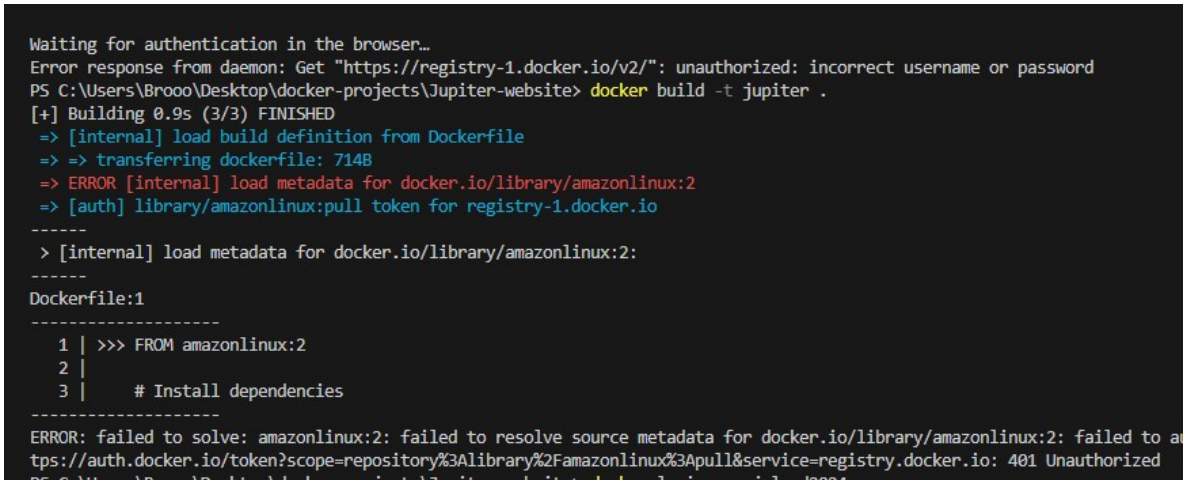
Deploy a Static Web App on AWS with Docker, Amazon ECR, and ECS

Deploying a static web app on AWS using Docker, Amazon Elastic Container Registry (ECR), and Amazon Elastic Container Service (ECS) provides scalability, reliability, and efficiency in managing your application.

When inputting the below docker syntax to build the container image, I came across the following error ‘failed to authorise’. After a bit of research, I noticed that my docker credentials in VSCode were not up to date.

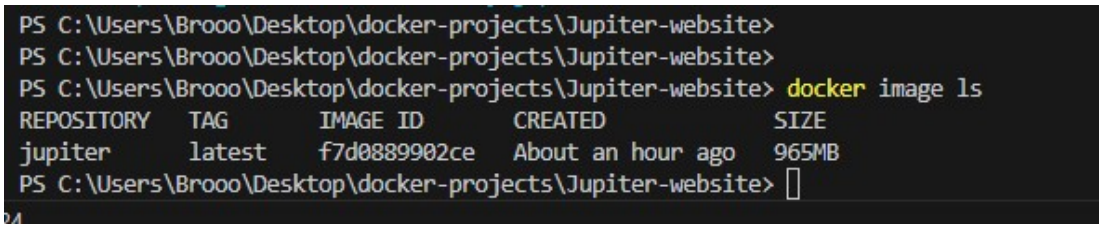


To resolve this, I had to log into docker via the docker extension in VSCODE. Please do the following:

- In the Docker extension panel, look for the **"Registries"** section. It’s typically near the bottom of the Docker sidebar.
- Hover over the **"Registries"** heading and click the **plus (+) icon** that appears.
- Select **"Docker Hub"** from the dropdown menu.
- Enter your Docker Hub **username** and press Enter.
- Enter your Docker Hub **password**

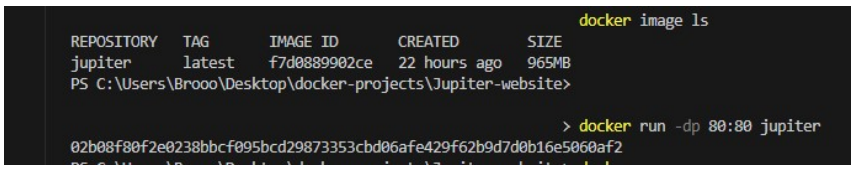
Now your credentials have been updated you can rerun the following command ‘docker build -t Jupiter .’ You should have the following output:

Based on the dockerfile I put together you can see all the steps the file went through. To check the image created you can type the following command in the terminal ‘docker image ls’.

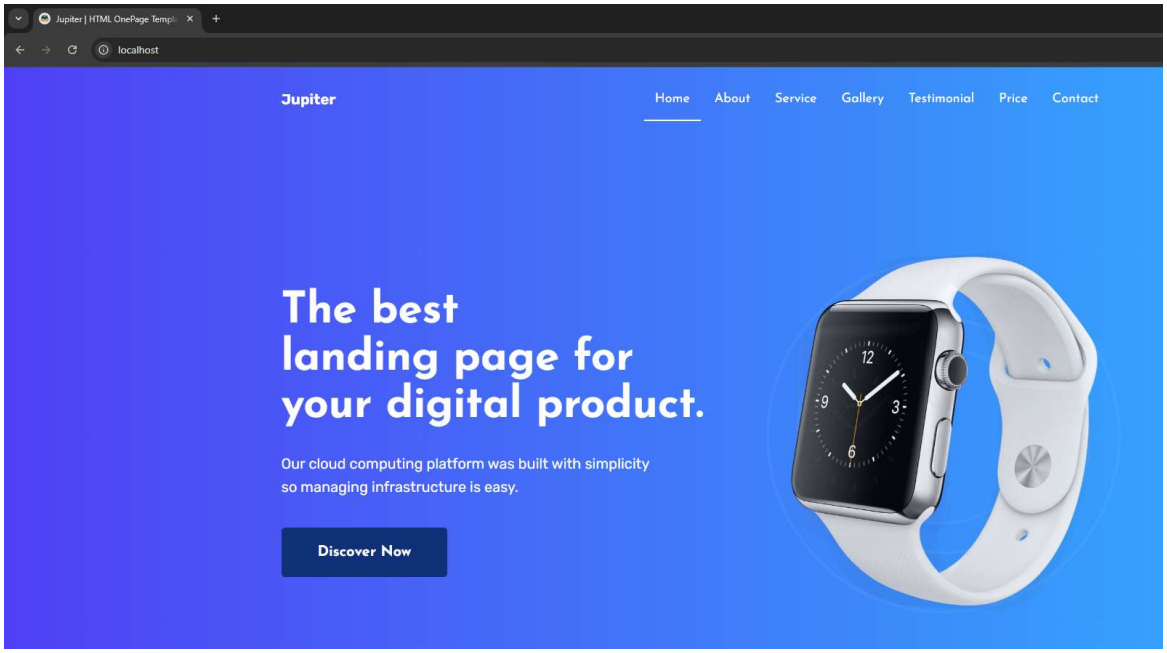


Now I’ve have build the docker image, we are going to run image as a container via post 80 which will allow us to access the site via the internet.

In the VSCode terminal write **‘docker run -dp 80:80 jupiter’**. Docker run will create and run the container from the image we created earlier. **‘-d’** runs the container in detached mode which means the container runs in the background instead of occupying the terminal session. **‘-p 80:80’** maps the containers to port 80 to the hosts port 80 (your local machine). **‘jupiter’** this is the name of the docker image we created earlier and docker will run this image as a container.



Based on our recent command to run the docker image, we should be able to access the Jupiter website via ‘http://localhost:80’. Write this in your search engine and you should see the below website.



GREAT! Your site is now up and running via the localhost. To stop the container you can write the following commands in the VSCode terminal.

1. Docker ps – to show the current running containers.

```
PS C:\Users\Brooo\Desktop\docker-projects\Jupiter-website> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                NAMES
584493cb228c   jupiter   "/usr/sbin/httpd -D …"   4 minutes ago   Up 4 minutes   0.0.0.0:80->80/tcp   youthful_ardinghelli
```

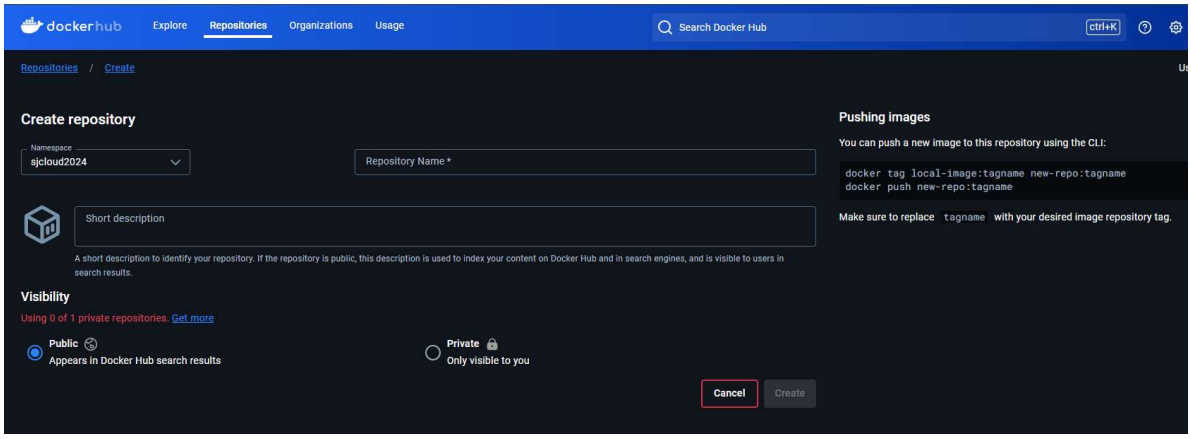
As you can see, I have one container running which is the Jupiter container.

2. Docker stop <container ID> - this command will stop the container.

```
PS C:\Users\Brooo\Desktop\docker-projects\Jupiter-website> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                NAMES
584493cb228c   jupiter   "/usr/sbin/httpd -D …"   4 minutes ago   Up 4 minutes   0.0.0.0:80->80/tcp   youthful_ardinghelli
PS C:\Users\Brooo\Desktop\docker-projects\Jupiter-website> docker stop 584493cb228c
584493cb228c
PS C:\Users\Brooo\Desktop\docker-projects\Jupiter-website>
```

Create a Repository in Docker Hub

To create a repository in docker hub, you will need to go to ‘hub.docker.com’ in your search engine. Once you are logged in, go your repository and click create repository. Give your repo a name and description, followed by making the visibility public and click create.



Congrats! You have now created a docker repository.

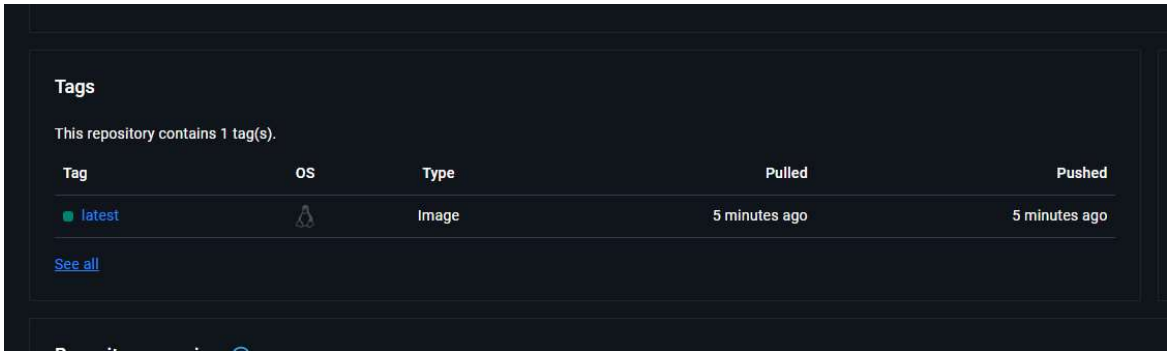
Push the Image to the Docker Hub Repository

In this section, I'm going to change the name of our image and push the docker image to the docker repository we created in docker hub. To do this you will need to do the following:

1. Log into docker via the VSCode terminal – this is to ensure I'm logged in and the image will be pushed to docker. Write 'docker -u <yourusername>' and write your password. Once completed you should get a response saying, 'login succeeded'.

```
PS C:\Users\Brooo\Desktop\docker-projects\Jupiter-website> docker login -u sjcloud2024
Password:
Login Succeeded
```

2. Changing the name of our image – to do this we you will need to enter the following command into the VSCode terminal: 'docker tag jupiter <yourusername/Jupiter>. Once entered you can run the 'docker image ls' command to see the changes.
3. Pushing the image to your docker repo – run the command 'docker push sjcloud2024/jupiter' in the terminal. Once you run and all checks have been pushed, you can go to your repo in docker and you should see the image tagged as 'latest'.



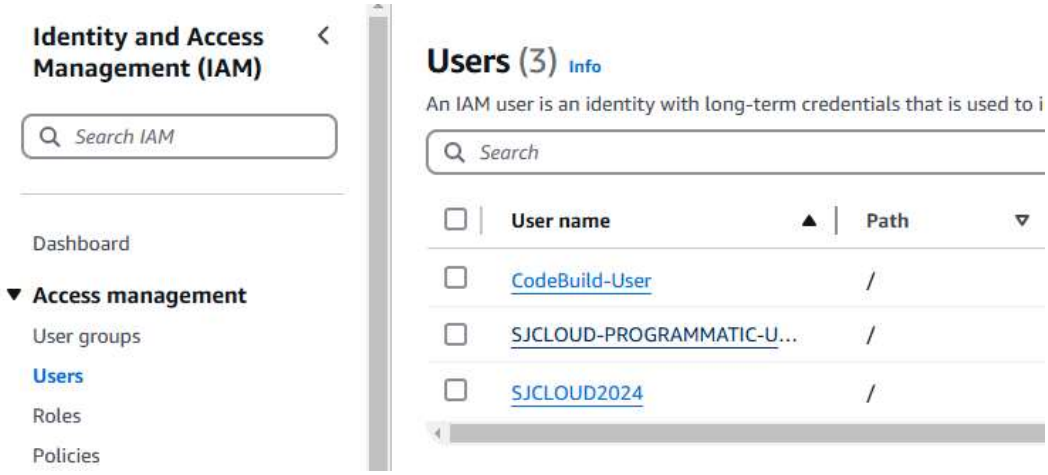
Install the AWS Command Line (CLI) on windows computer and create an IAM user

As this documentation is for showing the deployment of a static website on AWS with docker, I will not be going through the process of installing the AWS CLI and creating an IAM user with admin access.

