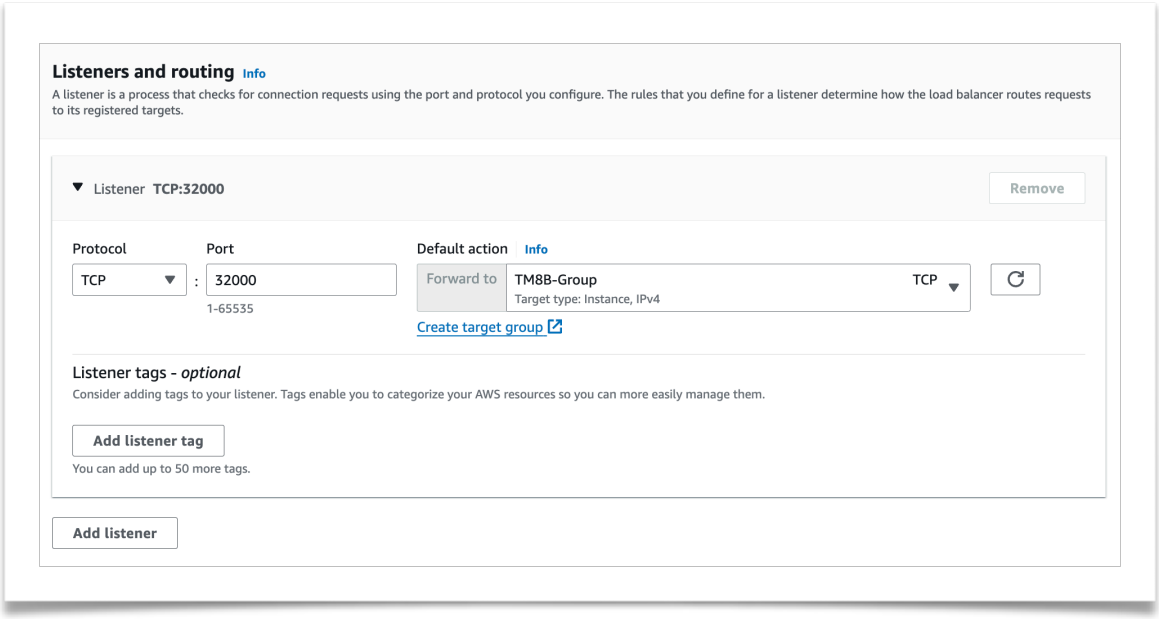
- On the left menu, select "Load Balancers" and Click "Create Load Balancer"
- Select create under the "Network Load Balancer" pane - because we have a TCP application.
- Expand the: "How Elastic Load Balancing works" and read it. It is important.
- Name your load balancer, leave the defaults as "internet-facing" and IPv4.
- Under Network mapping, select the Magic8Ball VPC we just made, and you will see the mappings update to the 2 subsets in 2 regions that we defined. **Select both.** Leave the IPv4 address to be assigned by AWS.



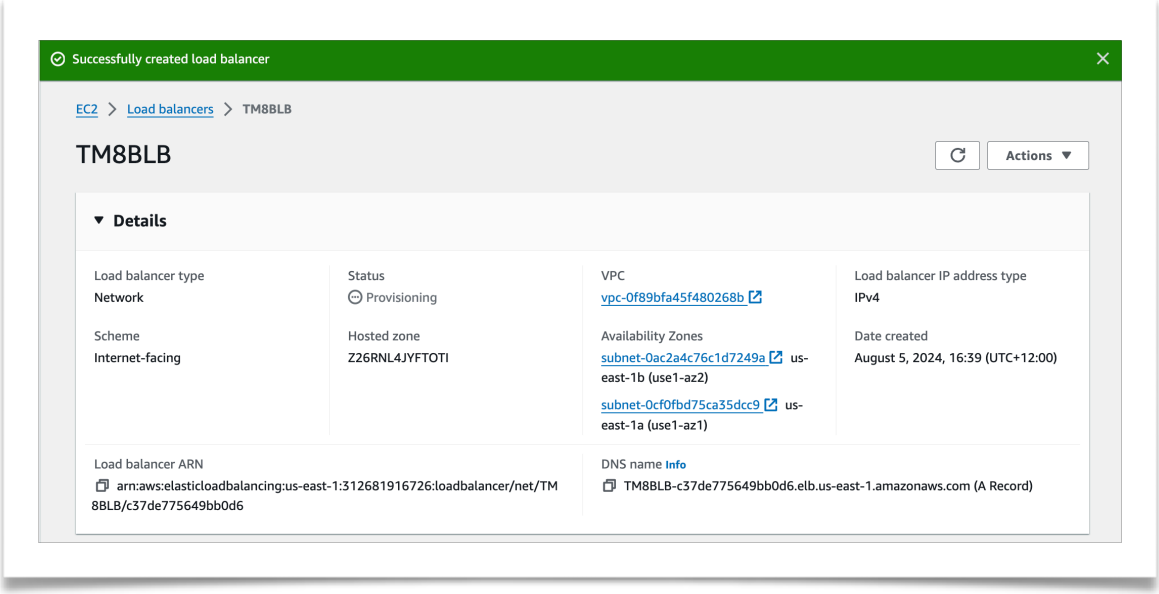
- Under "Security groups" choose the security group you made and **add** it alongside the default one.



