Cloud Computing A3

Site URL's:
Elastic Beanstalk: cloudfitness.click
API Gateway: api.cloudfitness.click
ECS/EC2: admin.cloudfitness.click

Student Names:
Tomas Haddad: s3811596
Alicia Hallal: s3811836

Introduction

Usage of fitness and health technology, platforms and applications are steadily on the rise, with 39% of adult Australians confirming that they used digital technology to assist in meeting their fitness goals during lockdowns[1]. As such, there is an increasing demand for applications which help users create, plan, and manage their exercises and workouts. Many existing web or mobile applications aim to do just that, but often either provide a broad array of services not useful to these specific needs or are too restrictive and lack any meaningful user interaction.

CloudFitness is a web application that is designed to help users quickly find exercises appropriate for their experience level and interests. Users can select from exercises contained within the public database or can create their own custom exercises if they do not already exist. The motivation for this application is to help users engage with particular exercises so that other like-minded users can benefit from the same exercises by the following: reading their comments, viewing the likes on the exercises, and building their exercise profile pages. This way they can build more efficient and enjoyable exercise routines with the exposure of different exercises and user comments.

CloudFitness is made up of three separate applications which all provide specific functionalities.

Main User Application:

- Users can search exercises by their names
- Allow users to view exercise view and like counts
- Allow users to comment on exercises
- Users can add exercises to their profiles
- Users can submit their own exercises via upload functionality, these must be approved before being published, or upload personal exercises to their profile.

Admin Application:

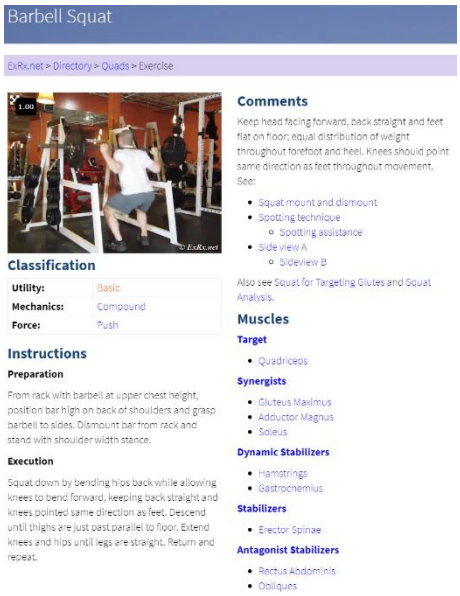
- Load exercises from API Gateway
- Edit exercises on API Gateway
- Set approved exercises to pending
- Set pending exercises to approved
- View all exercises available from API gateway

REST API

- Get all exercises in the database
- Get exercises by type and name
- Get exercises by approved status
- Update exercise approved status (this requires an API key)

Related work

exrx.net



Exercise Prescription (ExRx) is an online provider of fitness-related information. Their available content includes an exercise library containing over 1,900 entries, fitness calculators, aggregated information covering a diverse range of topics such as weight loss, nutrition, and more.

Of particular interest in the context of this report is their exercise library, which is organised into muscle groups and the type of equipment needed, if any. Each entry provides a short summary of the exercise, including its classification, instructions on how to execute the exercise, a general comments section, and a list of the muscles involved.

A major downside of the website's simple, minimalist structure is that it does not provide any user interaction beyond its public forum; the content of each exercise is controlled entirely by the staff's administration.

wger.de

wger provides a public database of exercises in addition to providing tools for managing meals, nutrition, and weight tracking. Users can use their website to search for exercises based on equipment or muscles used. The defining feature of this website is that it provides a REST API for users to retrieve information such as equipment, exercise information, exercise category, meals, ingredients, and more.

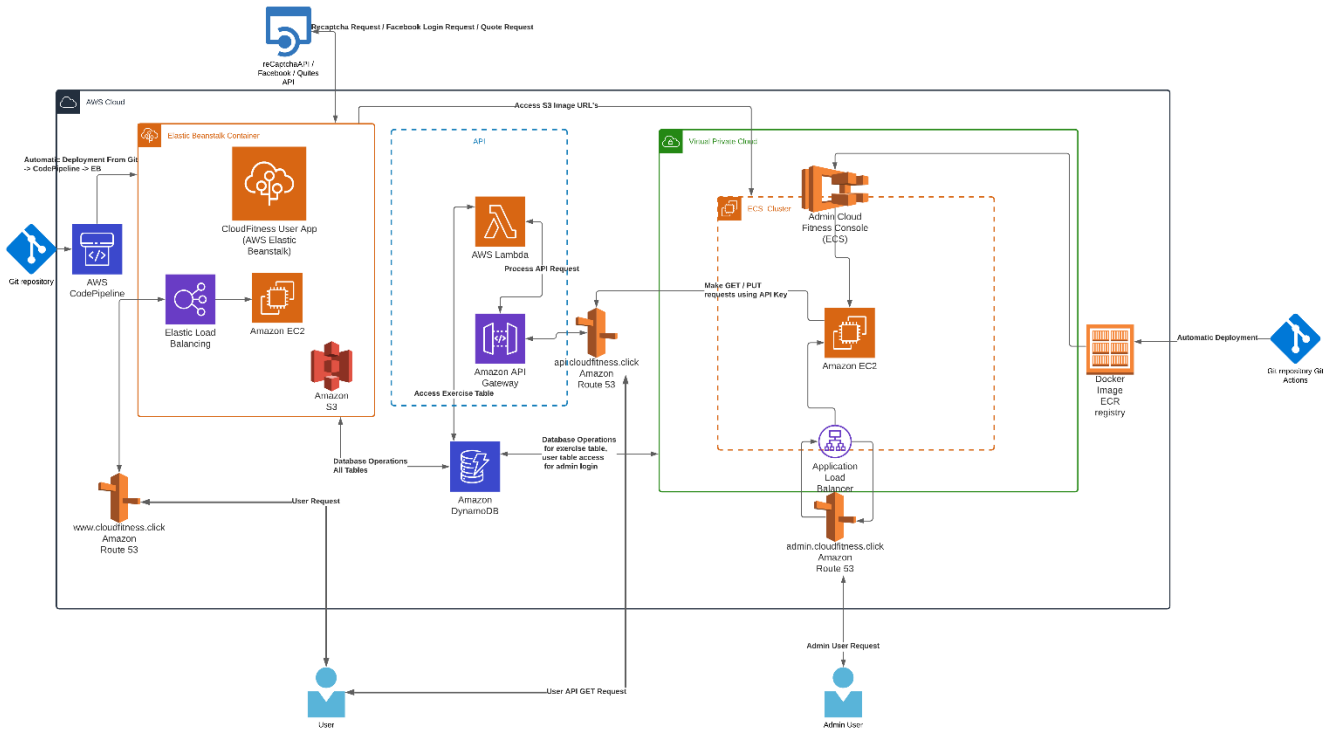
Unlike exrx.net, users can submit exercises to the website which are added to the public database subject to administrator approval. Furthermore, users can create accounts to create and manage custom workout routines. However, a major limitation of the website is that users cannot create or manage custom exercises and are limited to the public database of approved exercises.

bodybuilding.com

This website is the world's largest online fitness product and information provider, but also provides registered users with the ability to build custom workouts under their "BodySpace" workout builder. Users can build custom workout routines using exercises available in the public database. However, again, users cannot create their own exercises without administrator approval. Furthermore, the majority of exercise information is locked behind paywalls, and the broad scope of the website will not likely attract many users for this specific functionality.

System Architecture

System architecture diagram



System descriptions

Elastic beanstalk (5)

Elastic Beanstalk gives the developer a simple, low configuration solution that is also scalable during higher demand situations. It was the perfect option when trying to decide what was the best way to deploy the main site. This is because it is easy to configure with automated AWS deployment systems such as CodePipeline, and also sets up its own S3 bucket and EC2 instances. This made it quick and easy to deploy the flask application main site once the code for it had been developed. It also had the ability to run container commands before deployment, meaning we could have more control over what happened on the EC2 instances before the application started up, in this case we used it to ensure that the DynamoDB tables were created before the site deployed. Furthermore, access to DynamoDB was easy to configure by giving permissions to the Elastic Beanstalk Role in the IAM AWS console. Elastic Beanstalk has its own load balancer automatically set up upon deployment, which was extremely easy to hook up to Route 53. It has two connection points www.cloudfitness.click and cloudfitness.click. The application load balancer was easily configured using .ebextension config files. Which means that each time the system is deployed, we can be confident that HTTP and HTTPS will work correctly.

ECS (5)

Elastic Container Service allows more control over how the admin flask site was deployed, whilst also giving us a fully managed platform solution. If one of the EC2 instances which hosted the admin site failed, ECS would automatically deploy another instance with the

last working image on it. It also gave us a secure way to deploy our application using github actions and Elastic Container Registry. It does take more time to create the Dockerfile and deployment files initially when using ECS, however, it does pay off in the long run. Environment variables are set as secrets and then automatically built into the application meaning that nothing secure is uploaded to any public/private instance. After the initial set up future deployments are all completely automatic using GitHub actions. Furthermore, ECS gives more options in terms of scalability in the future, we do not expect too much demand currently from the admin site but this may change in the future, with more submissions/uploads on the main site. Using ECS scaling the admin application becomes extremely simple and there would be almost no downtime in launching a new instance that was larger and could handle more requests.

DynamoDB (2)

Separate DynamoDB tables are used to store a number of different types of records: the user table stores login information; the exercise table stores information about each exercise needed to generate URLs, facilitate searching, and render pages; the comments table stores user-submitted comments on exercise pages; the metadata table contains static information, such as exercise types, accessible by the REST API; and existing and custom made exercises are stored in two separate tables to allow users to maintain their own custom exercises separate from exercises submitted to the main website.