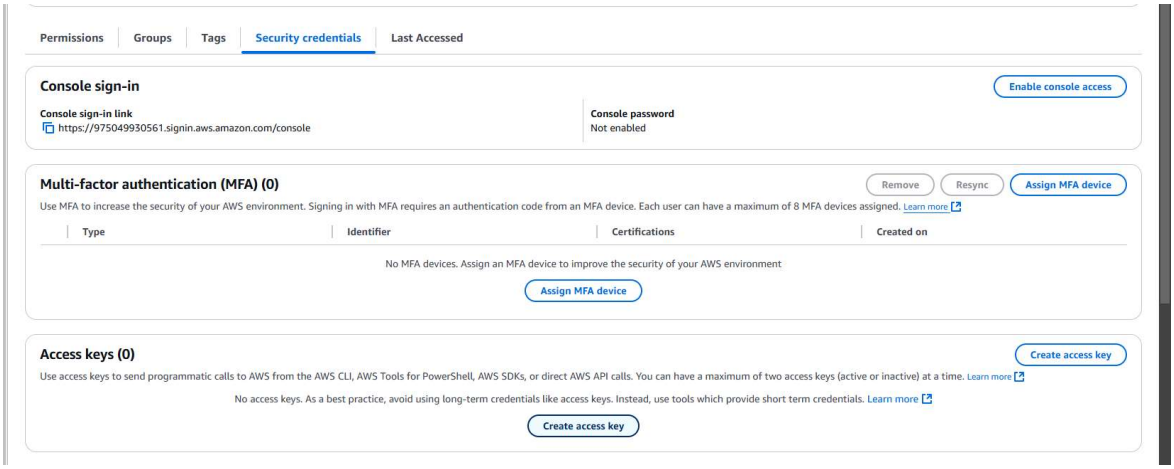
Create Access Key for an IAM User

In this section we will go through the process of creating an access key for the IAM user created earlier.

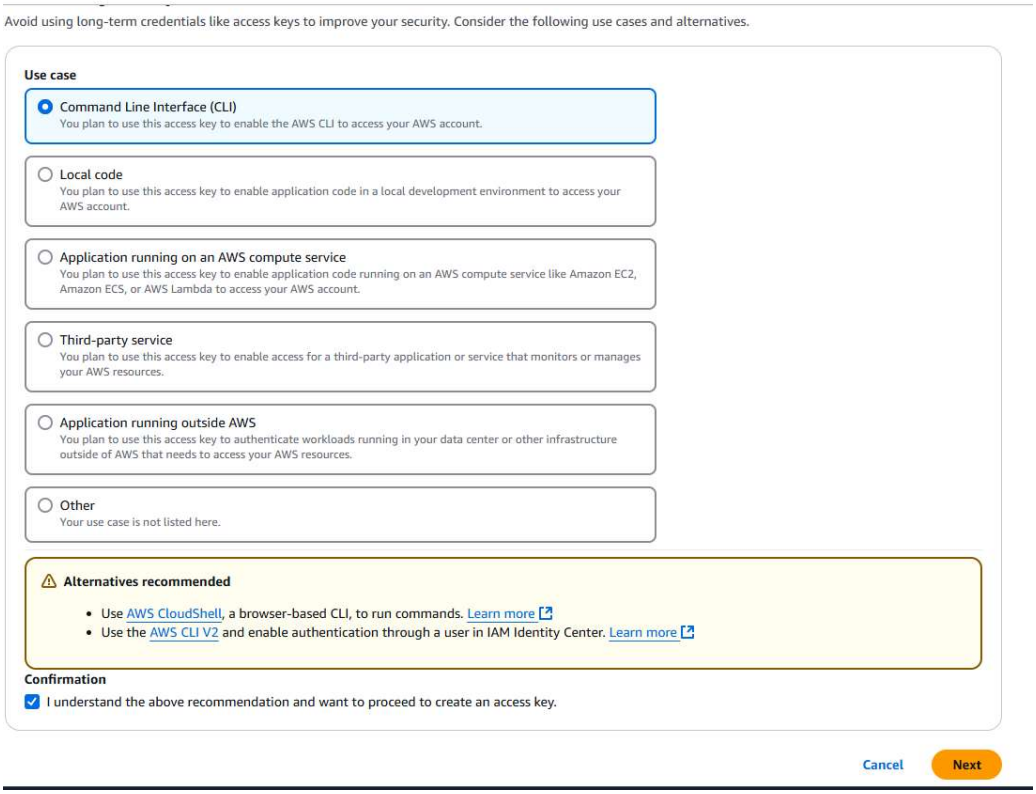
1. Go to users within the AWS management console and click on the user your created for programmatic access.



2. Once within the user page, click on security credentials and scroll down to access keys and click create access key.



3. Click Command line interface (CLI) and click the confirmation below.

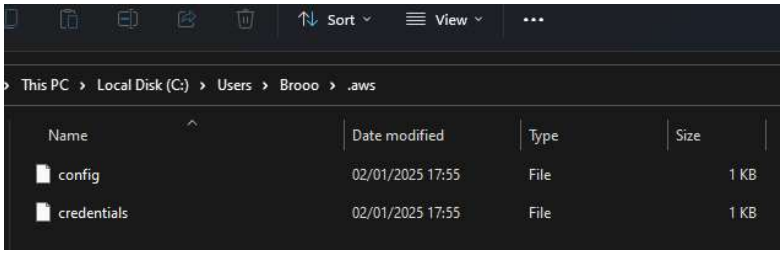


4. The next page will ask you to provide a tag description for your access. We will leave this blank and click next.
5. GREAT! You have now created your access keys for your user. Save your access key and secret access key as you will need them later. IMPORTANT note your access keys will only be shown once, and you will not be able to see them again. Make sure you save them down.

Create a profile and configure access keys for programmatic access

1. Open the command prompt and write ‘aws configure’ and press enter. You will get a prompt to enter your access key ID and secret access key. Use the access keys created from the previous step. You will also be asked to provide a default region name. In this example I used ‘us-east-1’. The following prompt will ask for a default output format and for this you can leave blank and press enter.

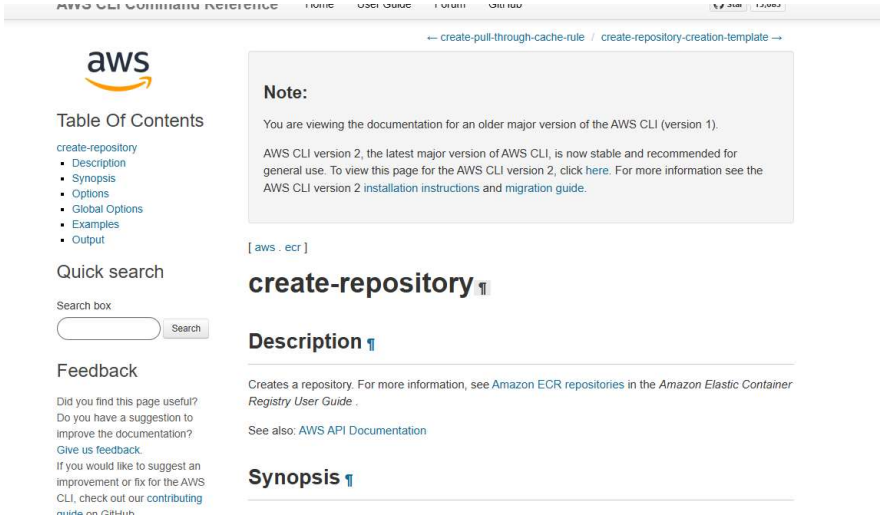
2. To check your access keys have been stored, go to your ‘C drive’, click ‘users’, then click on your computer user name, followed by ‘.aws’ and you will be able to see your credentials and config file saved in this folder.



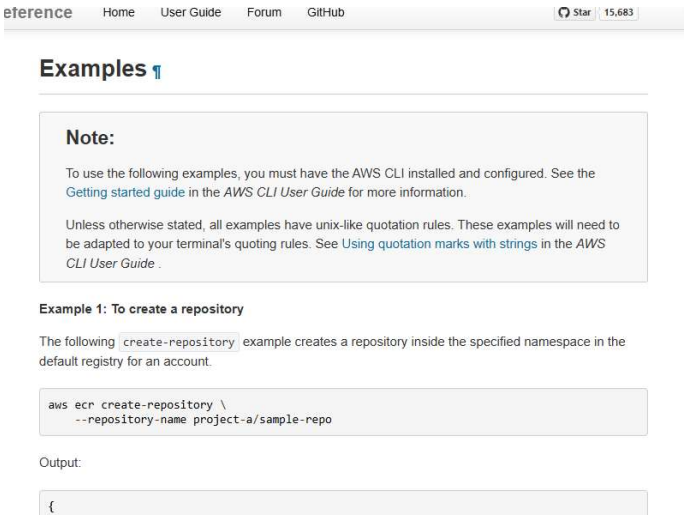
Create an Amazon ECR Repository to Store your Image

In this section we are going to create an ECR repository and store our image in the repo.

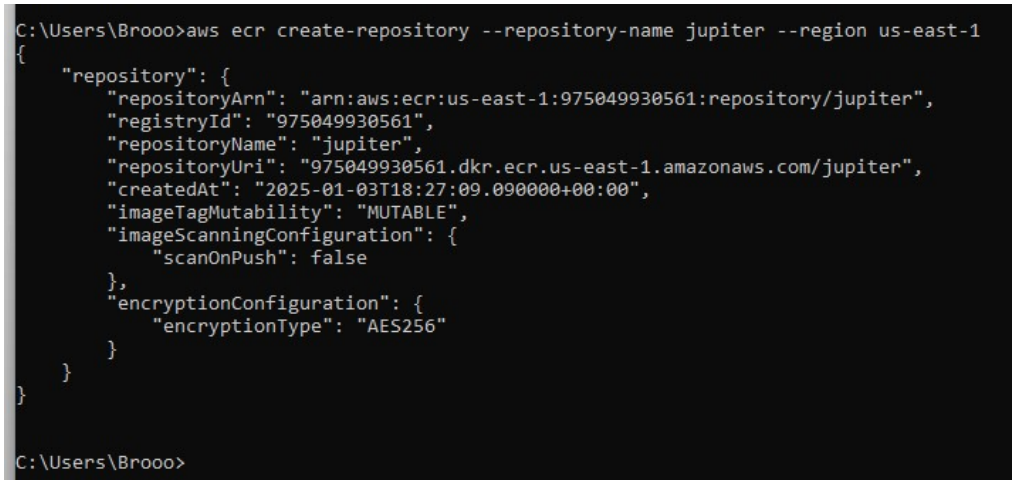
1. Got to google and write ‘ aws cli create ecr repository’. Click on the first link which says aws documentation. You should see the below page.



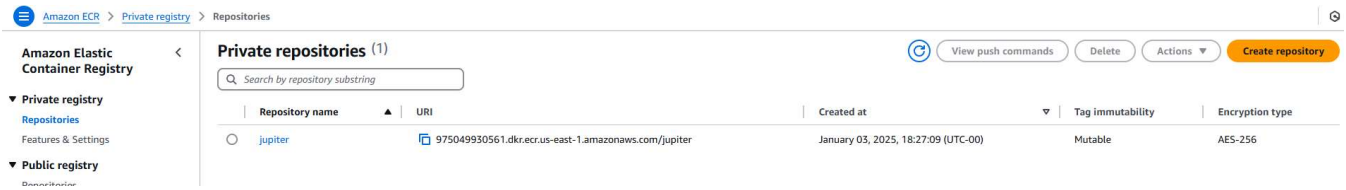
2. Scroll down to ‘examples’ and the first example will show the aws cli command to create a repository.



3. Copy the cli command and paste it into your computer CMD. I changed the name of the repo and added a region as I want the repo to be in ‘us-east-1’. See below for reference: (aws ecr create-repository --repository-name jupiter --region us-east-1)



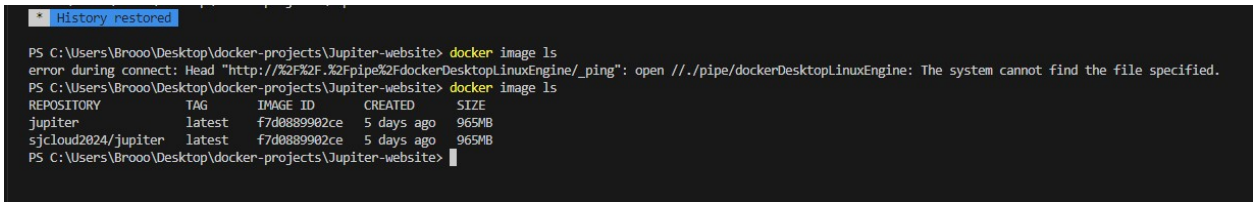
4. Next you will need to save the output and store this in your notepad, as we will need this information for the next step.
5. Finally, we can go to the aws console to see if our ECR has been created. In the management console, search ‘ECR’ and you should see the repository we created earlier under private.



Push the Image to your ECR Repository

Now we have created our repository in ECR, we will push our image to the ECR repository we created in aws.

1. Go to VSCode and open the terminal. Make sure you are in the Jupiter-website directory. Next you will need to write ‘docker image ls’ to list all the images within docker. The first image is the image we rename and the second image is the image we originally created.



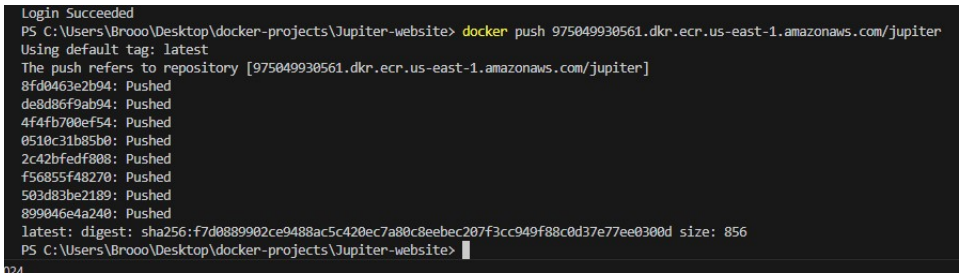
Note: when I first enter the command ‘docker image ls’ I got an error. This was because I was not signed into my docker desktop.

2. Next, we need to tag our image to push the image to ECR. In the terminal write ‘docker tag <repositoryUri>’ which can be found in the information you saved in your notepad from the previous section and press enter.

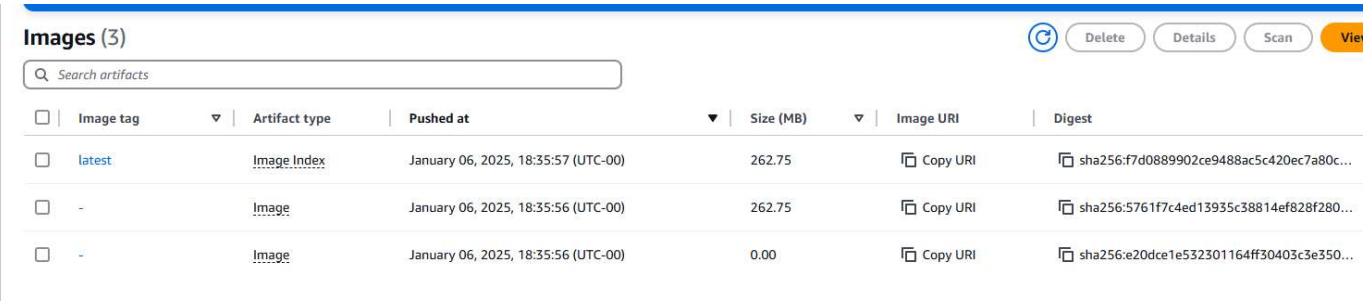
3. Now you have submitted the tag via the terminal. Write ‘docker image ls’ to see the taged image we created.
4. Next, we will need to log into aws via the VSCode terminal before we will be able to push the image to ECR. To do this you will need to enter the following command in the terminal – ‘aws ecr get-login-password |docker login --username AWS –password

stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com’. You will need to change the account id to your account id and this is the first 12 digits of your URI. You will also need to change the region.