- Under “Listeners and routing” choose the target group we just made and correct the port.



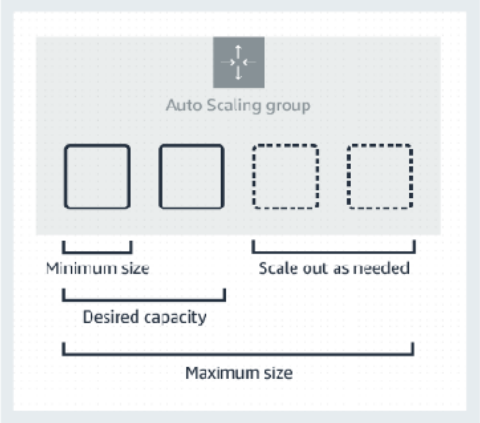
- “Create” the Load Balancer, and you should get a success message like this:



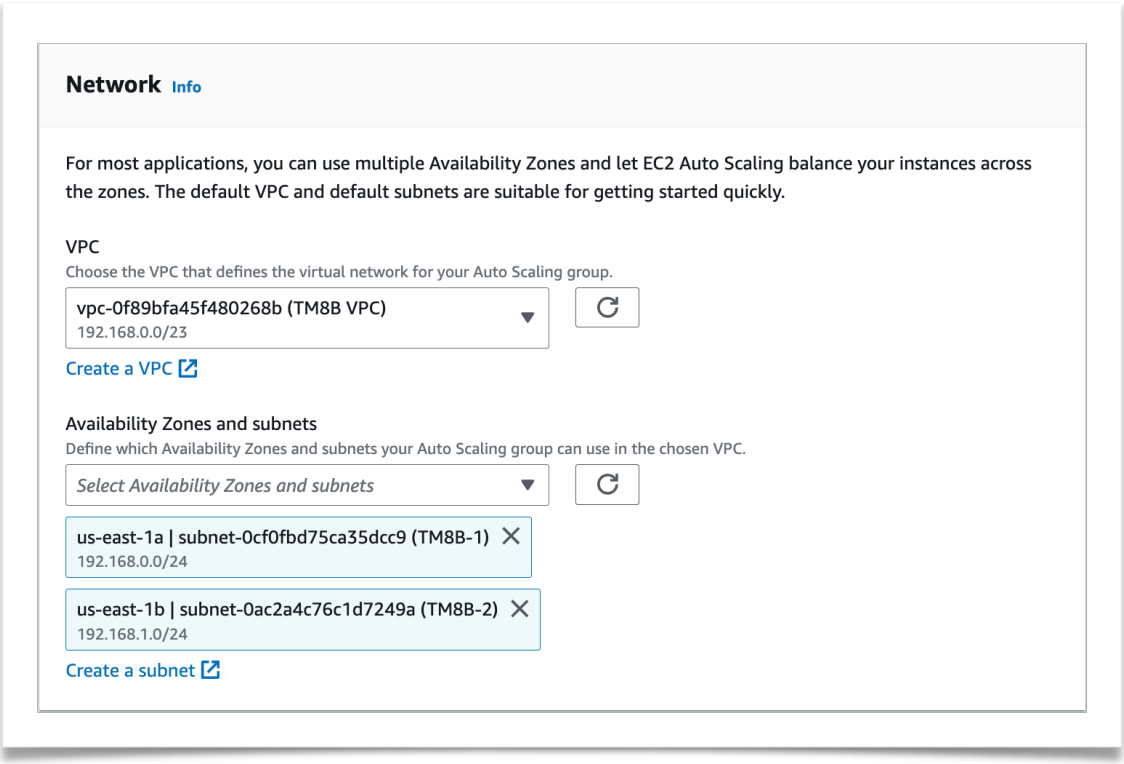
- Don’t worry about the ‘provisioning’ status - there is no need to wait until it is ready, it may take some time anyway - so we’ll continue.