API Gateway (5)

API gateway acts as a single-entry point for our backend REST API. Using API gateway's proxy functionality, the request body generated when users visit a valid API URL is effortlessly passed to a lambda function, which then processes the URL based on the parameters given.

Lambda (5)

Amazon Lambda's serverless architecture allows backend functions to be run only when required. Like wger.de, we expose our database to developers as a REST API via *api.cloudfitness.click*. Since this is not a service that needs to be online at all times, a lambda function is instead triggered by API gateway when a valid API call is made.

Route 53 (2)

Amazon Route 53 is used for domain registration and routing. A domain is needed in order to utilise the Facebook login API, which only works with HTTPS-authenticated websites. Furthermore, adjacent services such as the REST API can be mapped to their own subdomains, which allows for changes to be made to application business logic without affecting the frontend web application.

ReCAPTCHA API

reCAPTCHA is used to protect websites from spam or automated abuse (such as bots). The API here is used for login on the main site www.cloudfitness.click and uses reCAPTCHA v2 Checkbox. An API key pair was specifically generated for this site so that it can only be used on this domain, information about requests on the reCAPTCHA api are available to the logged in developer who owns the API key pair.

Facebook login API

Facebook login API allows users to log into the main site www.cloudfitness.click with their facebook logins. This facebook login has an developer account attached to its credentials with cloudfitness as the authorised website domain. Facebook API login only allows users to use the website if it is HTTPS/SSL secure. Facebook user details are automatically sent to the dynamodb database upon their first login to generate them a localised account for the site, users will not be able to log into the site via normal means using these accounts, only via the facebook login API.

Zenquotes API

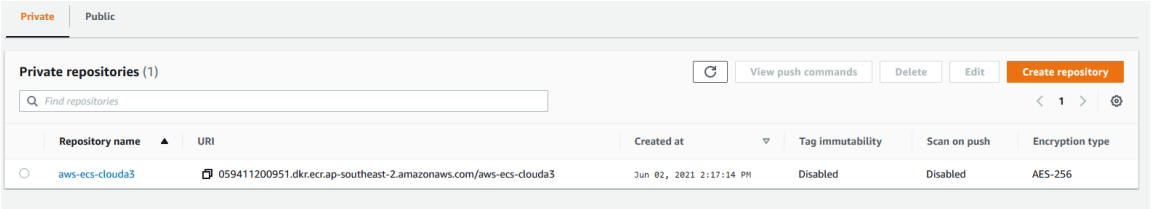
This simple API is presented to users when they view the main site home page www.cloudfitness.click. It is an API which randomly selects an inspiring quote and presents it to users. This is meant to motivate exercise users of the site so they can get the best out of the exercise routines each time they view the home page.

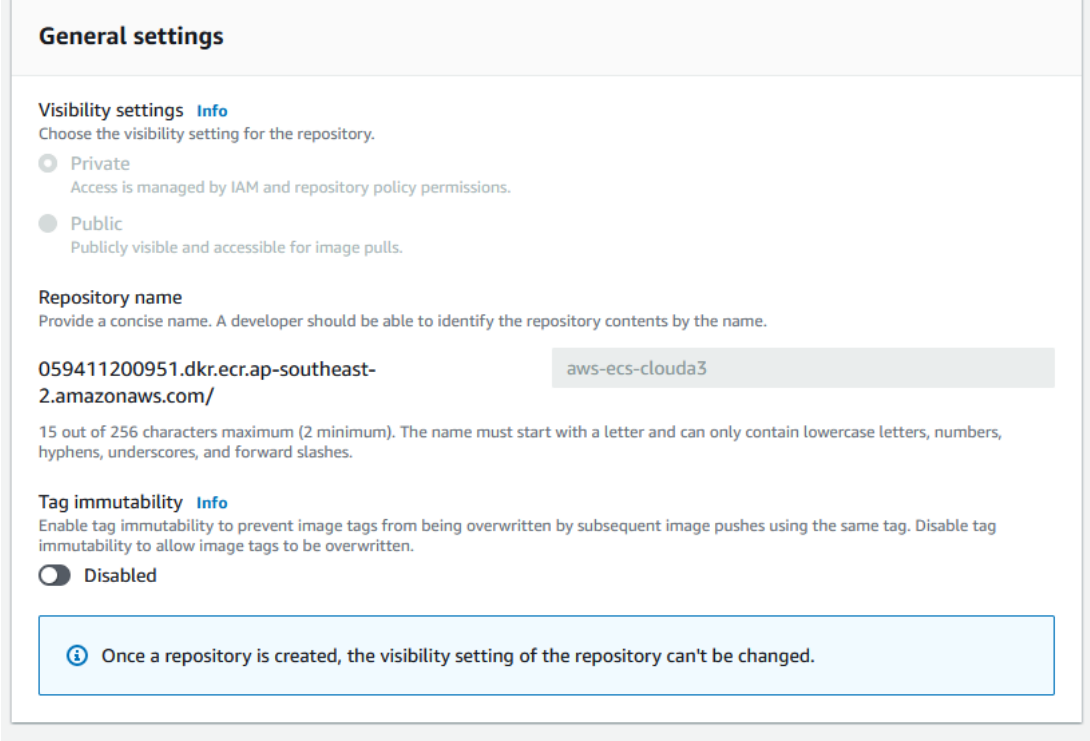
Developer Manual

Section 1: Admin Site *(Elastic Container Service (EC2) / Elastic Container Registry / GitHub Actions)* [2]

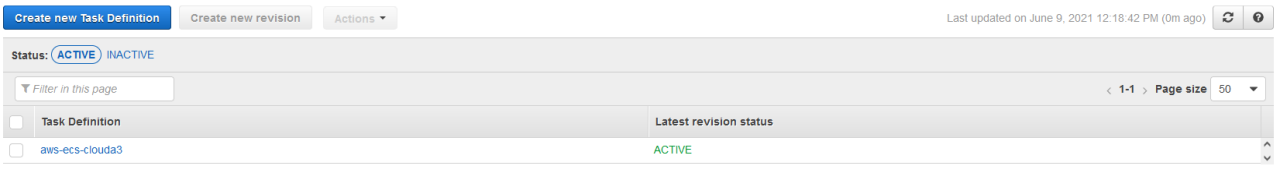
Before beginning this step of development, a functioning admin application is needed, complete with a Dockerfile so that it can be used to upload to the EC2 instance. This application needs to be associated with github.

1. **Create the repository on Elastic Container Registry, and set visibility to private so that it can not be accessed and give it a meaningful name:**

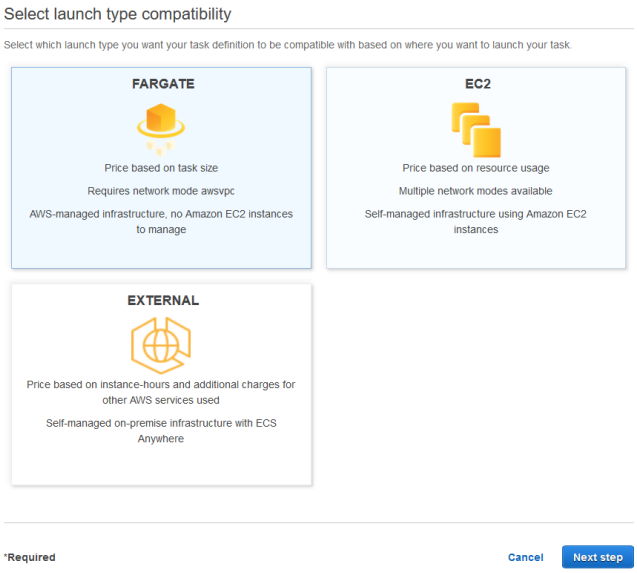




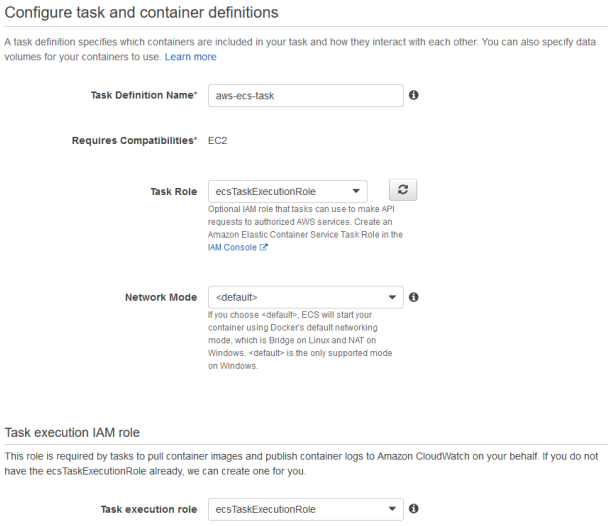
2. **Create the Elastic Container Service Task Definition:**
   This will specify how many containers should run as part of the task, what type of server they will run on (in this application we are using an EC2 instance) and how the instances will link together. Click create new task definition.