5. Now we have logged in, we can push the container image to AWS. In the terminal write the following command ‘docker push <URI from aws console> and press enter.



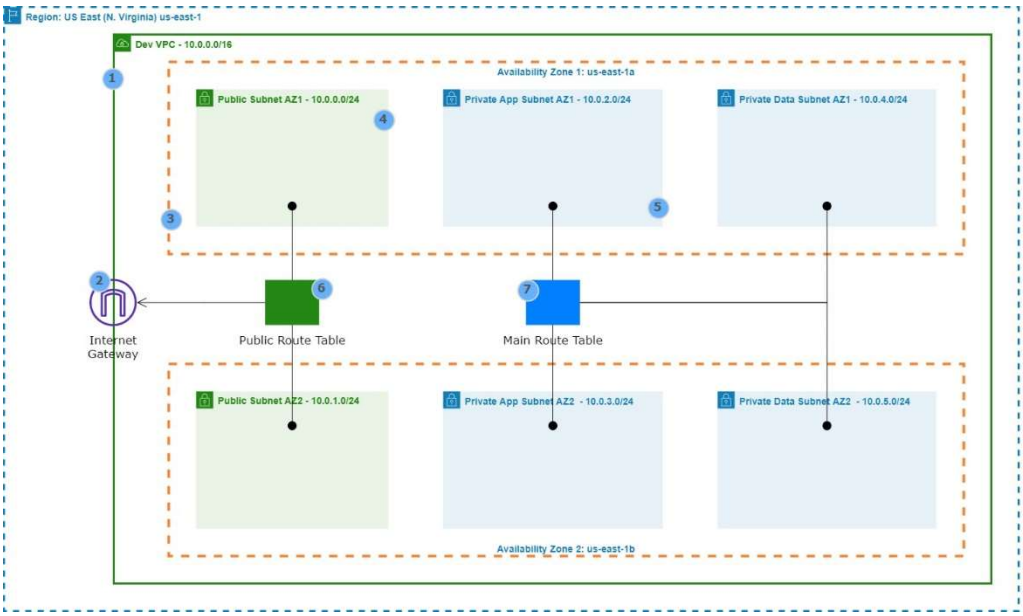
6. Let’s check our container image has been pushed to our ECR repository by going to the management console. Make sure to fresh the page, click on the Jupiter repository and you should see your image and the tag for that image is called ‘latest’.



Build a Three-Tier AWS Network VPC

To complete this project, we will start by a three tier VPC architecture. In a three tier vpc architecture your vpc is divided into the following tiers:

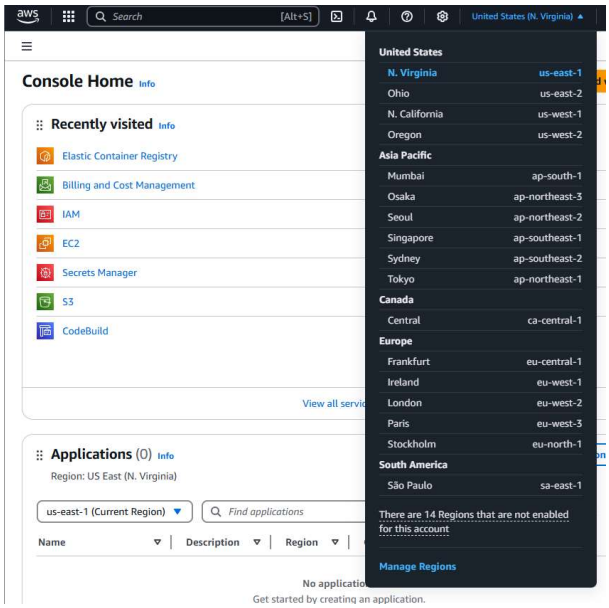
1. Tier 1 – Public subnet: this will host resources such as NAT gateway, load balancers and bastion host.
2. Tier 2 – Private subnet: this is the subnet that will host our web servers e.g. EC2 instances
3. Tier 3 – private subnet: this subnet will host our database



All 3 subnets will be duplicated across multiple availability zones for high availability and fault tolerance. Finally, we'll create an internet gateway and route table to allow the recourses in our vpc to access the internet.

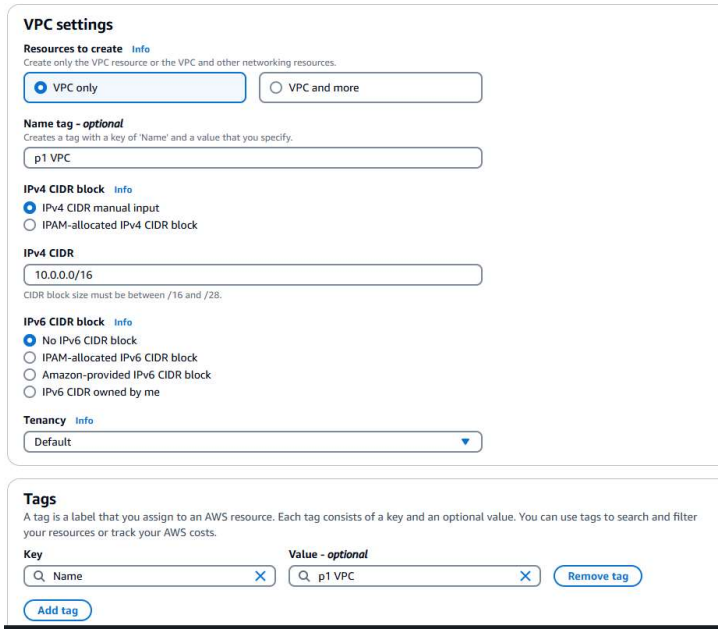
Creating a VPC

The first thing you will want to do is select the region you want your vpc to be places. In this example I will be using ‘us-east-1’.

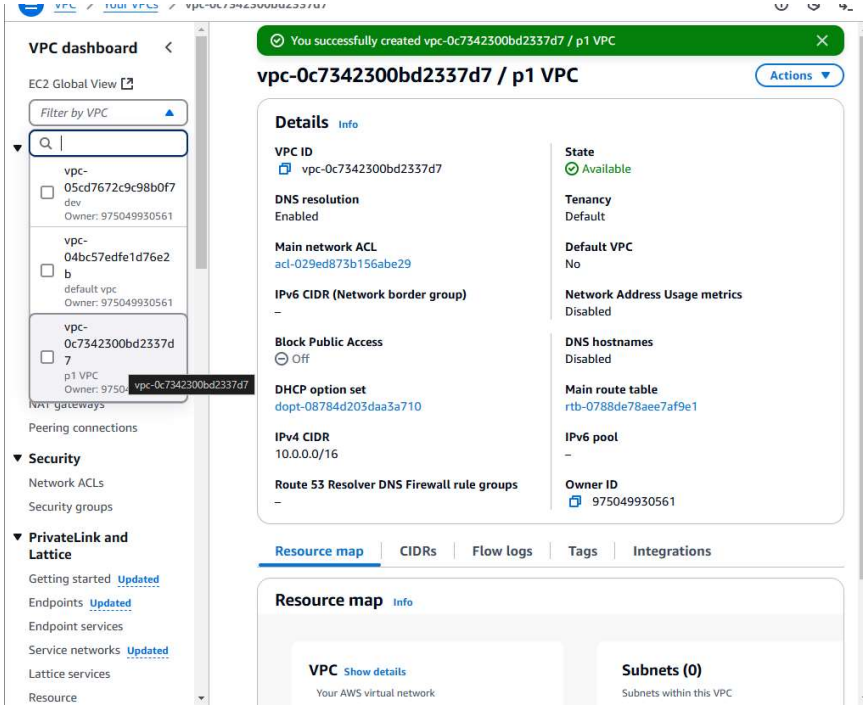


Once your region has been selected, navigate to the vpc dashboard by using the search bar. Click vpc then click create vpc.

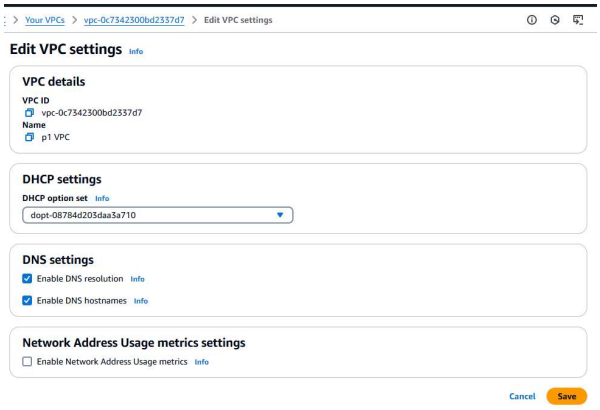
Give your vpc a tag name and input your chosen IPV4 CIDR block. In this example I have chosen 10.0.0.0/16. The next three setting can be left as default and click create vpc.



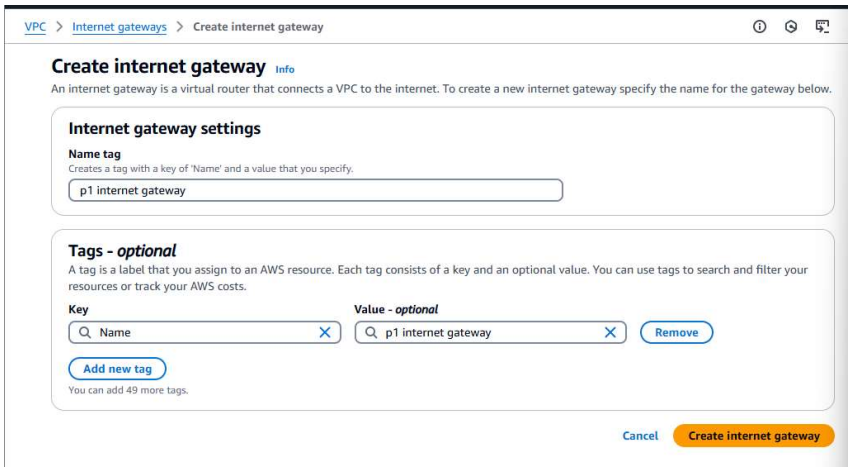
Great! You have now created your vpc. Next, we’ll be enabling DNS host name in our vpc. To do this first you will need to filter by your vpc.



Then click actions, edit VPC setting and then enable DNS hostnames.

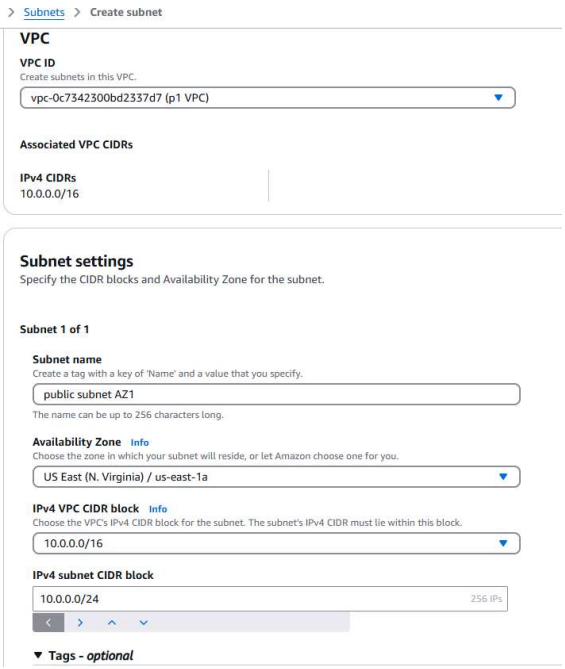


The next step is to create an internet gateway for our vpc. Make sure your vpc is still selected via the filter on the left hand side. On the left hand side under ‘virtual private cloud’ click internet gateway, then click create internet gateway. You will need to give your internet gateway a name. In this example I have called my internet gateway ‘p1 internet gateway’.

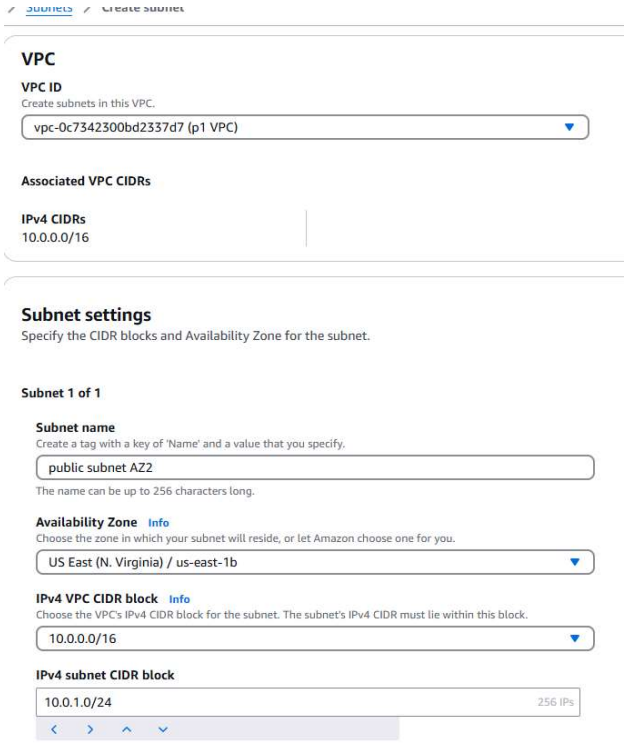


In this next step we will need to attach our internet gateway to our vpc. Click actions then ‘attache to vpc’. At this point you should be able to search for your vpc which was created earlier. In my case, I can see my vpc called ‘p1 vpc’. Note to remember: you can only attach one internet gateway to one vpc.