Creating an Autoscaling Group

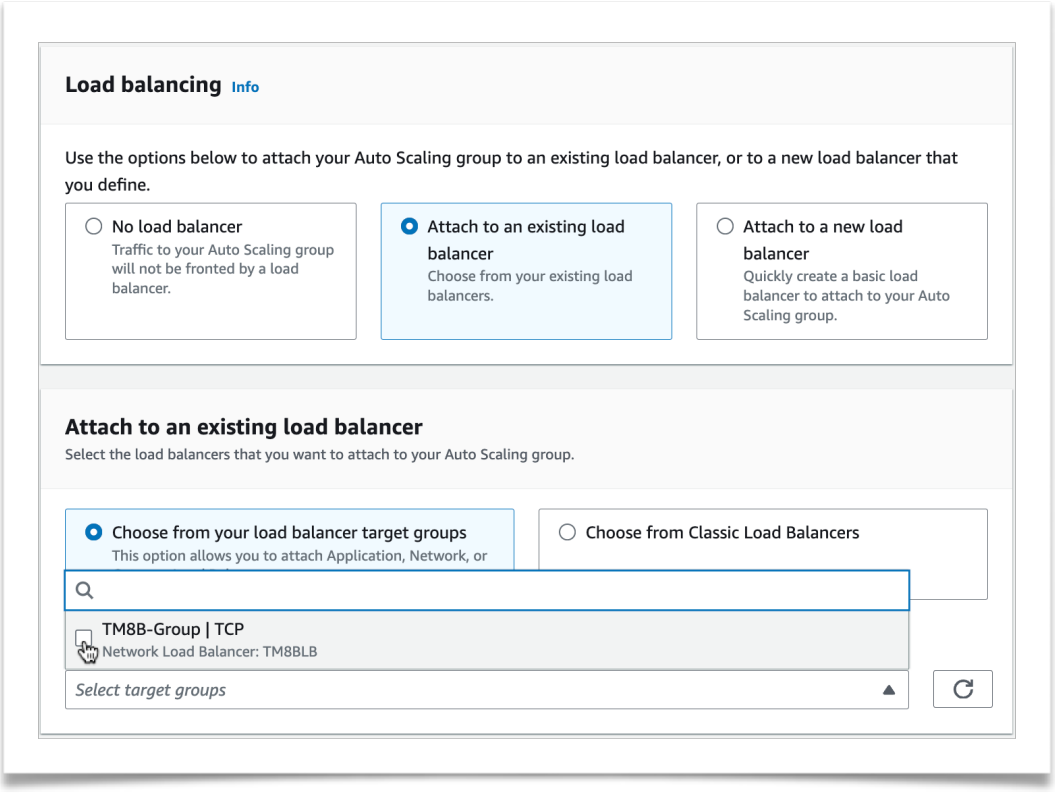
- Select Auto Scaling Group from the Left bar
- Read the page about how all this works.



- Then click “Create Auto Scaling Group”
- Name and then select the launch template you made
- Version - leave it as default.
- Click next
- Choose your VPC and **both** of your subnets, using the boxes on the dropdown.



- Click Next
- On Configure advanced options
- Select “Attach to an existing load balancer”, and choose yours from the drop down.