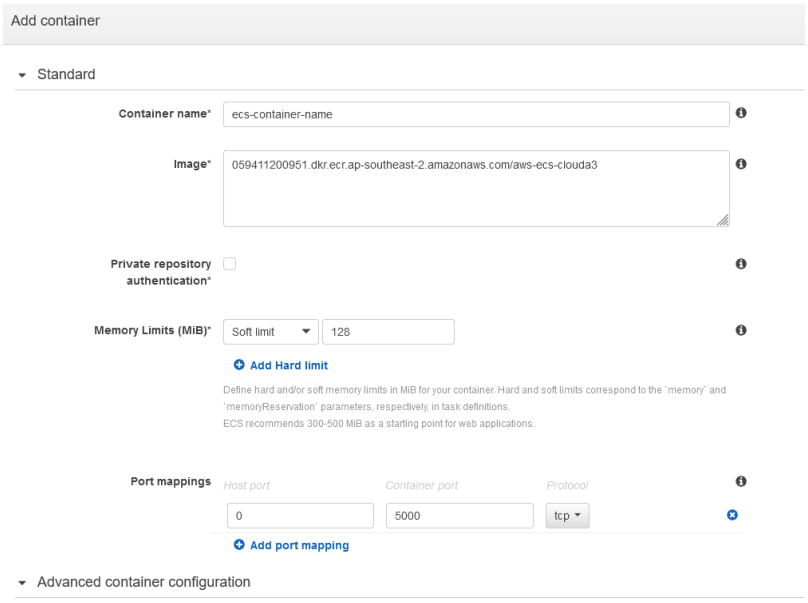
Select EC2 instance for launch type compatibility



Give the task a meaningful name and set the Task role to ecsTaskExecutionRole, Network mode to default, IAM role should be ecsTaskExecutionRole



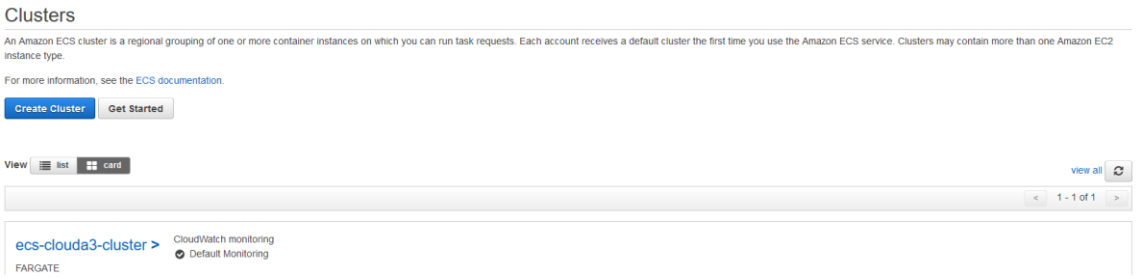
Click ‘add container’ in container definitions, give the container a meaningful name and add the URI of ECR repository created in the previous steps. A memory limit of 128mb is set here as the application does not need to use much memory, and it will be hosted on port 5000, so port 5000 is set as the container port.



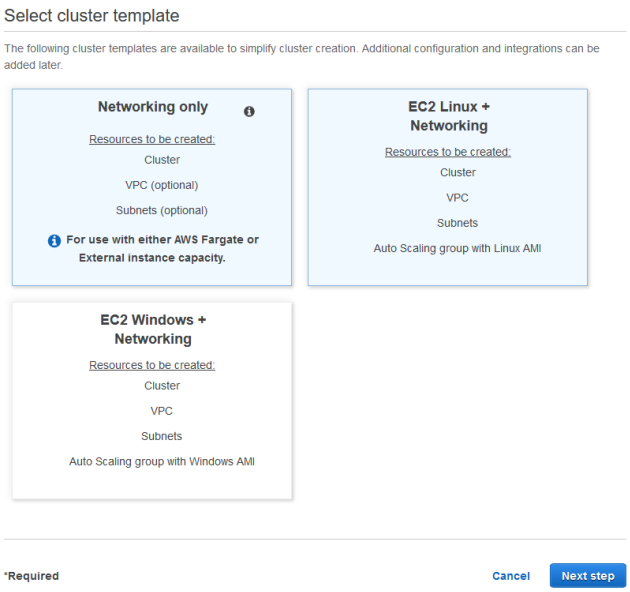
No other configurations are needed, so click add the container and then create the task so that it shows up in the task definitions screen.

3. **Creating the Elastic Container service Cluster:**

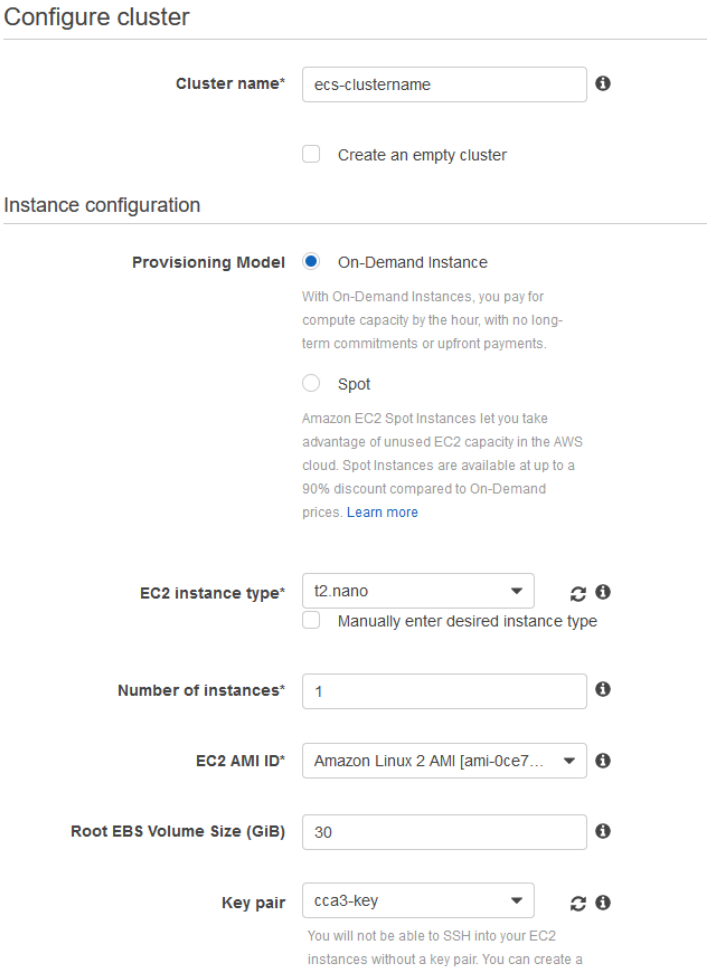
The cluster for ECS is a grouping of the tasks/instances



To create the cluster, click ‘create cluster’. Select the cluster template EC2 linux + networking.

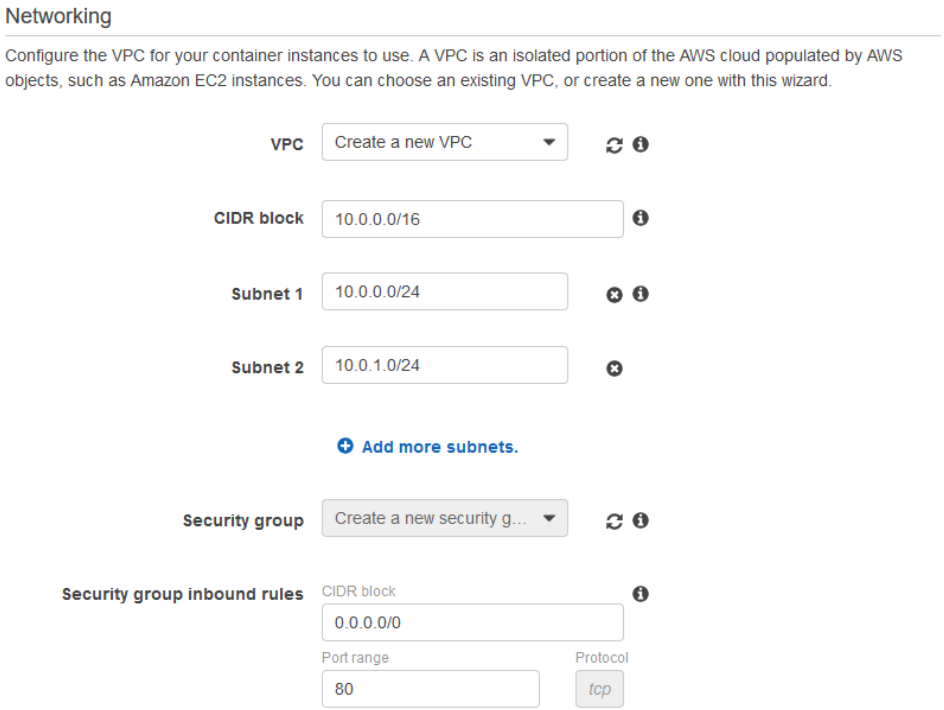


Give the cluster a meaningful name, set the ec2 instance type to t2.nano and the number of instances to 1. As the task we are running here is a web application that does not need much memory.



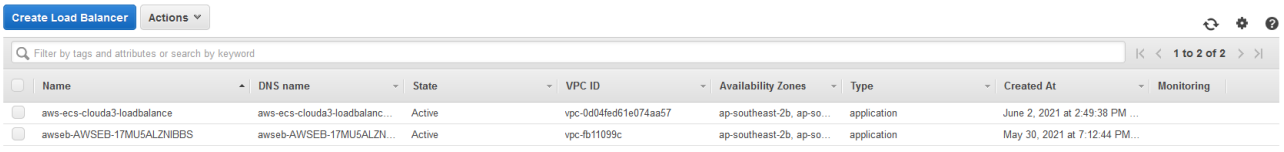
Ensure that the key pair is set so that the instance can be accessed with SSH as this cannot be changed after creation.

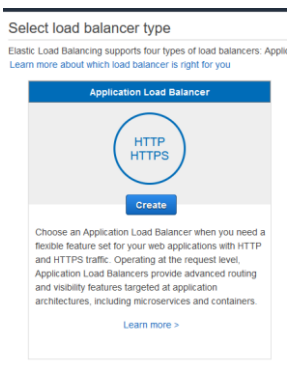
For networking within this cluster a new VPC will be created so that all components within the cluster can talk to each other but will not be accessed by outside components not part of the system.