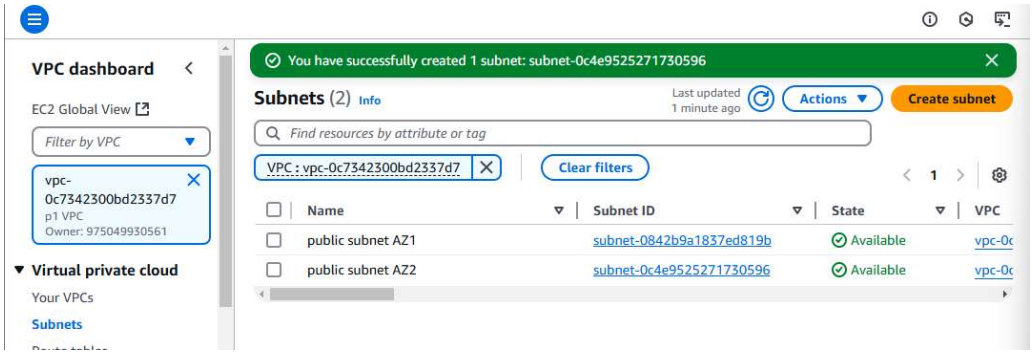
Now we are going to we are going to create our two public subnets in our two availability zones. On the left hand side make sure to filter by you new vpc and then below click on subnets. Next click ‘create subnet’. Make sure to pick the vpc you created earlier. In my case my vpc is called ‘p1 VPC’. Under subnet settings you will need to give your subnet a name and in our reference architecture our first subnet is called ‘public subnet AZ1’. The availability zone will be ‘us-east-1a’ and the subnet CIDR block will be ’10.0.0.0/24’. Leave all tag options as default and click create subnet.



To create you second subnet, follow the same steps as above however, the availability zone will be ‘us-east-1b’ and the CIDR block will be ’10.0.1.0/24’. See below for reference.

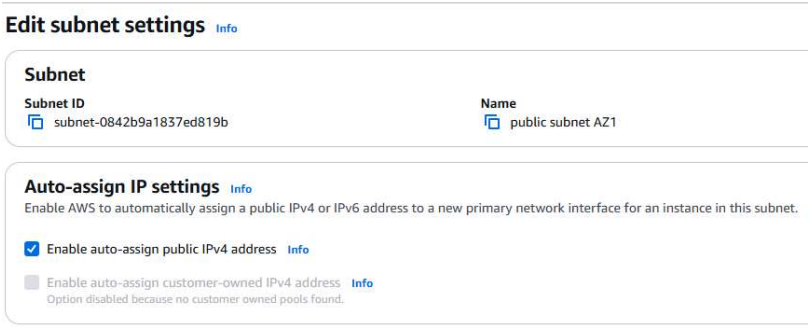


To see the new public subnets you created, you will need to clear all filters and filter by your chosen VPC and then you will see the two public subnets you created.



Next this we are going to do is enable to the auto assign the IP settings. This means any time we launch an EC2 instance in the public subnets we created, the EC2 instances will be assigned an IPV4 address.

To do this, select your first public subnet and click actions at the top then click edit subnet settings. In the next page under ‘auto-assign IP setting’ tick the box enable auto-assign public IPV4 address. Then press save.

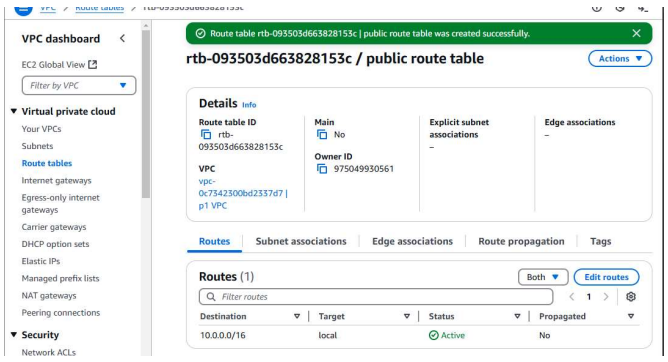


Make sure to do the same actions for your second public subnet.

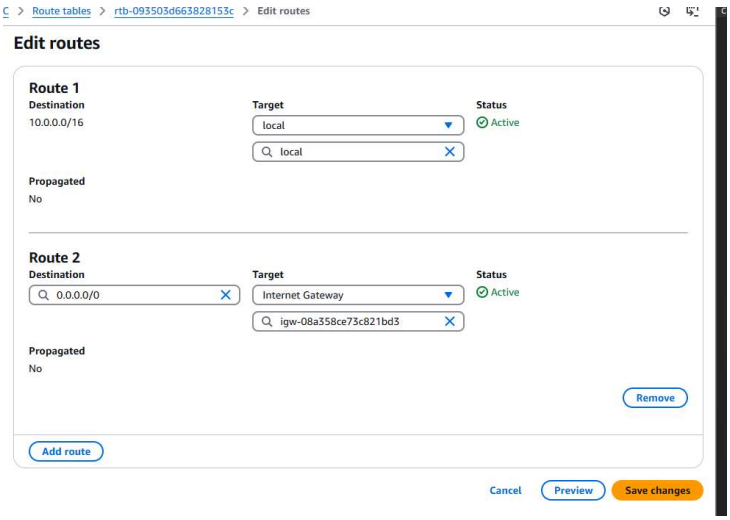
Creating a route table

To create a route table, click on route tables in the left-hand side of the management console, also make sure your filter is still filtered by your chosen vpc. You will notice that you have one route table already in your list. This table was created when we created the vpc earlier. Note that this table is the main route table and is private by default.

Click create route table, give you route table a name and select the your vpc from the drop down list, then press create route table.

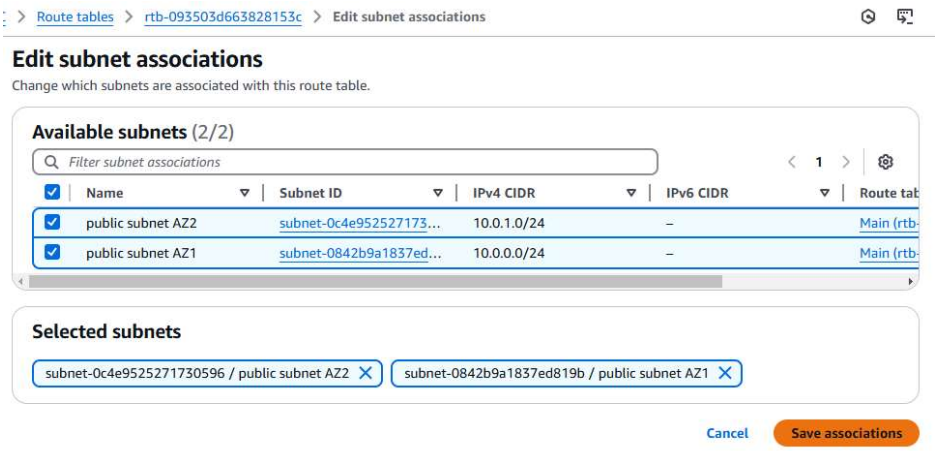


Great! You have created a public route table. Next, we will need to add a public route to our new route table. The public route will allow our route table to route traffic to the internet. To do this click ‘edit route’, add route and in the destination under route 2 type ‘0.0.0.0/0’. The target will be our internet gateway in the drop-down list and below this you should be able to select your internet gateway.



Associate the two public subnets with the public route table

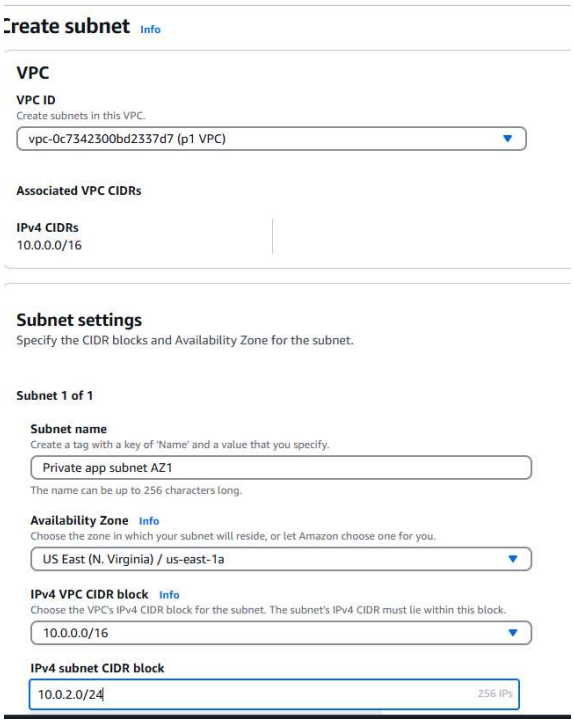
By associating our two public subnets with the public route table will ensure all our resources in the public subnets can access the internet or be accessed by it. To do this, within the public route table we created click ‘edit subnet associations’ within the explicit subnet associations. Once within the edit subnet association page, you will be able to see the public subnet your created. Tick both subnets and click ‘save associations.



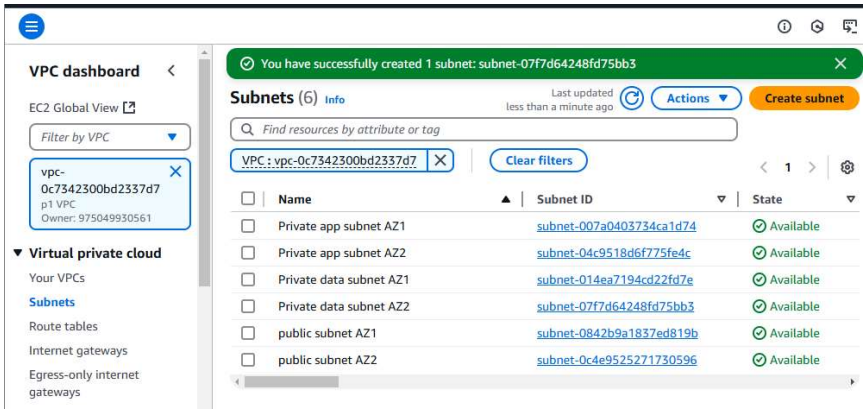
Now within the subnet associations tab, you should see both public subnets listed under the ‘explicit subnet associations’.

Creating private subnets

Navigate to the subnet tab on the left-hand side of the management console. Click create subnets and make sure you select your vpc ID. Under subnet settings we are going to call our first private subnet ‘ private app subnet AZ1’, this will be in the ‘us-east-1a’ availability zone, the IPV4 subnet CIDR block will be ’10.0.2.0/24’.



You will need to do this step three more times as we need four private subnets. The details for the private subnets are within the main architecture above. Once all subnets have been created, filter by your vpc and your subnet page should look like below.



You should have 6 subnets:

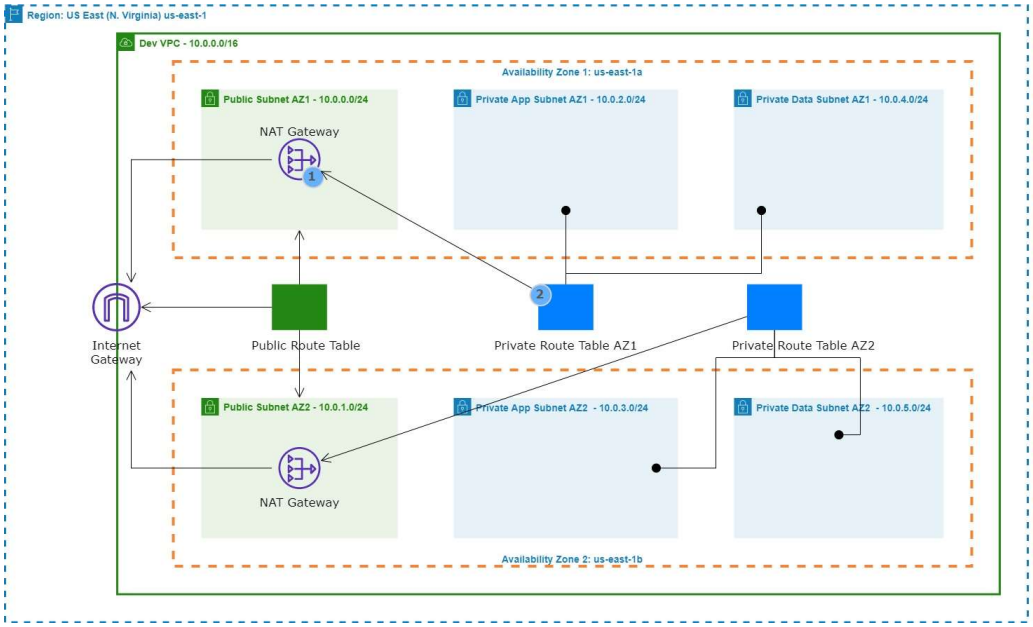
- Two private app subnets
- Two private data subnets
- Two public subnets

Create NAT Gateways

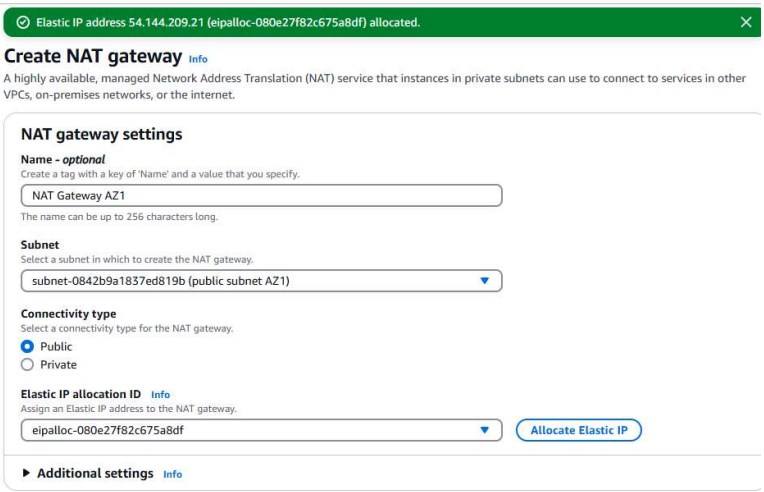
In the section we are going to create a NAT Gateway in the first and second availability zones using the below reference architecture. With reference to the below architecture, we will create a NAT gateway in the public subnet AZ1 and create a route table that we'll call 'private route table AZ1'.

Once we the route table has been created, we'll add route to the route table to route traffic to the internet through the NAT gateway. We'll also associate the route table to the private App subnet AZ1 and private Data subnet AZ1. This process will be replicated in the second availability zone.

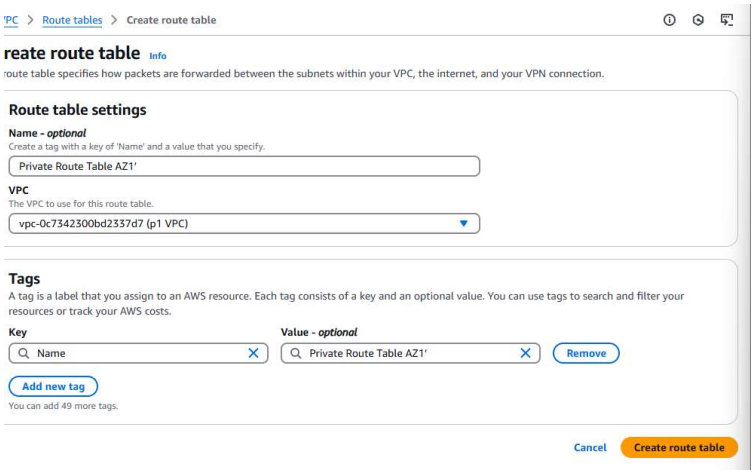
Note: A Nat Gateway enables instances in private subnets to access the internet or other AWS services without exposing those instances to incoming internet traffic.