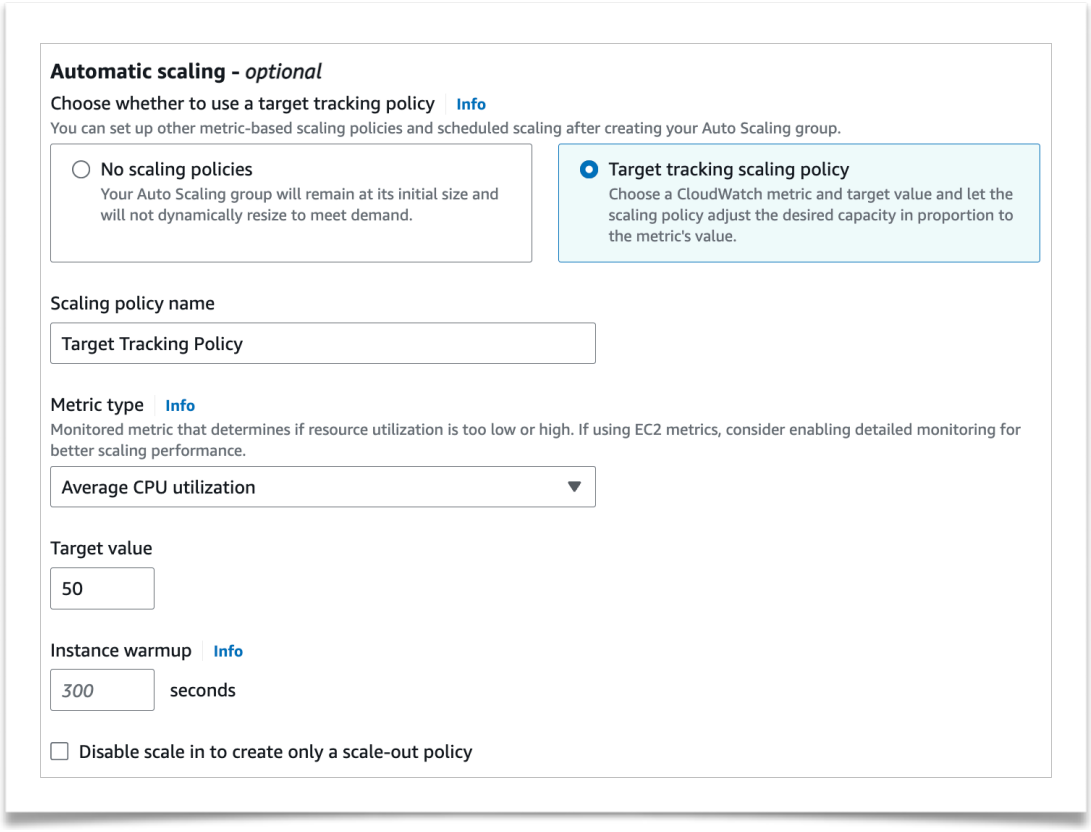


- Scroll down
- Do **not** enable “additional health checks”, the basic ones are enough.
- Select “enable group metrics collection within CloudWatch”
- Click Next

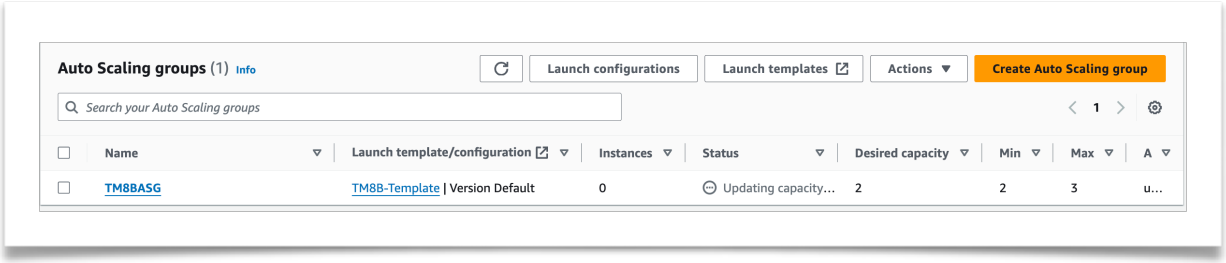
Configuring Scaling

- Set the group size - desired capacity to 2, this will control how many instances are launched . Then in the scaling pane, set the min desired capacity to 2, and the max to 3. This will control how the instances will be scaled depending on demand. 3 instances maximum is safe, there are secret limits in AWS Academy accounts - and it won’t tell you when you exceed them, it just won’t work - very frustrating!