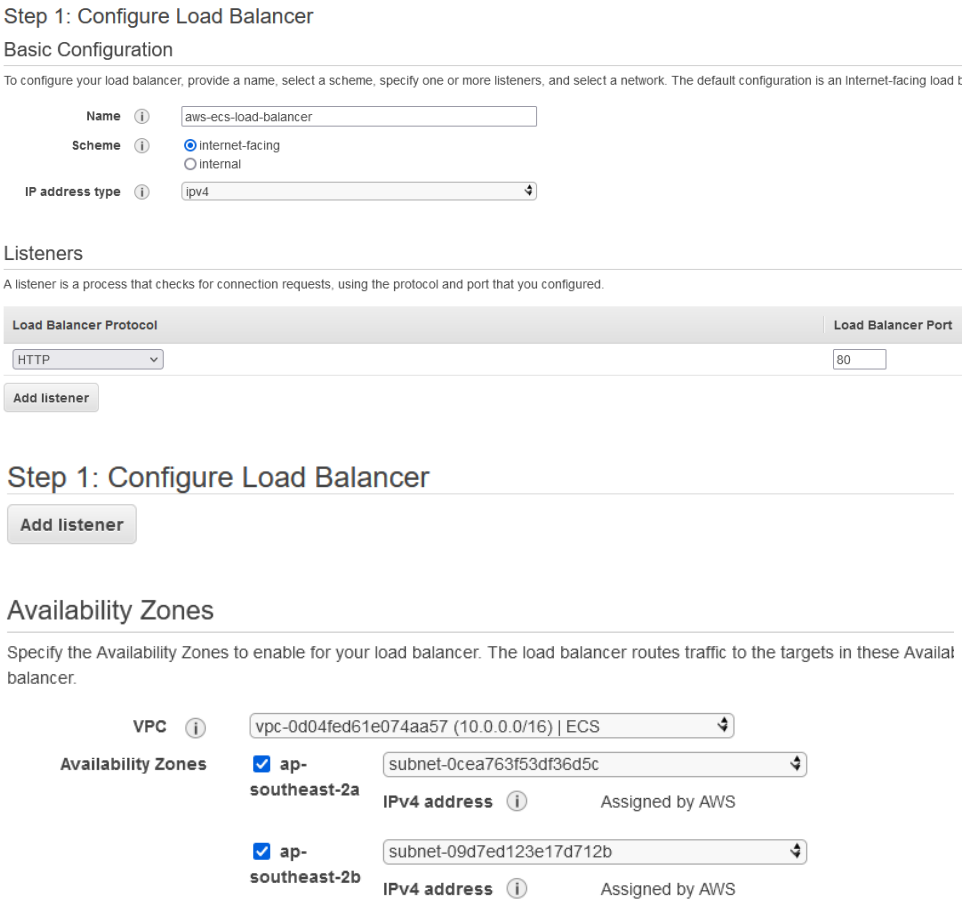
Create the cluster with these settings.

4. **Create an application load balancer for our application.**
Before moving to the next step, we need to create a load balancer so that we can distribute the incoming traffic across the tasks that will run in our ECS service. We can do this in the **EC2 Console** and select **Load Balancers** in the side menu. Click ‘Create Load Balancer’ and select ‘HTTP HTTPS’ Application Load Balancer



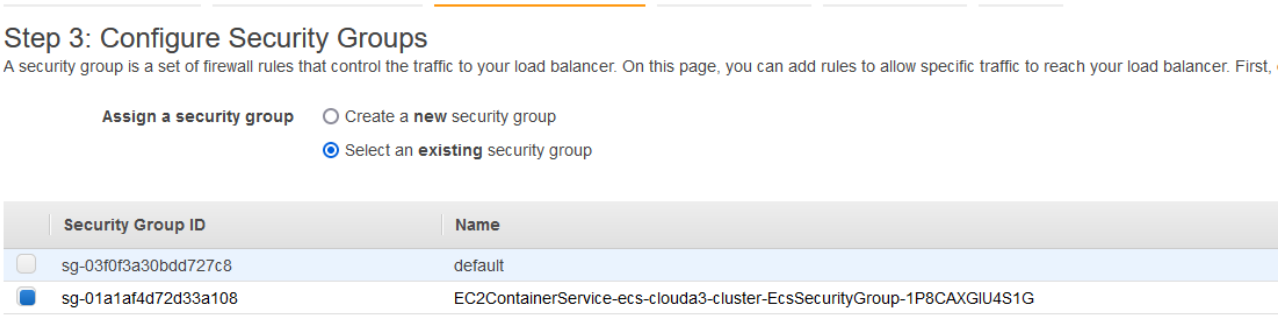


Ensure you set a meaningful name to the load balancer. The most important step of creating this load balancer is to ensure that the Availability zone VPC is set to the same one that we previously made when creating the cluster so they can talk to each-other within the VPC.

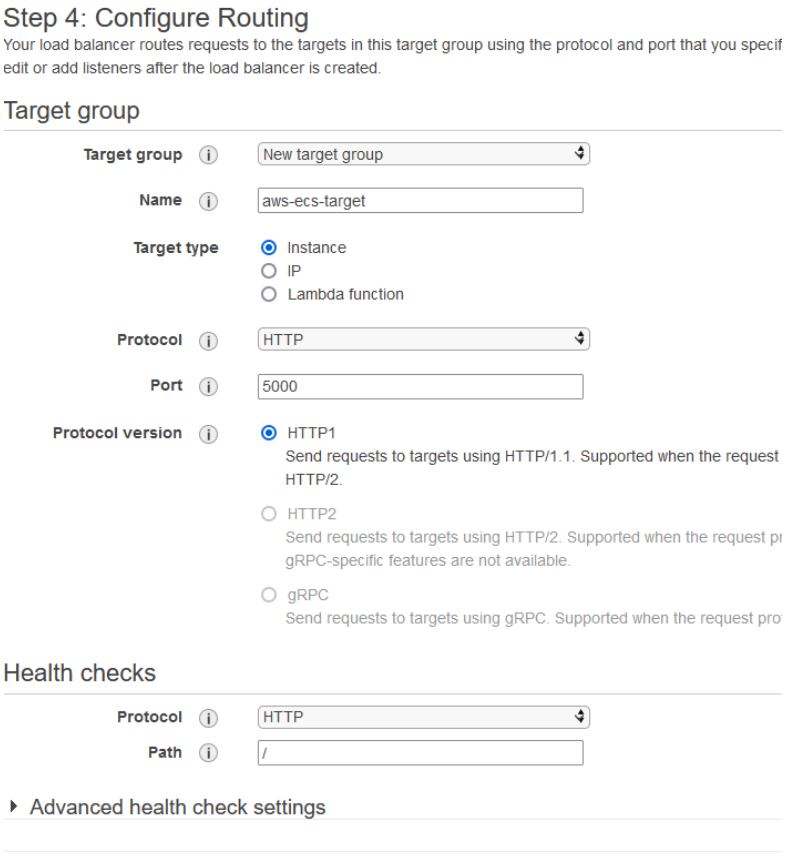


Click ‘Next’ and ‘Next’ again to ‘Configure Security Groups’

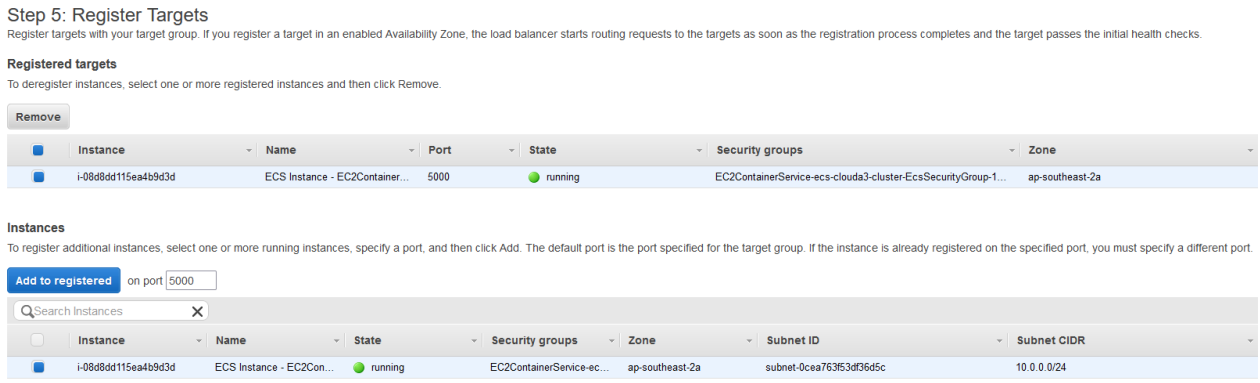
Set the security group ID to the same one created for the EC2 Service.



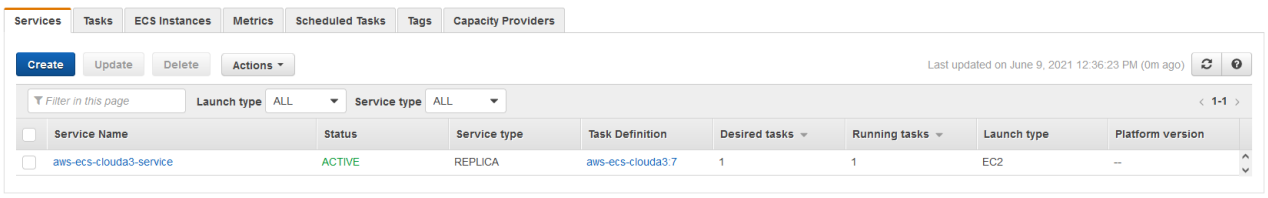
Click ‘Next’ to the ‘Configure Routing’ screen. Give a name to the target group to create one and set the port to 5000, as this is the port we are hosting the application on.



Click ‘Next’ to ‘Register Targets’ select the target and press *Add to registered*.