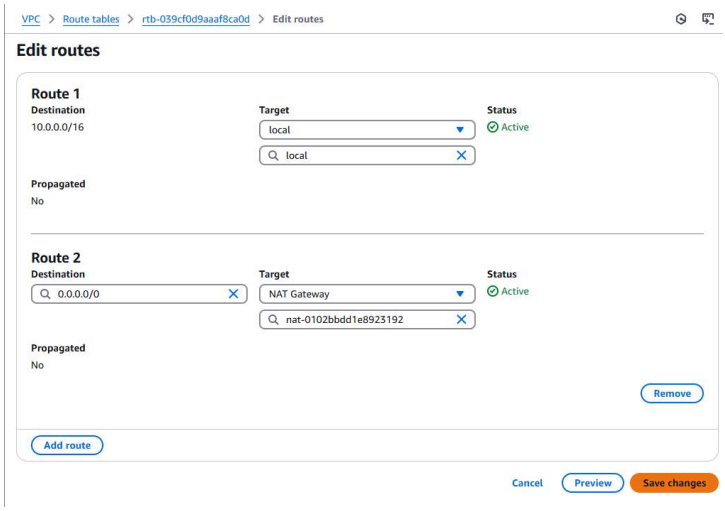
To create our first NAT gateway, navigate the vpc created earlier and on the left-hand side of the management console, you should be able to see 'NAT gateway'. Give your NAT Gateway a name, under subnet pick the public subnet we created and under elastic IP allocation ID click allocate elastic IP.



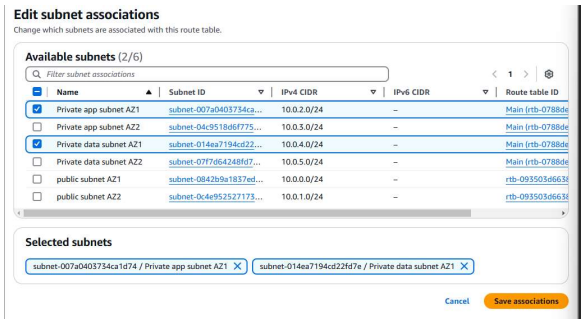
Great! You have created a NAT gateway. Next, we will create a route table, and we'll call this route table 'private route table AZ1'. Navigate to the route table page and click create route table. Give your table a name and select the correct vpc which should be the vpc we created earlier. Once this has been done, you can click create route table.



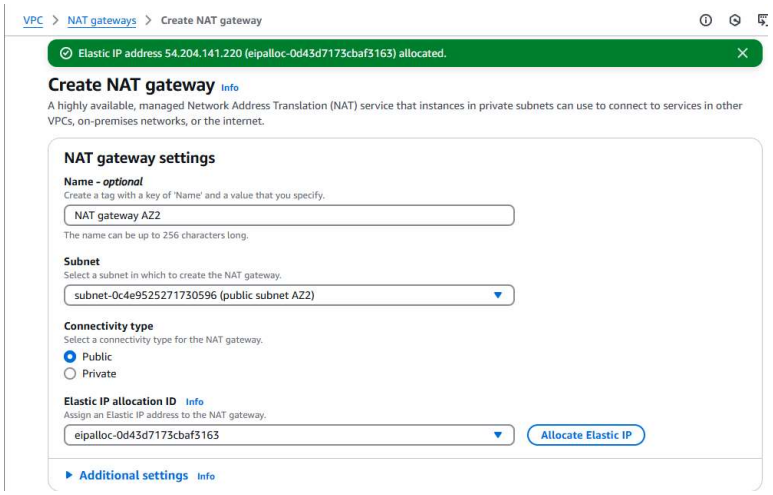
Now we're going to add a route to our private route table AZ1 to route traffic to the internet through the NAT gateway in the public subnet AZ1. To do this, click actions and then 'edit routes'. The destination will be '0.0.0.0/0 and the target will be 'NAT gateway' and you should be able to select the public NAT gateway AZ1 we created.



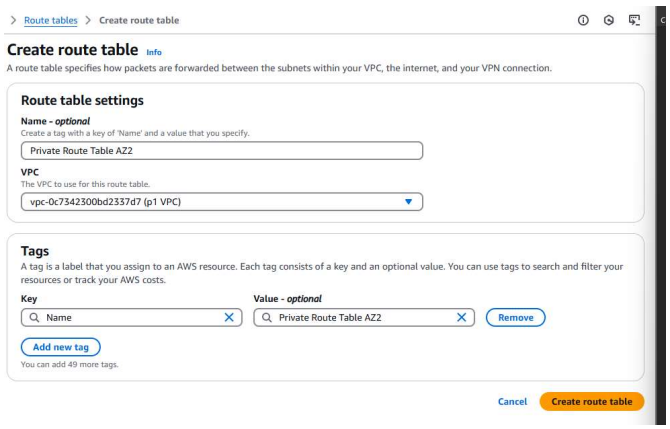
Next, we'll associate the private route table AZ1 to the private app subnet AZ1 and private data subnet AZ1. You will need to click 'subnet associations' and then under 'explicit subnet associations' click 'edit subnet associations'. Once in the edit subnet associations page, you will need to tick the private app subnet AZ1 and private data subnet AZ1 then click save associations.



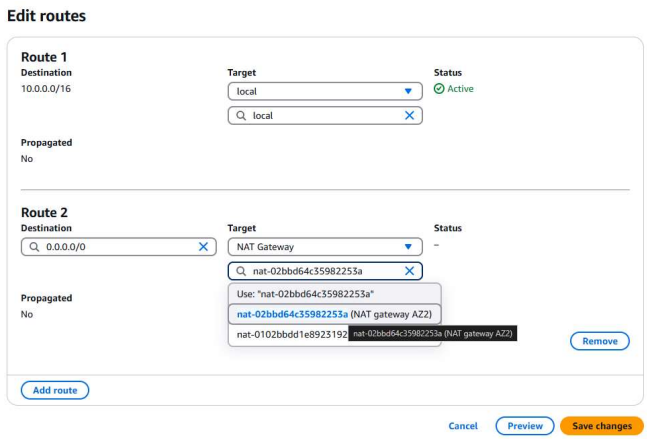
Now we will create the second NAT gateway in the public subnet AZ2. To do this please following the same steps we did to create the first NAT gateway.



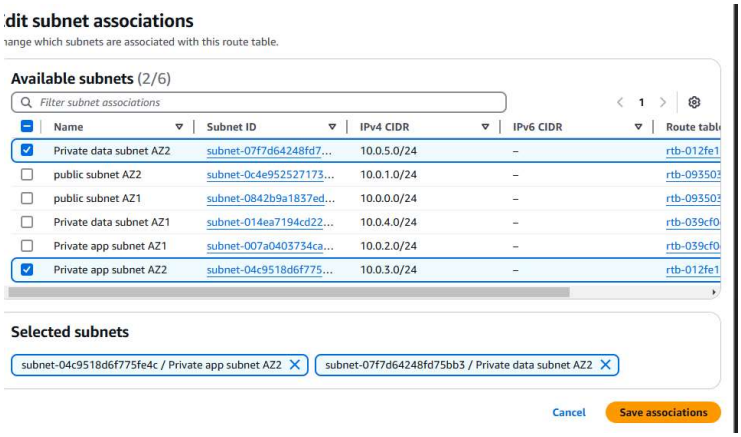
Next, we'll create the second private route table. This process will be the same as what we did to create the first private route table. Please refer to above steps.



Following the creation of the second NAT gateway and second private route table, we'll need to create a route to the second route table to route traffic to the NAT gateway in the public subnet AZ2. As mentioned above this will be the same process as before, therefore please refer to above steps. Note: this route will be linked to the NAT gateway in AZ2.



The final process is to associate the private route table to the private app subnet AZ2 and the private data subnet AZ2. This again will be the same steps we took above to create the associations for AZ1, however, this is new related to AZ2.



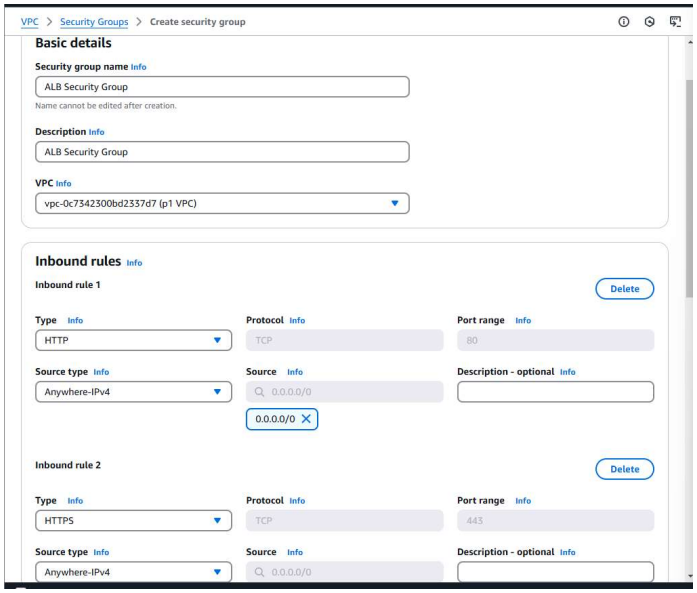
Create Security Group

In this section we are going to be creating two security groups:

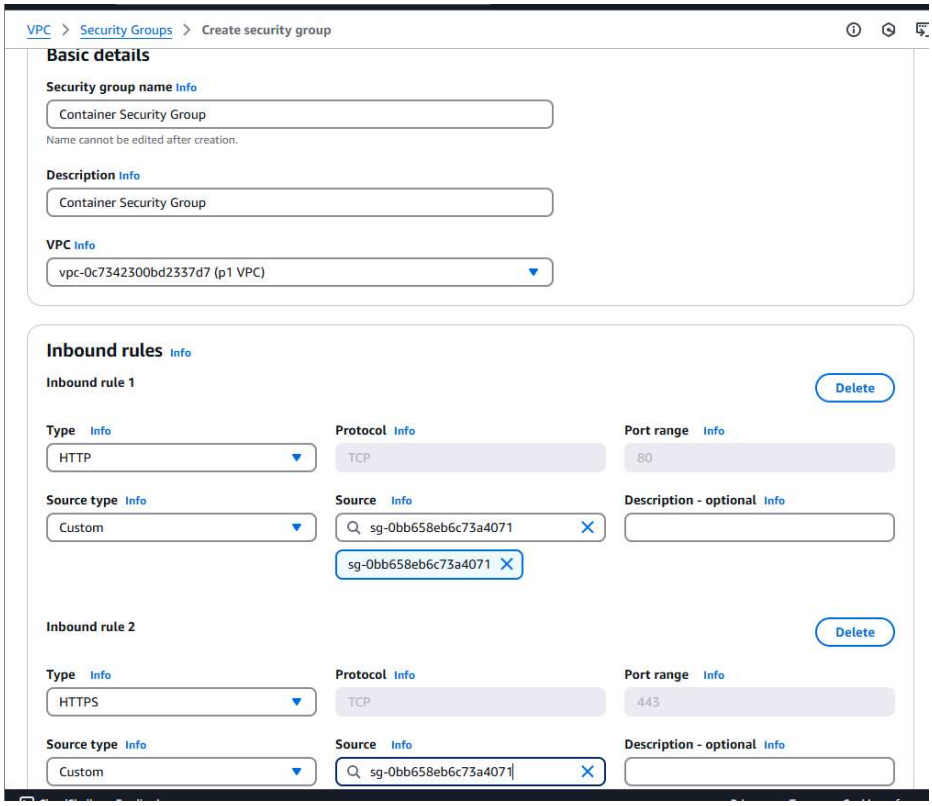
1. Application load balancer security group – open port 80 and 443 from anywhere
2. Container security group – opening port 80 and 443, source of traffic will be from the ALB security group

Navigate to the VPC dashboard in the management console. Then filter by the vpc we created earlier in this documentation. Once filtered scroll down on the left-hand side and click ‘security groups’. We’ll call our security group ‘ ALB security group’ and select our VPC from the dropdown list. We’ll also need to add two inbound rules:

1. Type: HTTP – Source: 0.0.0.0/0
2. Type: HTTPS – Source: 0.0.0.0/0



Next, we're going to create the container security group. This will be the same process has creating to ALB security group. However, the inbound rules will be a little different. The first inbound rule will be 'HTTP' on port and the source will be our ALB security group. The next inbound rule will be 'HTTPS' on port 443 and the source will also be the ALB security group. Once completed you can click create security group.

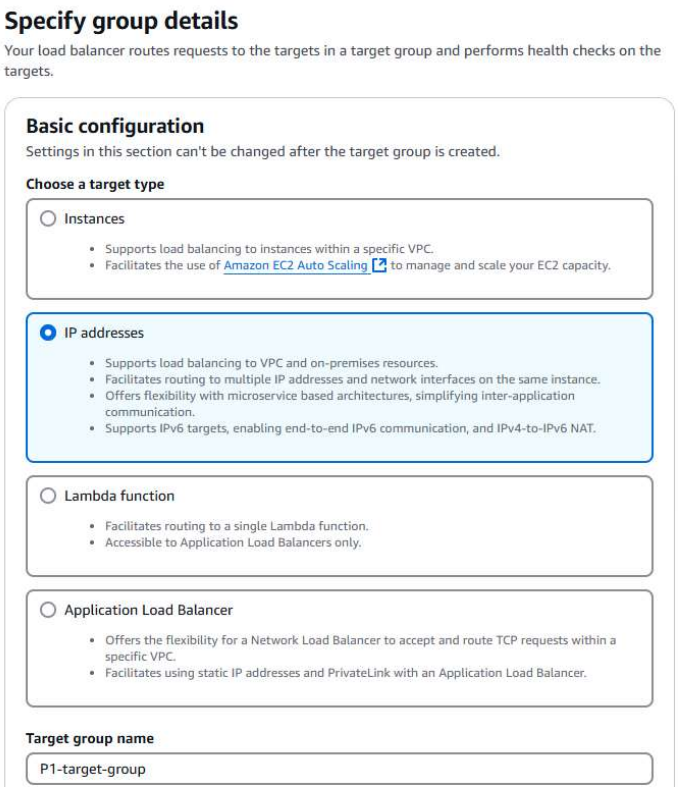


Creating an application load balancer

In this section, we'll be creating an application load balancer. This will be used to route traffic to the ECS task we'll create later.

Navigate to the EC2 section within the management console. Before creating the application load balancer, we'll need to create a target group and when the ECS creates our tasks it will add those tasks to the target group, then the application load balancer will route traffic to the tasks in the target group. On the left-hand side scroll down and under load balancing you should see 'target groups'.