- Choose “Target tracking scaling policy” - I suggest you leave the defaults, we’re really just turning it on.



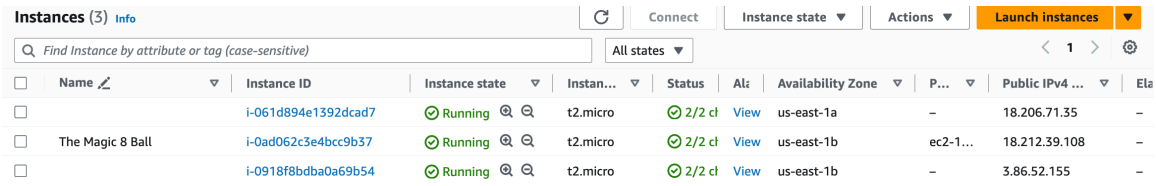
- We don’t need any of the other cool features so click “skip to review”
- Check the review panel
- Click create autoscaling group



- New instances will eventually be spooled up to match your policies, you may need to wait a while.

Checking it works (Testing)!

1. Now click back to the instances pane, and you should eventually see the number of new instances which you set as your desired capacity. The ones created by the autoscaler/LB are the unnamed ones.



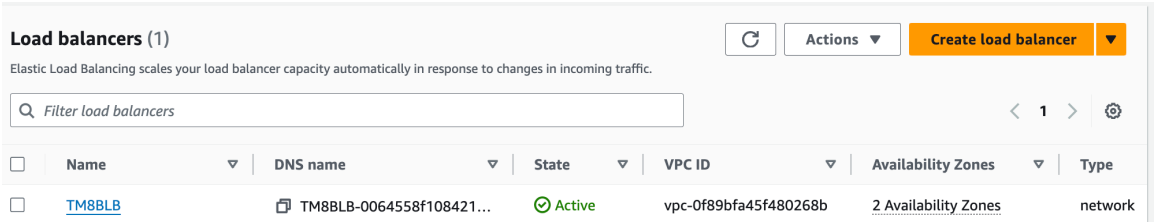
2. The idea is because we set these up with a public IP address (this is separate to how they are accessed via the Load Balancer) - then we can test them the same as normal. You can copy the IP addresses from the instances list and test with your client and SSH, to ensure they are normally working instances. Ones that the autoscaler happened to make for you - according to your workload policies.

   It is worth noting, that in the image below, you can see the outside public IP address being used by the client, but your running app, only knows about its local IP address, which is the one which is inside the VPC you made. This is because we're on the inside of a NAT (Network Address Translation) system.

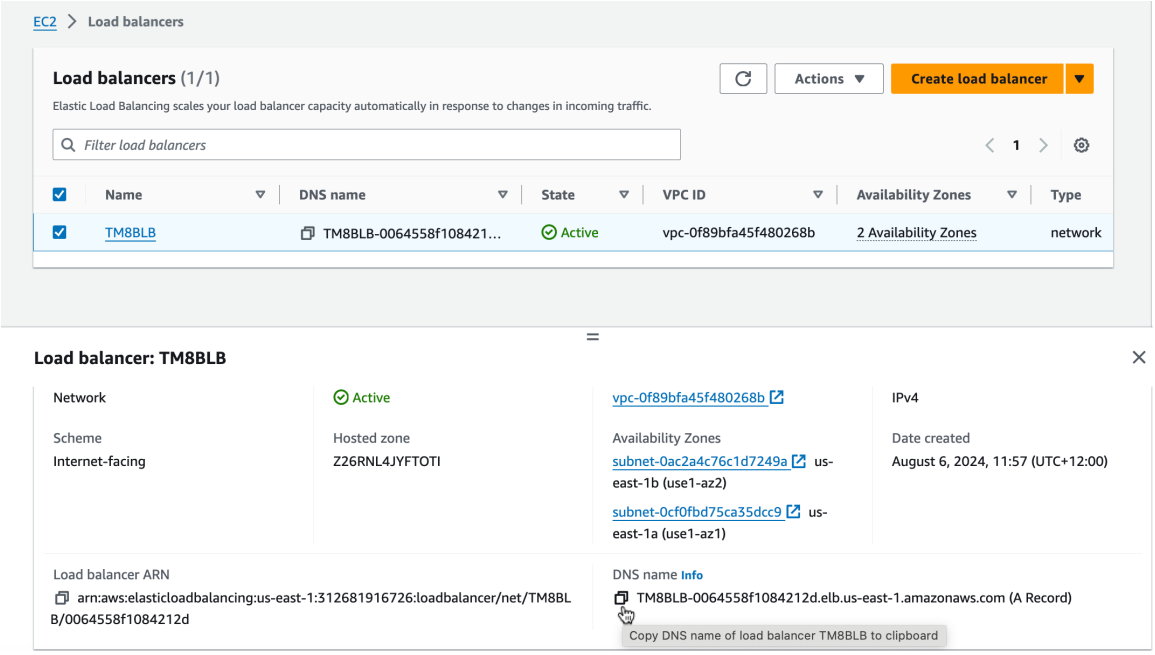
```
~$java Magic8BallClient 18.206.71.35 32000 "Can I haz chezburger"
Magic 8 Ball says: My reply is no (192.168.0.227)
~$java Magic8BallClient 3.86.52.155 32000 "Can I haz chezburger"
Magic 8 Ball says: Better not tell you now (192.168.1.226)
~$
```