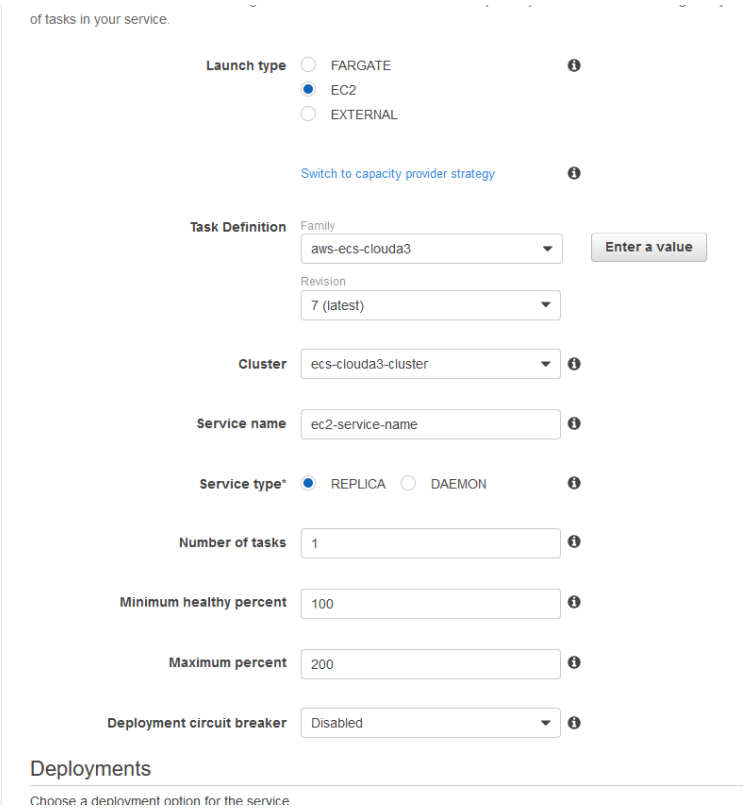


5. **Create the ECS Cluster Service:**
   Now that we have a cluster, you can click the created cluster to add a service to it.



The service allows us to better manage our tasks inside the cluster, specifically how many copies of the task we will run.

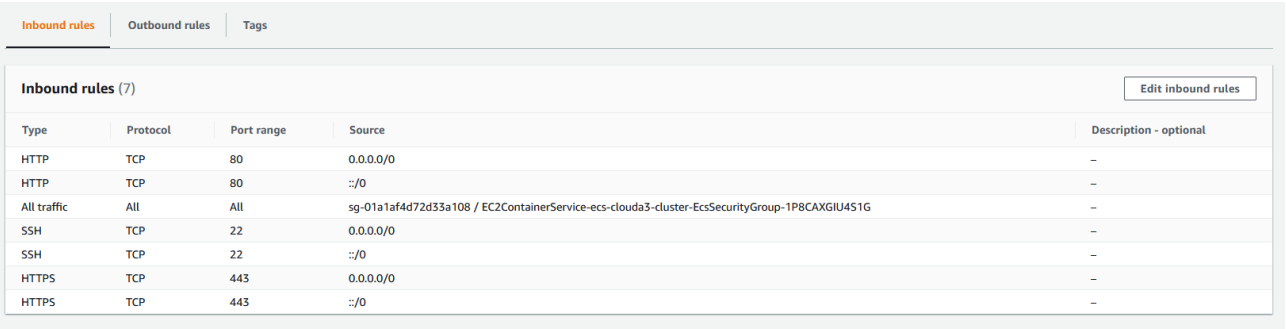
Click ‘Create’ and set the launch type to EC2, ensure the task definition is the one previously created above and the cluster is the previously created cluster. Set the service type to REPLICA and for this application we will be using 1 task.



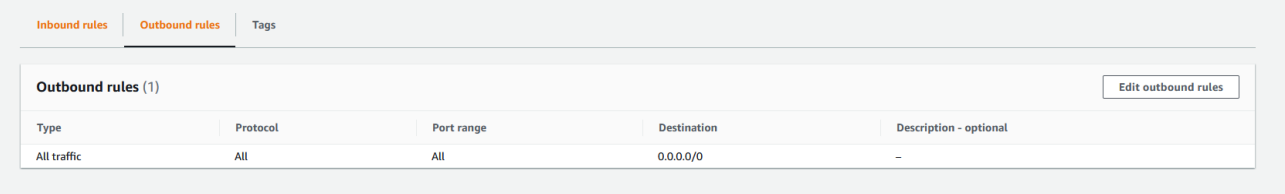
Click ‘Next’ to set the load balancing settings of the application to ‘Application Load Balancer’ and select the load balancer we created in the previous step. Click ‘Add to load balancer’ set the ‘Production listener port’ to port 80 and set the ‘target group name’ to the target previously created. Click ‘Next step’ then click ‘Create service’.

6. **Set the EC2 Instance Security settings to the following:**

Inbound Rules: Note that ‘All traffic’ is set to the security group of the EC2 instance itself.



Outbound Rules:



7. **Set the Domain name for the admin site in Route 53**
   See Section n-3a: Route 53 *(Routing traffic – Admin site)*.

8. **Ensure the Dockerfile is created before moving onto the steps below:** As this admin ECS site uses a Docker Image to create the application, a Dockerfile of how to set up the environment before launching the application is needed and should look exactly like below.

   Dockerfile

```
# get base image
FROM python:3.7

RUN apt-get update
RUN apt-get install -y gunicorn
RUN apt-get install -y python-gevent

# set ENV variables
ENV FLASK_ENV=production

ARG AWS_ACCESS_KEY_ID
ARG AWS_SECRET_ACCESS_KEY
ARG AWS_DEFAULT_REGION

ARG API_GATEWAY_KEY

ENV AWS_ACCESS_KEY_ID $AWS_ACCESS_KEY_ID
ENV AWS_SECRET_ACCESS_KEY $AWS_SECRET_ACCESS_KEY
ENV AWS_DEFAULT_REGION $AWS_DEFAULT_REGION
ENV API_GATEWAY_KEY $API_GATEWAY_KEY      You, 4 days ago • added apigatewaykey to

ADD ./requirements.txt ./requirements.txt

RUN pip install -r ./requirements.txt

# copy files
COPY . ./admin-project
WORKDIR /admin-project

# start gunicorn web server
CMD ["gunicorn", "-b", "0.0.0.0:5000", "application:application", "--workers=5"]
```

   This dockerfile ensures our environment contains all the necessary components before running the application and ensure that each time we deploy, the deployment will run exactly the same. It sets all the environment variables of the environment, copies the admin project folder and launches the application using gunicorn on port 5000.

To test the application image locally, a docker-compose.yml file must be created exactly as the one below, which will build and run the image locally so that testing of the image can be done without deploying the application. You will need to set the environment variables in your terminal before composing this locally:

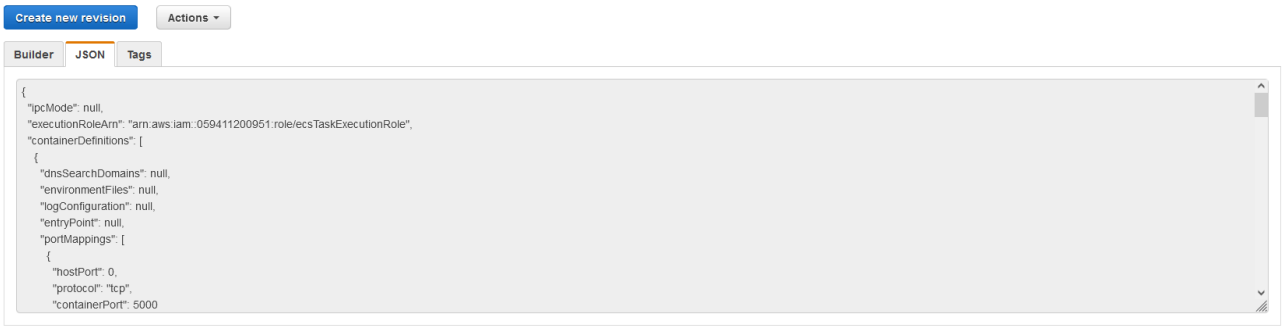
Docker-compose.yml

```yaml
version: "3.1"      You, a week ago • updated admin for ecs

services:
  aws-clouda3-ecs:
    build:
      context: ./
      dockerfile: ./Dockerfile
      args:
        - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
        - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
        - AWS_DEFAULT_REGION=${AWS_DEFAULT_REGION}
        - API_GATEWAY_KEY=${API_GATEWAY_KEY}
    container_name: aws_clouda3_ecs
    ports:
      - 5000:5000
    environment:
      - AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}
      - AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}
      - AWS_DEFAULT_REGION=${AWS_DEFAULT_REGION}
      - API_GATEWAY_KEY=%{API_GATEWAY_KEY}

    network_mode: bridge
```

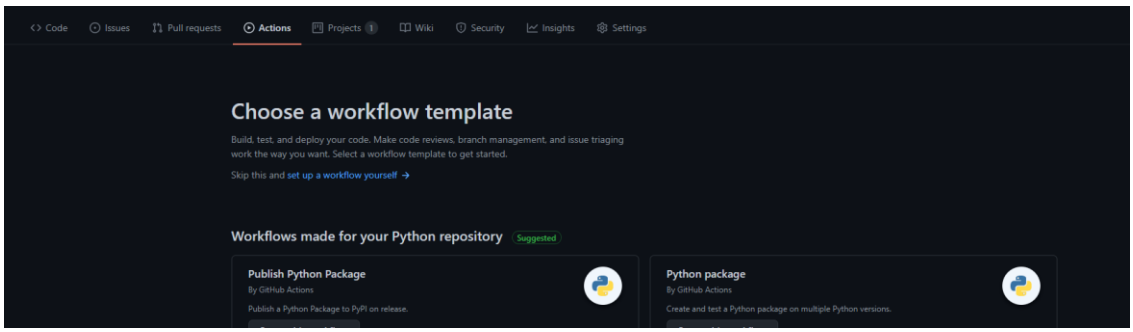
9. **Deploying with GitHub Actions**

To deploy the application automatically, you must first create an 'aws-task-definition.json' file which contains the task definition information copied from the task json on AWS. Click the task definition created in the previous steps and then click the latest revision.