1. Basic configuration will be 'IP addresses'
2. Give your target group a name of your choice. In my case I called my target group 'P1-target-group'

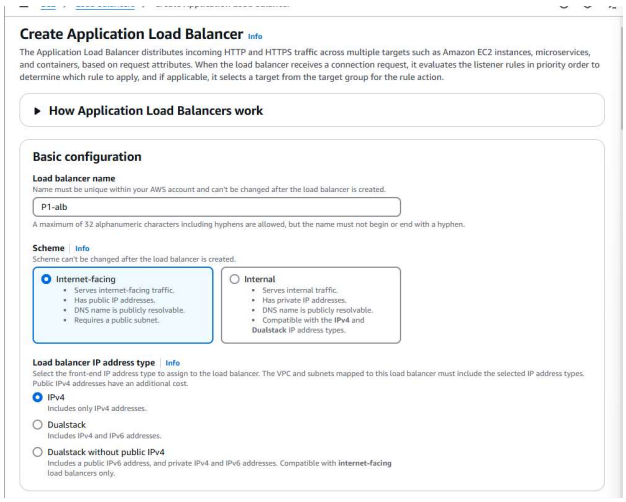


3. Under the VPC section select the VPC you created earlier and all other setting can remain as default.
4. On the following page which is the register target section make sure your VPC has been selected from the drop-down list. You can leave all other settings as default.

As you can see under the review targets, we have no targets as we have not yet created our ECS tasks. Once the ECS task has been created the ECS service will automatically add the targets to this section.

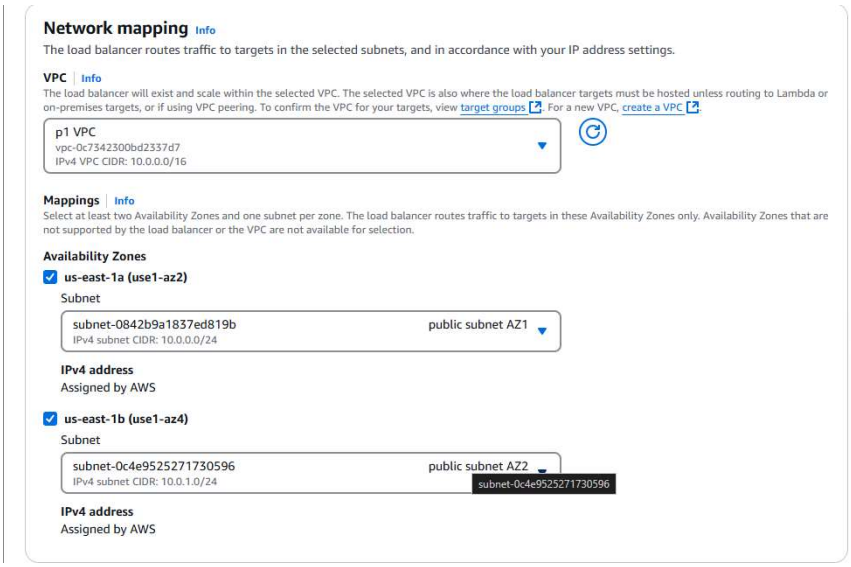
Now let's create our application load balancer!

Make sure you select application load balancer, give your alb a name and the scheme will be internet-facing.

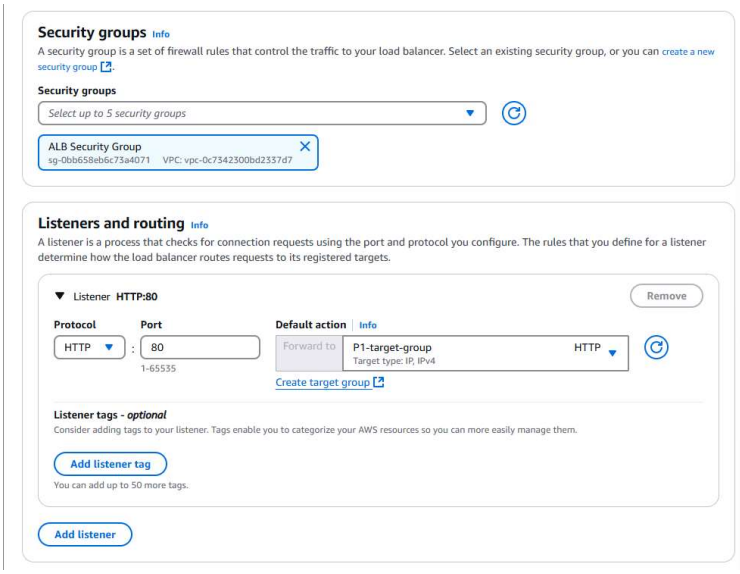


In the network mapping section, make sure you select your VPC from the drop down list. Under mapping select your two availability zones.

Note: your subnets should be your PUBLIC subnets and NOT your private subnets



Next remove the default security group and select the ALB security group from the dropdown list. The default action for listeners and routing will be the target group we created earlier.

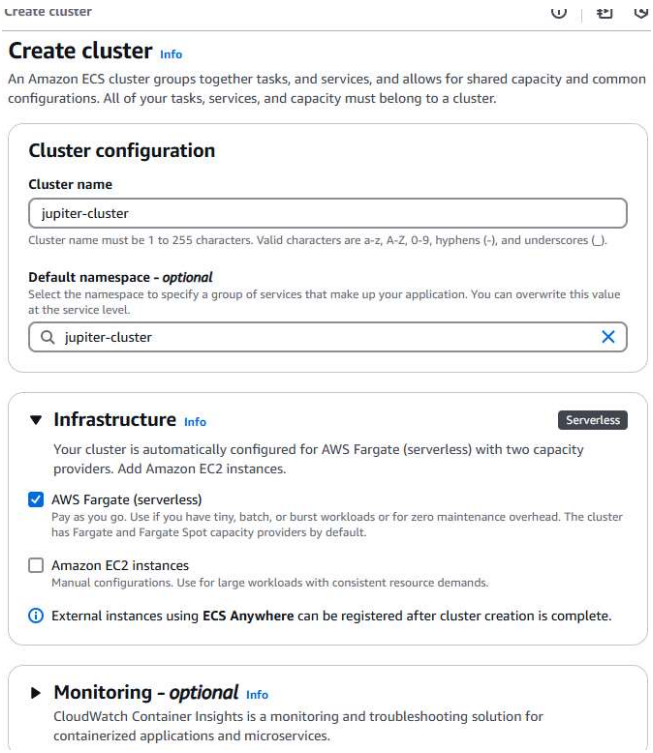


Once the above has been completed, you can scroll down and click create load balancer.

Create an ECS cluster

In the previous topic, we pushed our image to the ECR repository and now we are ready to run our ECS fargate container.

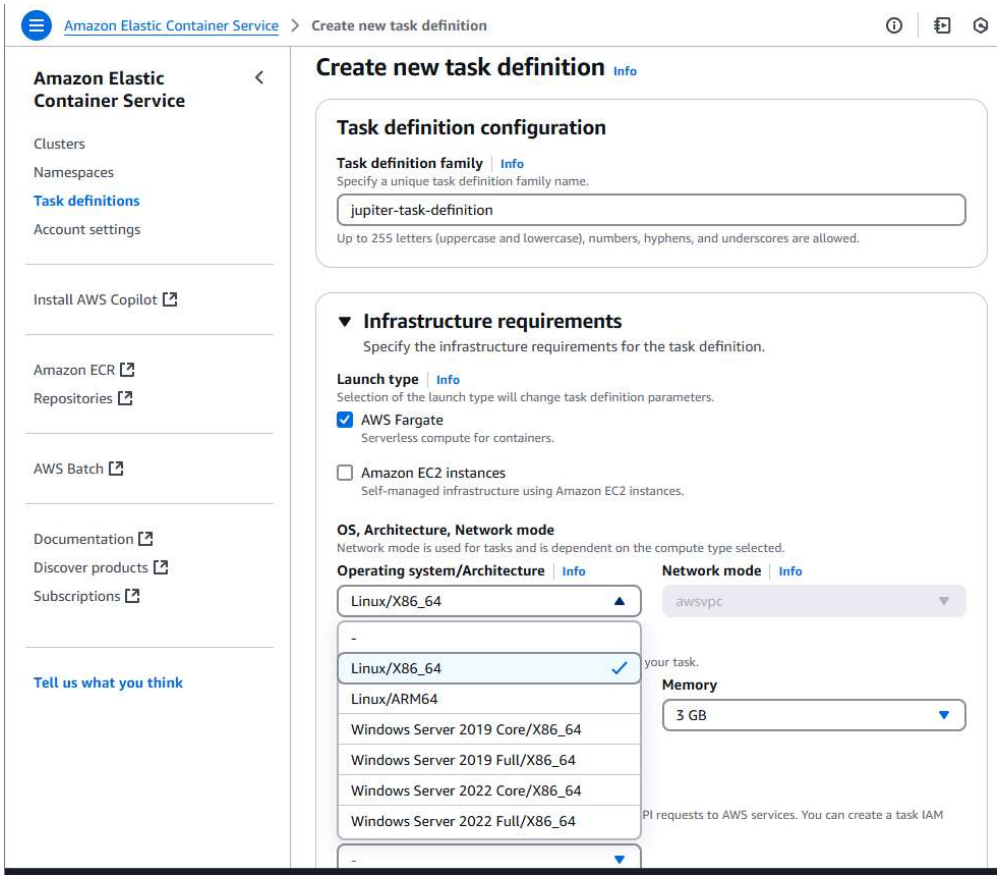
In the search bar, search for ECS and on the left had side select clusters. Within the create cluster page you will need to give your cluster a name. In my case I called my cluster ‘jupiter-cluster’. Under infrastructure I tick the AWS fargate (serverless). This is because we do not want to deal with the underlying infrastructure and want AWS to handle this. In a real work environment, you are also likely to chose this option.



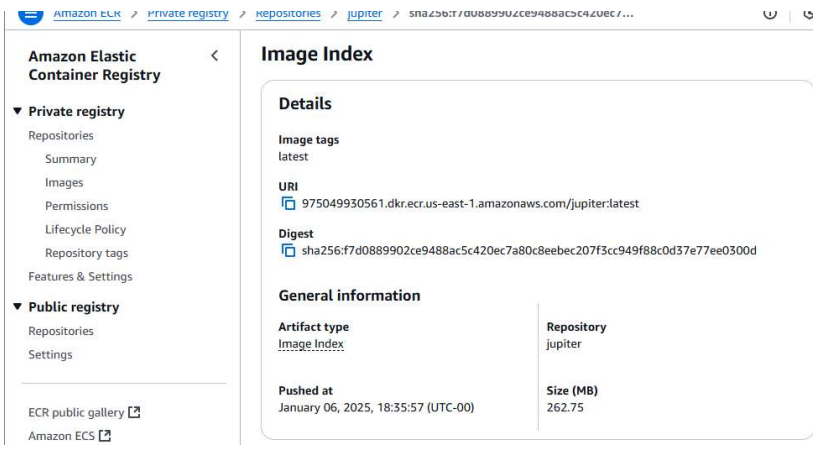
All other options can be left as default and click create.

Create task definition

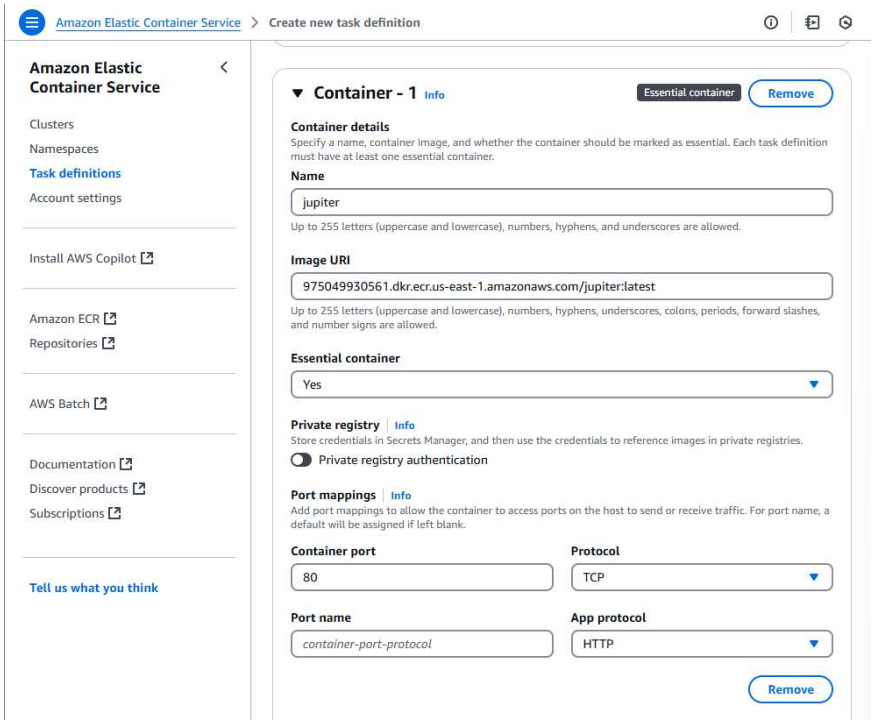
Now we have created the ECS cluster, we will need to create a task definition. Give your task definition a name, make sure under infrastructure AWS fargate has been selected. You operating system/architecture will be linux/X86_64 if you created your docker image on a windows computer. However, if you created your docker image on a Mac computer select linux/ARM64.



Under container – 1, give your container an name and also input your URI. This can be found in your AWS ECR. Navigate to the ECR repository and select the repo you created earlier. Once within this repo, you will need to select the ‘latest’ image tag from the images. Within the image index you will be able to see the URI.



Paste the URI in the image URI.

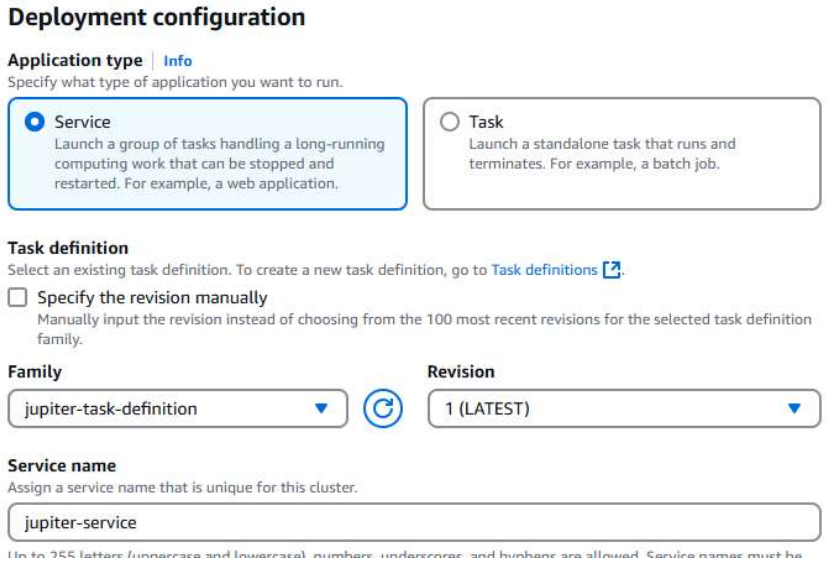


At this point you can leave all other settings as default and click create.