3. **Once** everything has started up and passed the various checks, and we're convinced that the instances made by the autoscaler and your template are all working correctly - we can test your load balancer by selecting load balancers from the left bar and selecting your LB.

   The important thing here is that you will use only one IP address to contact your Magic 8 Ball service, the idea is to hide which instance you're talking to, you only need to know the DNS name of the load balancer, therefore it won't matter if any IP addresses change at any time, in response to instances starting, dying or simply being moved.



4. Copy the DNS name



5. Use it with your client, this is mine and you can see by the internal IP addresses, that both instances, in the two different regions have been transparently contacted using the same DNS name:

```
~$java Magic8BallClient TM8BLB-0064558f1084212d.elb.us-east-1.amazonaws.com 3200
0 "Can I haz chezburger"
Magic 8 Ball says: Reply hazy, try again (192.168.0.227)
~$java Magic8BallClient TM8BLB-0064558f1084212d.elb.us-east-1.amazonaws.com 3200
0 "Can I haz chezburger"
Magic 8 Ball says: It is certain (192.168.1.226)
~$
```

6. Just be wary in the lab - It can take a **long long** time to update the first time, as DNS is involved - also, you want to be certain your instances are healthy, and you can find this in the Target Group menu.

7. Terminate one instance, or two, or three. Conduct an experiment or several.

   QUESTION 1: What happens? It can take a long time - so be patient.

8. Adjust your autoscale settings - you will notice a new instance created if you increase your minimum (assuming you started with 1).

   QUESTION 2: What happens if you decrease the maximum or preferred?

9. Monitoring. Look under the Target group and select the monitoring tab. You should be able to see the effects of steps 7 and 8. I suggest you screenshot this and add commentary about what you did and how this impacted what the monitoring saw. This is experimental, so document lots.

10. Do likewise for the (different) monitoring tab under Load Balancers. I suggest you screenshot this and add commentary.

    QUESTION 3: Monitoring evaluation: look at the outputs of 9 and 10. Compare and contrast what happened based on how you 'poked' at the settings and configurations and bring this explanation back into your understanding on the Load Balancing architecture.

Submission

Attribute any code sources/AI etc. in your code comments. If we have any questions about the submission, you may be asked to complete a demo.

- Submit your server and client source code, with everything needed to compile and execute it:
    - your .java or .py source files - and ensure you have named as per section 3:
        - Magic8BallClient.java or Magic8BallClient.py
        - Magic8BallServer.java or Magic8BallServer.py
    - Do **not** submit .jar or class files.
- Submit and comment on many many 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)
- Finally: do not terminate your instance - you may be able to use 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 on the instance unless you terminate, and if you do terminate - then it is back to square one for you!
- Don't forget to delete your Loadbalancer - once you're done. You will be charged if you leave it up.