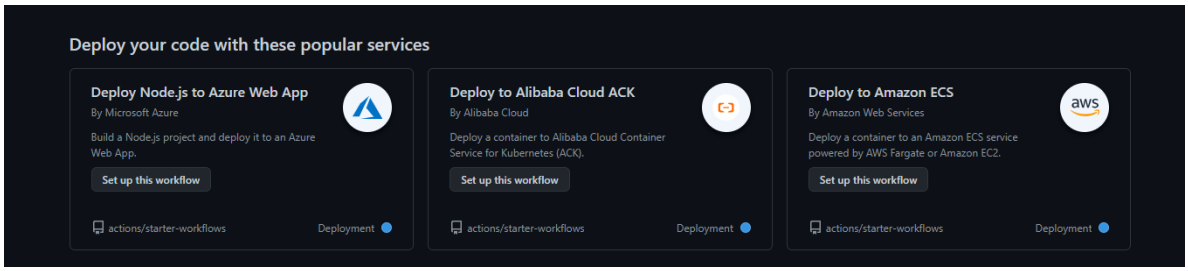
Click the JSON tab and copy the information here into the 'aws-task-definition.json' file:



To create the GitHub Actions workflow, click ‘Actions’ on GitHub



And scroll down to the Deploy to Amazon ECS service.

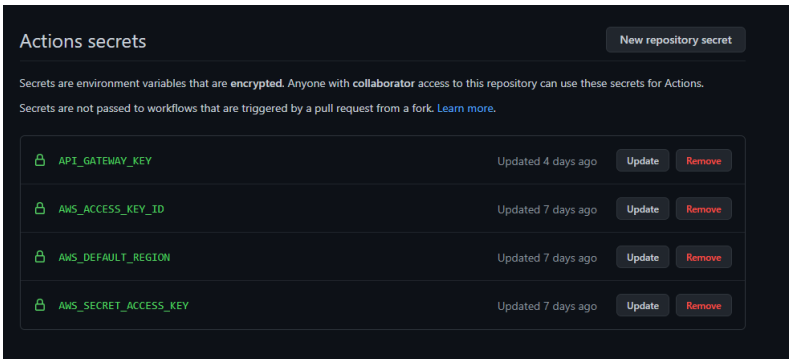


Click ‘set up this workflow’ so that GitHub will create the boilerplate code for the workflow in an aws.yml file which we will configure.

Below release and types, add ‘workflow_dispatch’ so that the deployment can be run without publishing a version. Give the workflow a name ‘Deploy to Amazon ECS’:

```yaml
on:
  release:
    types: [created]
  workflow_dispatch:

name: Deploy to Amazon ECS

jobs:
  deploy:      Alicia Hallal, a week ago • Create aws.yml
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
      - name: Checkout
        uses: actions/checkout@v2

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_DEFAULT_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push image to Amazon ECR
```

To set the environment variables for the GitHub deployment, we must set some ‘Secrets’ in the GitHub settings tab: Set the following secrets to the proper values.



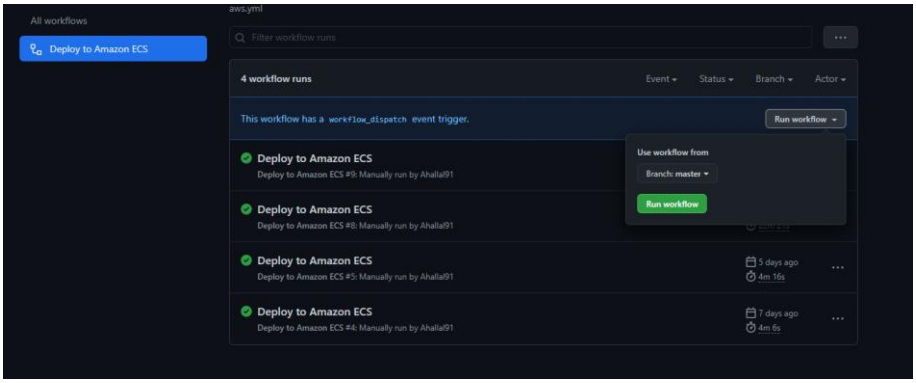
In the aws.yml file set the ECR_REPOSITORY name to the name of the ECR repository created previously, then enter the docker build commands to build the docker file with the environment variables like below.

```yaml
- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v1

- name: Build, tag, and push image to Amazon ECR
  id: build-image
  working-directory: admin-project
  env:
    ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    ECR_REPOSITORY: aws-ecs-clouda3
    IMAGE_TAG: ${{ github.sha }}
  run: |
    # Build a docker container and
    # push it to ECR so that it can
    # be deployed to ECS.
    docker build --build-arg AWS_ACCESS_KEY_ID=${{ secrets.AWS_ACCESS_KEY_ID }} \
    --build-arg AWS_SECRET_ACCESS_KEY=${{ secrets.AWS_SECRET_ACCESS_KEY }} \
    --build-arg AWS_DEFAULT_REGION=${{ secrets.AWS_DEFAULT_REGION }} \
    --build-arg API_GATEWAY_KEY=${{ secrets.API_GATEWAY_KEY }} \
    -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
    docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
    echo "::set-output name=image::$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG"
```

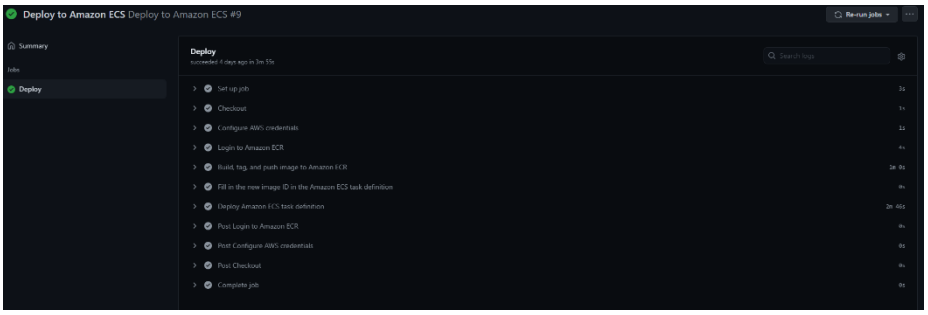
Finally, set the container-name, location of the aws-task-definition.json file, the aws service and cluster names to the ones made on aws in the aws.yml file. As this particular project was a subfolder of the main project the aws-task-definition.json file is in the admin-project sub folder.

Push these files to the git repository, then click ‘Actions’ -> ‘Deploy to Amazon ECS’-> ‘Run Workflow’ -> ‘Run workflow’

```yaml
- name: Fill in the new image ID in the Amazon ECS task definition
  id: task-def
  uses: aws-actions/amazon-ecs-render-task-definition@v1
  with:
    task-definition: admin-project/aws-task-definition.json
    container-name: aws-ecs-clouda3
    image: ${{ steps.build-image.outputs.image }}

- name: Deploy Amazon ECS task definition
  uses: aws-actions/amazon-ecs-deploy-task-definition@v1
  with:
    task-definition: ${{ steps.task-def.outputs.task-definition }}
    service: aws-ecs-clouda3-service
    cluster: ecs-clouda3-cluster
    wait-for-service-stability: true
```



You can then view the deployment information by clicking the Deployment task once it sets to pending:

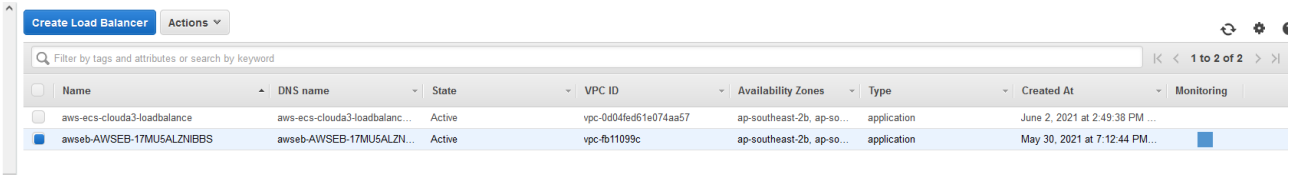


After this deployment step, the admin flask project should be uploaded to admin.cloudfitness.click and should be viewable and working.