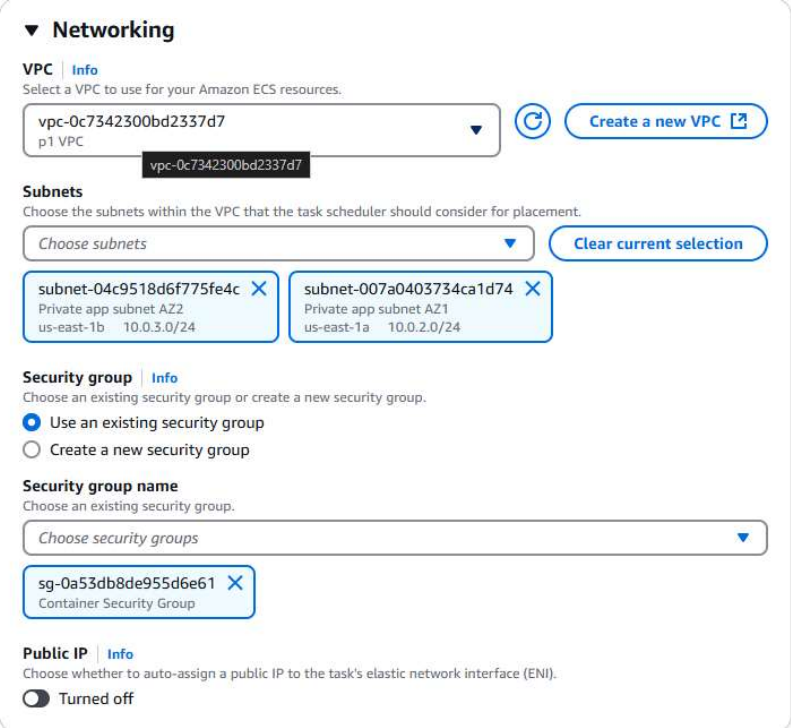
Create a Service

Now we've created the task definition, we'll create the ECS service that will create the fargate containers.

With in the ECS page, click clusters, then select the Jupiter cluster. Scroll to the bottom of the page and click create under the services tab. The setting within 'environment' should be left as default and under 'deployment configuration' select 'service'. Click the dropdown list under 'family' and select your 'task definition'. You will also need to give a 'service name', in my case i put 'jupiter-service'.

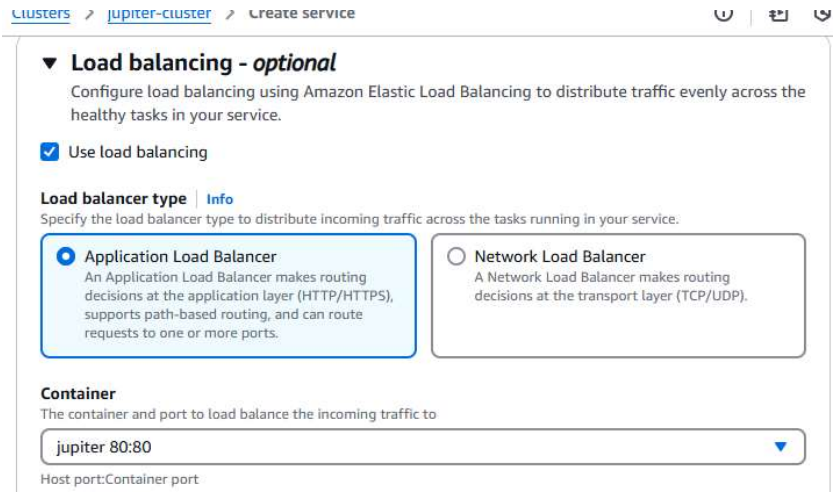


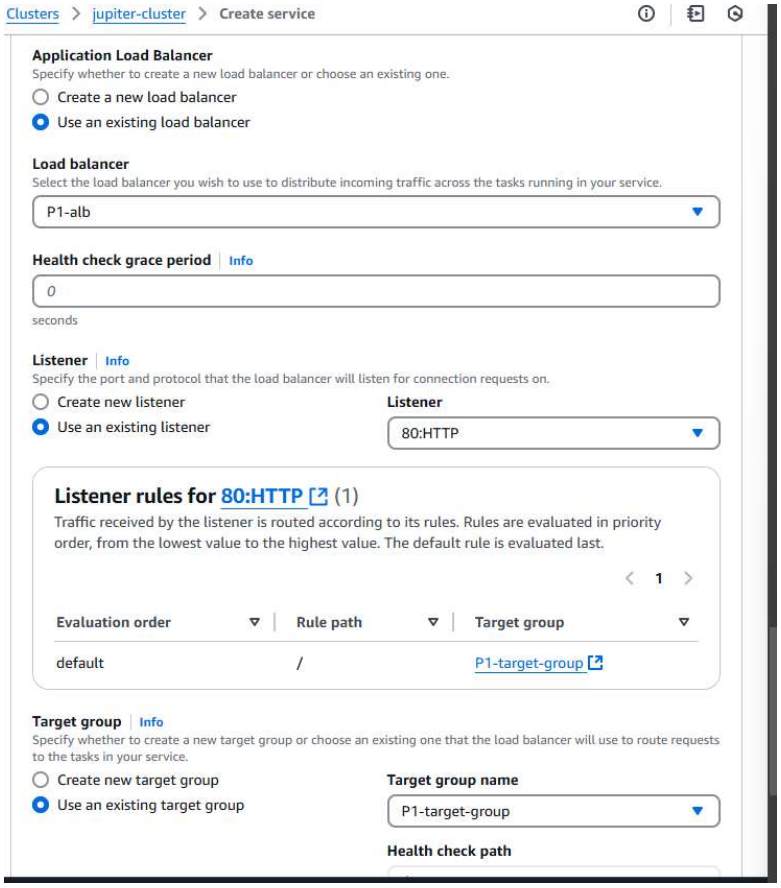
All other setting can be left as default. However, when you get to the networking section you will need to select your vpc from the dropdown list, remove all subnets selected and only select your 'private app subnet AZ1' and 'private app subnet AZ2'. You will also need to remove the default security group and select your container security group from the dropdown list. Next you will also need to turn off the public IP as our container will be in a private subnet.



Next, we'll need to adjust the setting within the load balancer section. Select:

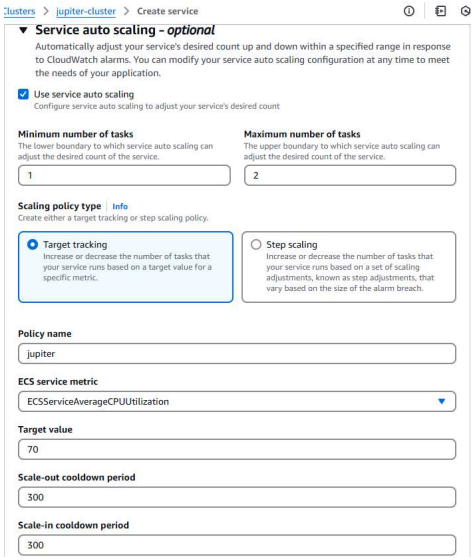
1. Use load balancing
2. Load balancer type: application load balancer
3. Under Application load balancer – select 'use an existing load balancer' and select your load balancer from the dropdown list
4. Under listener select 'use an existing listener' and select your '80:HTTP' listener from the dropdown list.
5. Select 'use an existing target group' and from the dropdown list select your target group created earlier.



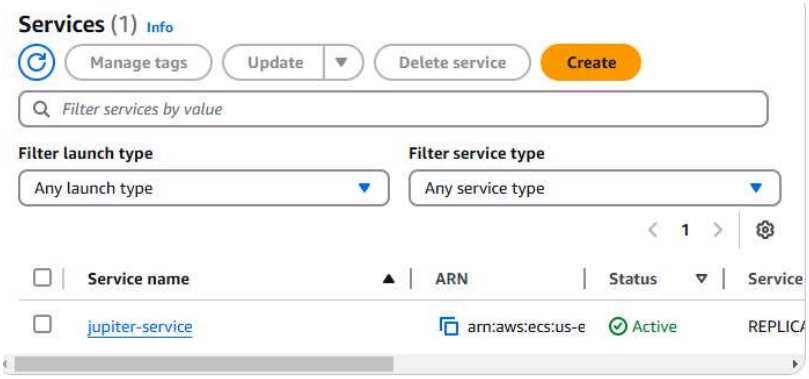


The setting we'll need to adjust next will be the 'service auto scaling':

1. Select use service auto scaling
2. Minium number of tasks will be (1) and maximum number will be (2)
3. Scaling policy type will be 'target tracking'
4. Give your policy a name. In my case I called mine 'jupiter'
5. ECS service metric: ECSServiceAverageCPUUtilization
6. Target value: 70
7. Scale-out cooldown period: 300
8. Scale-in cooldown period: 300



Finally, you can click create and after a few mins your ECS service should be created. When created you should be able to see your service under the services tab.

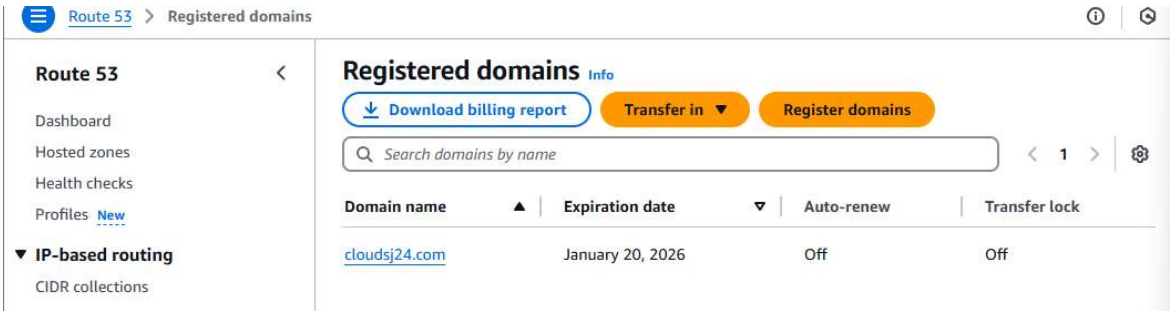


Register a new domain name in route 53

In the section we are going to register a new domain name in route 53, which will allow our end users to access our website using our domain instead of the DNS name of the load balancer.

First you will need to check if your chosen domain name is available via the register domain search.

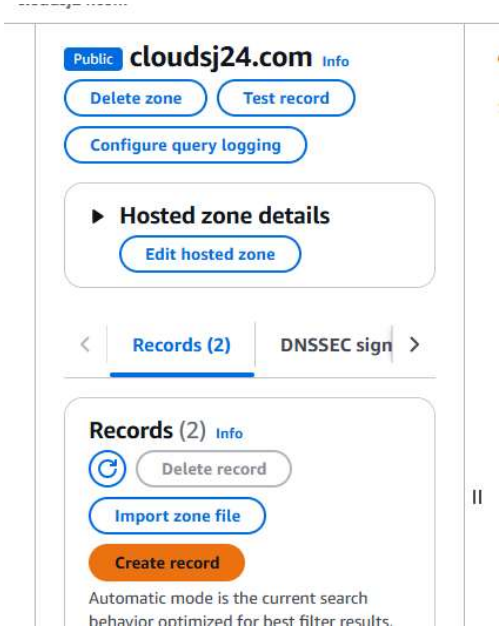
My domain name was available and my purchase price was $14. Go through the process of purchasing your domain name. The process to officially registering your domain name should take around 10-15mins to change the status to successful.



Create a Record in route 53

In the section we'll create a record set in route 53 to point our domain name to the application load balancer.

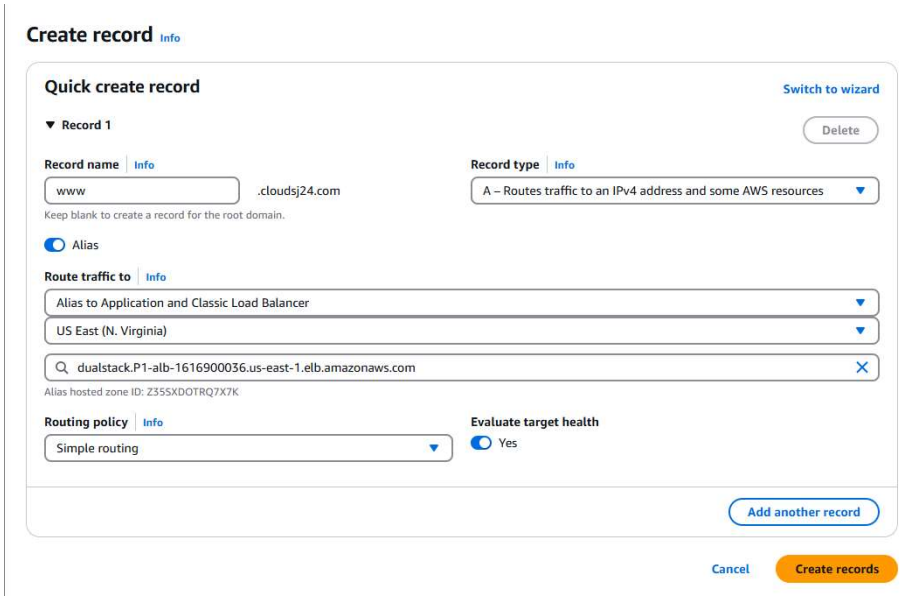
Navigate to the Route 53 in the management console and in the main dashboard click 'hosted zone'. Next you will need to click into your registered domain name and then click 'create record'.