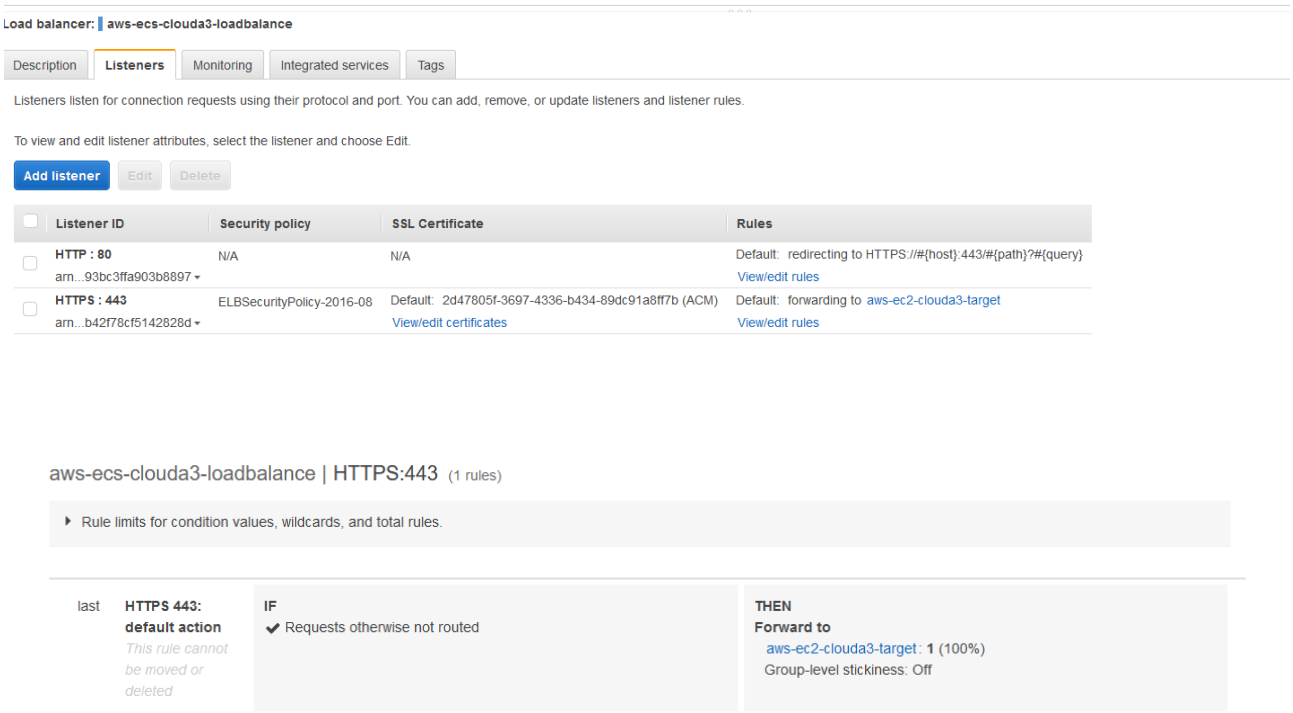
10. **Add SSL / HTTPS to the admin site.**

Before moving onto the next steps, please follow the guide in Section 6**:** Certificate Manager, to create an AWS certificate to use with HTTPS/SSL, if it has not already been creating.

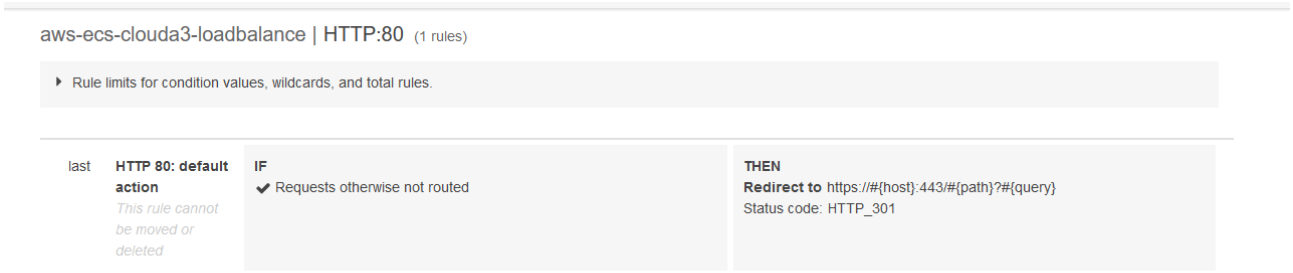
After this step, you can now create HTTPS redirect in the **load balancer** of the ec2 instance which was previously accessed before when creating the load balancer initially.



Select the load balancer which was used for the ec2 instance, then select ‘Listeners’ and create a HTTPS redirect to the target we previously created. Set the SSL certificate to the ARN of the one previously created.



Finally, to ensure that users are always redirected to use HTTPS, redirect the HTTP port 80 to HTTPS.



Now the admin.cloudfitness.click website will always redirect the user to use HTTPS, and should now be fully functional.

Section 2: Main Site *(Elastic Beanstalk / CodePipeline)* [3] [4] [5]

Before beginning this step of development, a functioning cloud fitness flask application is needed to upload to the code-pipeline.

1. **Create the Elastic Beanstalk .ebextensions configuration files.**

Before pushing any code to codepipeline, we first must ensure that the environment we set up on Elastic beanstalk will be the correct configuration and launch the flask website correctly. Furthermore, this project creates the tables in the dynamo DB database, so we also need to add this code to run before the launch. Create a folder in the app directory of the flask site called .ebextensions.

2. **Creating the python.config**

Add a file called python.config to this folder. We will first add some container commands which run just like normal command line interface commands in the container but automatically execute before the flask app deploys. This container command creates the database tables if they do not already exist.

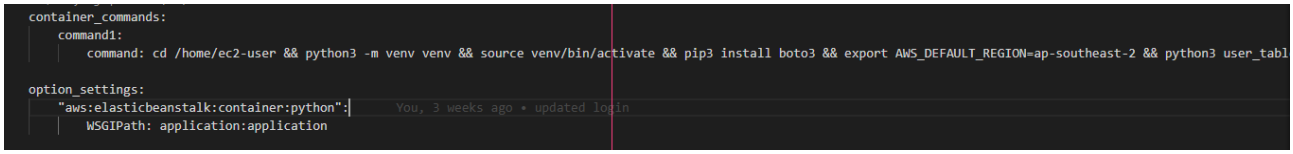
```
container_commands:
    command1:
        command: cd /home/ec2-user && python3 -m venv venv && source
venv/bin/activate && pip3 install boto3 && export AWS_DEFAULT_REGION=ap-
southeast-2 && python3 user_table.py && python3 profile_table.py && python3
comment_table.py && python3 exercise_table.py && deactivate
```

To ensure that the container commands run correctly they should be formatted as follows. It should have the heading **container_commands:** to signify the start of all container commands. The name for this command is **command1**: and the **command:** signifies the command to execute. Each new line should be tabbed over 1 but the command line should have no tabs/new lines in it.

The next step to configure the python.config file is to set the option_settings of the elastic beanstalk environment to know the path of the launch python file for the flask website.

```
option_settings:
    "aws:elasticbeanstalk:container:python":
        WSGIPath: application:application
```

The final format of the file should look similar to this.



Where the commands for command1 are written in a single line.

Creating the securelistener-alb.config

Please refer to this AWS developer guide for more specific configurations

https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/configuring-https-elb.html

We will now configure the https configurations for the elastic beanstalk flask site so that users can access the site with facebook login API.

Create a file in the .ebextensions folder called **securelistener-alb.config** this file name tells the environment this file will configure the application load balancer.

Enter the following details, port 443 directly relates to HTTPS

```
option_settings:
    aws:elbv2:listener:443:
        ListenerEnabled: 'true'
        Protocol: HTTPS
        SSLCertificateArns: arn:aws:acm:ap-southeast-2:059411200951:certificate/2d47805f-3697-4336-b434-89dc91a8ff7b
```

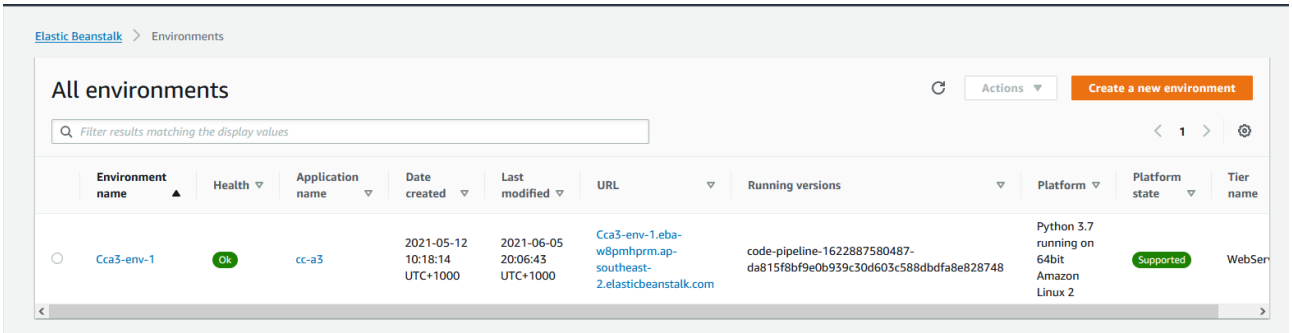
the SSL certificate should already be created from the admin site, please use the same certificate identifier, or if one has not been created, follow the steps in Section 6**:** Certificate Manager**.** Then set the SSLCertificatesArns to the ARN on the certificate.

Finally to ensure that all our traffic is redirected to HTTPS add this code above the option_settings HTTPS redirect we just configured

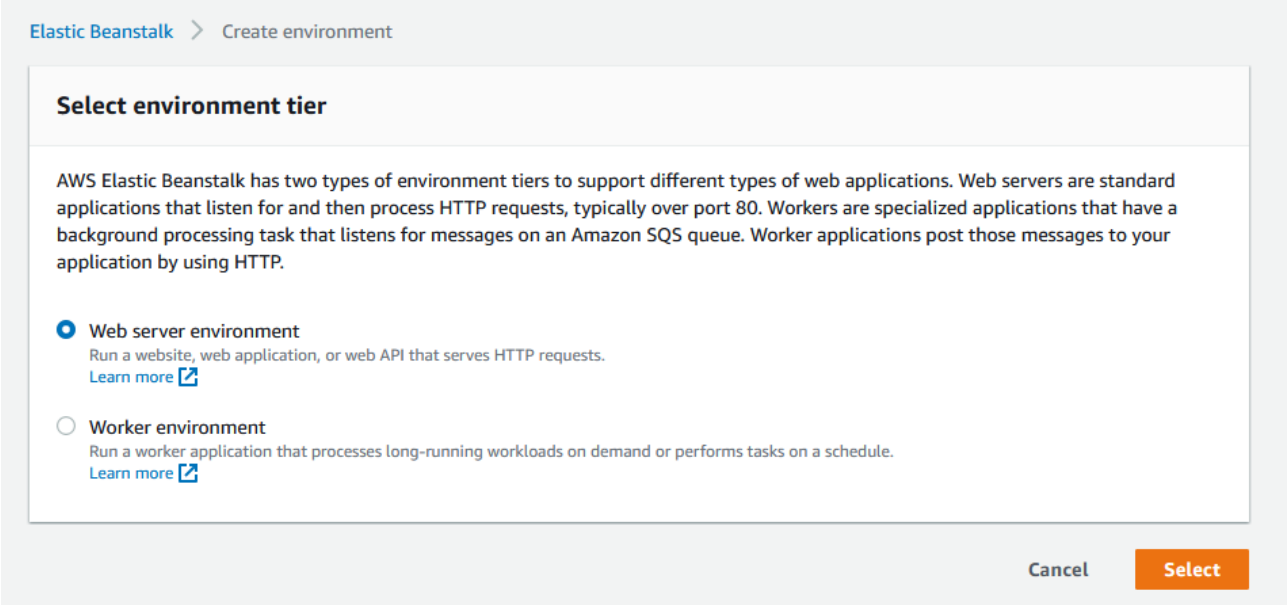
```
Resources:
 AWSEBV2LoadBalancerListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    LoadBalancerArn:
      Ref: AWSEBV2LoadBalancer
    Port: 80
    Protocol: HTTP
    DefaultActions:
      - Type: redirect
        RedirectConfig:
          Host: "#{host}"
          Path: "/#{path}"
          Port: "443"
          Protocol: "HTTPS"
          Query: "#{query}"
          StatusCode: "HTTP_301"
```