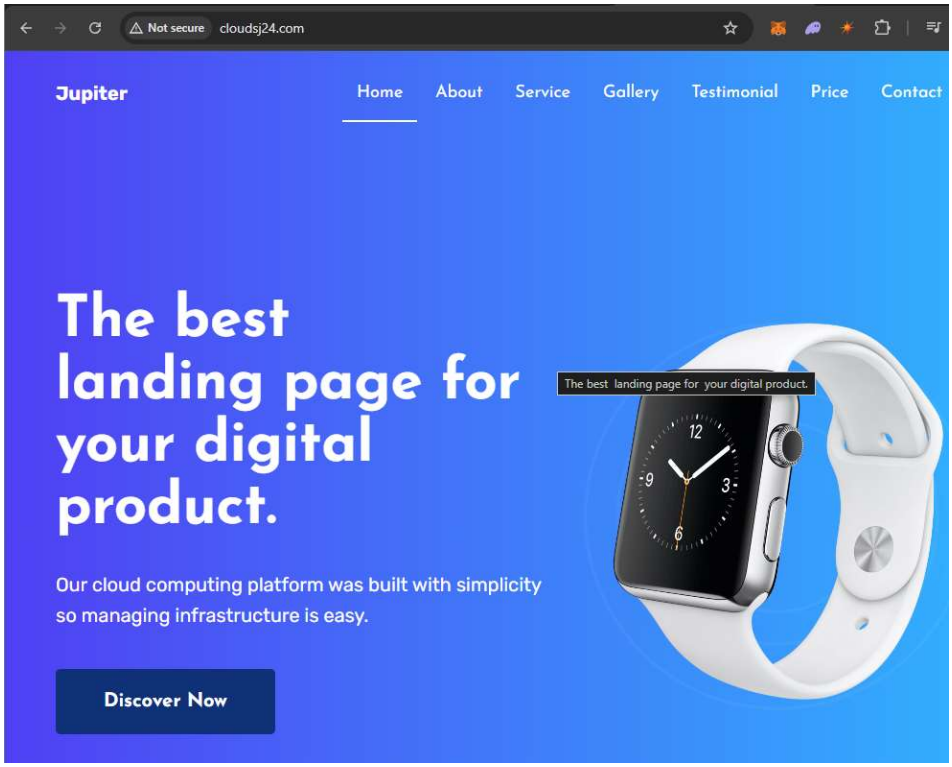
Under the create record section you will need to adjust the following settings:

1. Record name: www
2. Record type: A
3. Make sure Alias has been turned on
4. Route traffic to: Alias to application and classic load balancer, select the region you created your load balancer in, then select your load balancer

All other setting can be left as default.



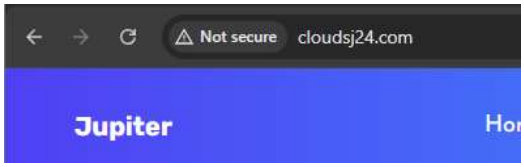
To check your domain is working, select your domain name from the records tab, then copy the record name in my case it will be ‘www.cloudsj24.com’. Search for your domain name in the web browser and your domain name should show your website.



Register for an SSL certificate in AWS certificate manager

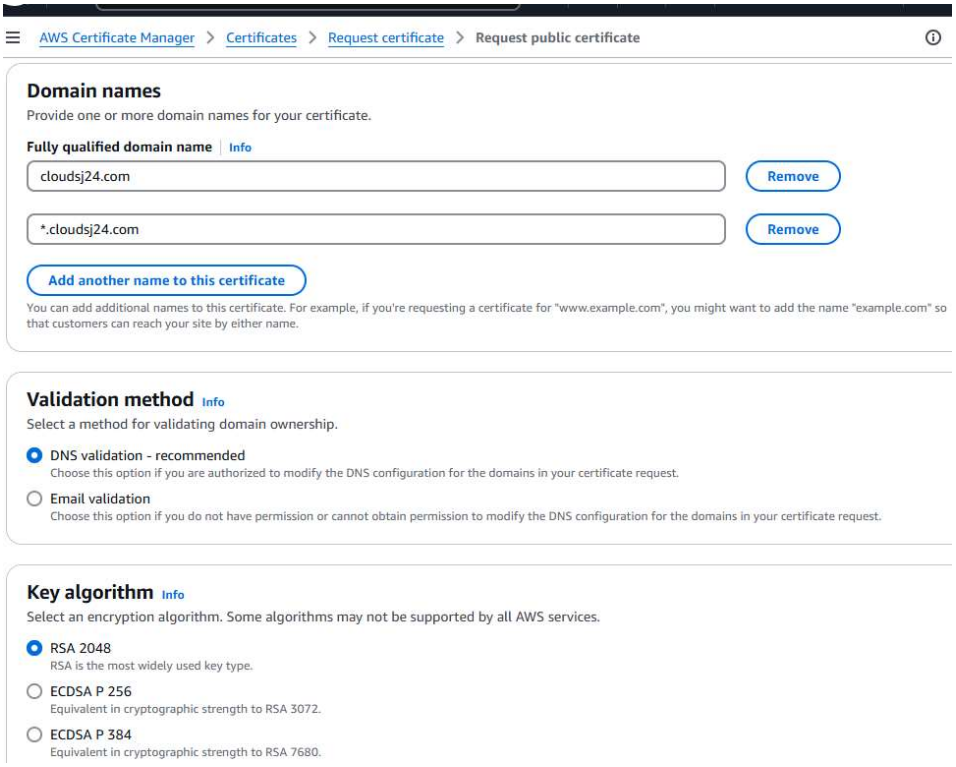
In this section, we’ll register for a free SSL certificate from the AWS certificate manage and we’ll use the SSL certificate to encrypt all communications between the web browser and our web servers.

Currently for our website the communication between the website and web servers are not secure.



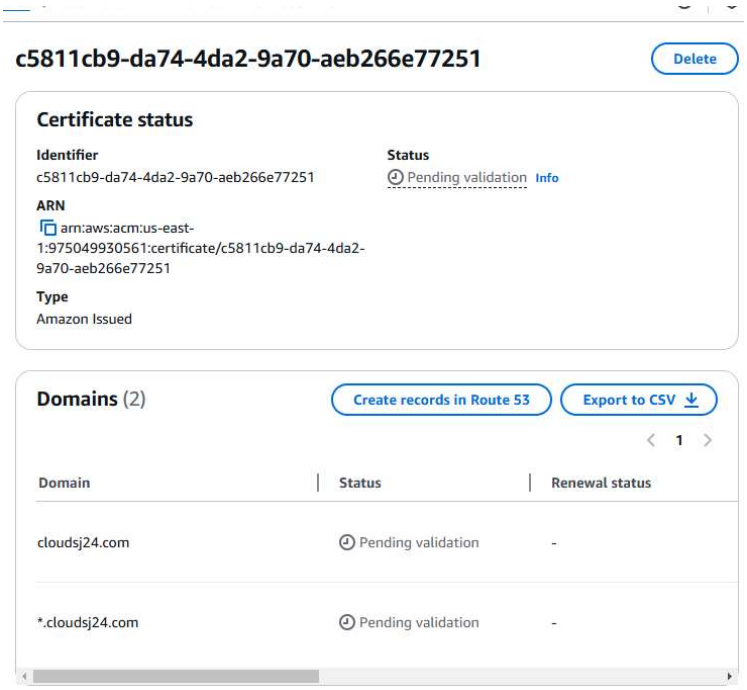
Note this is called encryption in transit.

Now search for ‘certificate manager’ in the management console then click ‘request a certificate’. We are going to request a ‘public certificate’. Next you will need to enter your domain name and also add another name to the certificate. The second domain name will start with a ‘*.’ As this will cover the www as the beginning of your domain name. All other setting can be left as default and click request.

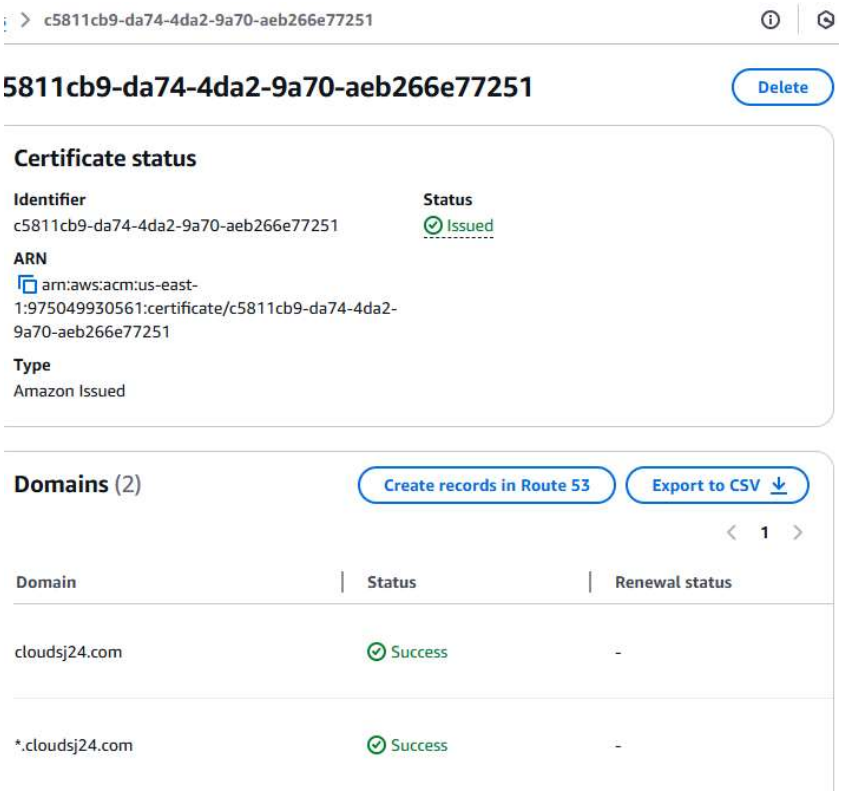


Great! You have created an SSL certificate. Now you will see that the status is saying ‘pending validation’. This is because we need to create a record set in route 53 to validate the domain name belongs to us.

To do this, click ‘create record in route 53’.



Make sure to select your domain and the wild card then click ‘create record’. Next you will need to refresh the page and the status of your certificate should be ‘issued’.



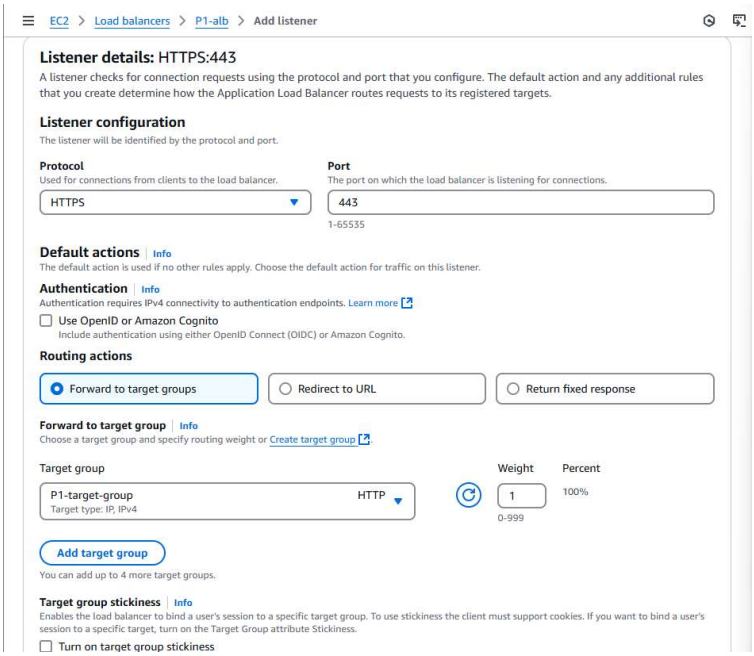
Create an HTTPS (SSL) listener for an application load balancer

In this section we will use the SSL certificate we registered to secure all communications to our website. To do this this in the management console search for ‘EC2’ and then on the left hand side scroll down and click load balancers.

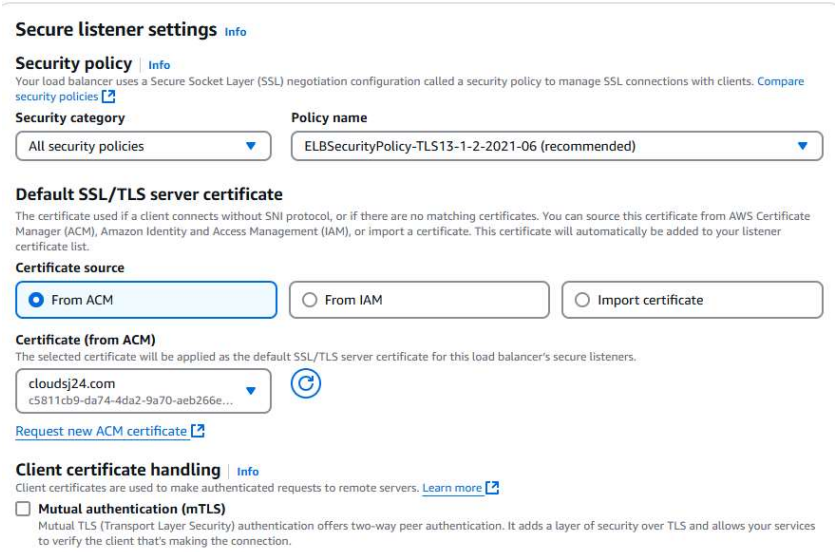
Select your load balancer created earlier and scroll down to ‘listeners and rules’. What we are going to do first is create the HTTPS listener. The HTTPS listener is going to secure the communication between the our end user and our website.

This means any time you see the lock icon in the web browser URL, this means the website is secure. In our case our website is not secure and by creating an HTTPS listener this will secure our website.

Now under the ‘listeners and rules’ tab select ‘add listener’. From the protocol dropdown list select ‘HTTPS’, the target group will be the target group we created earlier in the documentation.



Next you will need to scroll down and under ‘defult SSL/TLS server certificate’ make sure the certificate source ‘from ACM’ has been selected, and from the dropdown list select the SSL certificate created earlier. This should be the name of your domain name.



Now if you scroll down to ‘listeners and rules’ you should have two listeners, one on port 80 and one on port 443. The listener on port 443 is forwarding traffic to our target group and this is also being replicated by the listener on port 80. However, we need to modify the listener on port 80, so that any time a user tries to access our site on port 80 they will be redirected to port 443 for HTTPS.

Select the ‘HTTP:80’ listener, click the manager listener dropdown list and select ‘edit listener’. Once in the edit listener page, under default actions select ‘redirect to URL’ and under redirect to URL select ‘full URL’ then save changes.

Listener details

A listener checks for connection requests using the protocol and port that you configure. The default action and any additional rules that you create determine how the Application Load Balancer routes requests to its registered targets.

Listener ARN

arn:aws:elasticloadbalancing:us-east-1:975049930561:listener/app/P1-alb/e2313305f5e3e86a/635dbce79a737ee5

Listener configuration

The listener will be identified by the protocol and port.

Protocol
Used for connections from clients to the load balancer.
HTTP

Port
The port on which the load balancer is listening for connections.
80
1-65535

Default actions | Info
The default action is used if no other rules apply. Choose the default action for traffic on this listener.

Routing actions

- Forward to target groups
- Redirect to URL
- Return fixed response

Redirect to URL | Info
Redirect client requests from one URL to another. You cannot redirect HTTPS to HTTP. To avoid a redirect loop, you must modify at least one of the following components: protocol, port, hostname or path. Components that you do not modify retain their original values.

URI parts | Full URL

Full URL | Info
Enter the redirect URL.
https://#{host}/#{path}?#{query}
protocol://hostname:port/path?query

Status code
301 - Permanently moved

Now if you go back to your ALB, you will see that under 'listeners and Rules' the HTTP:80 listener is being redirected to HTTPS 443. This is all we need to do to encrypt the communication between the end user and website.

To confirm your site is secure, go back to your website and you should see the lock icon in URL.