This will redirect all HTTP port 80 traffic to port 443 automatically in our application load balancer.

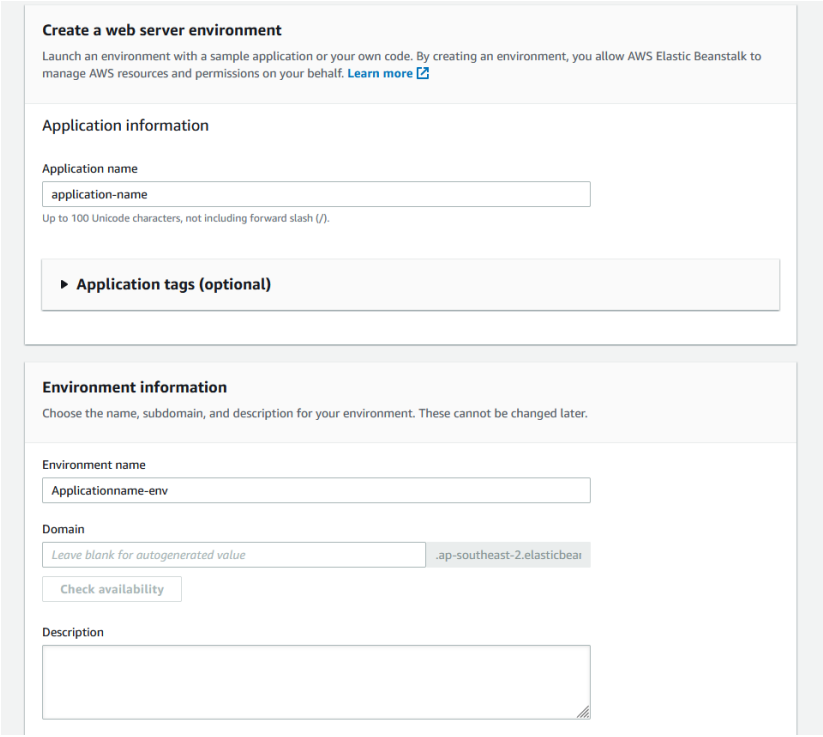
3. **Create the Elastic Beanstalk application environment**
   Go to Elastic Beanstalk in AWS and click ‘Create new environment’



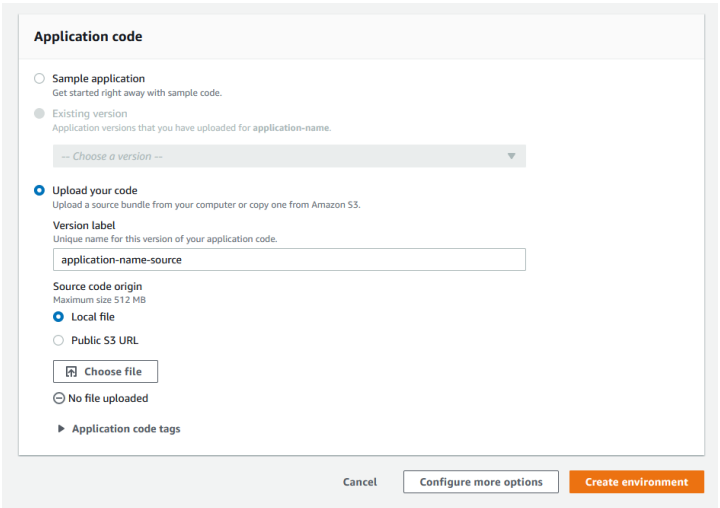
In the ‘select environment tier’ click ‘Web server environment’



The next screen lets you set the application name and environment name. Set these to meaningful values.

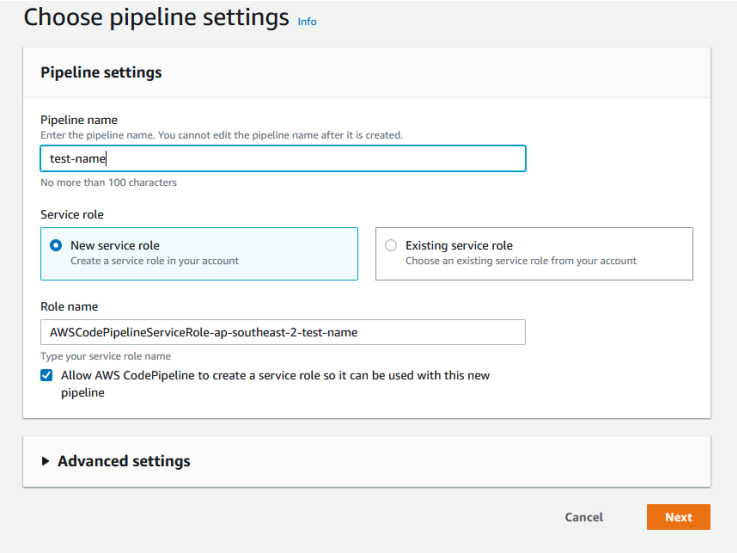


You will now need to upload the initial base code of the application as a zip file. Zip the whole flask application and select ‘Upload your code’, select ‘Local file’ and then upload the zip file project.

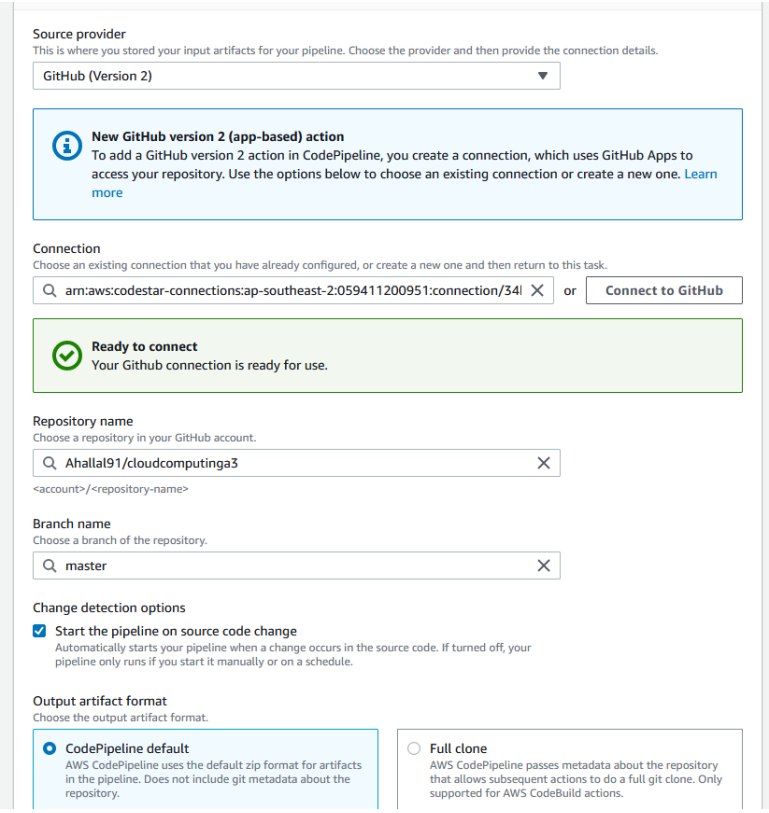


Finally click ‘Create environment’

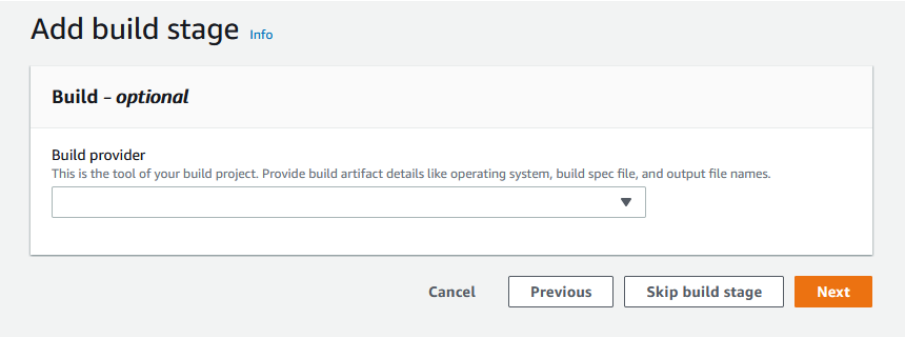
4. **Creating the CodePipeline for the website.**
   Codepipeline will allow us to automatically deploy code to our website in the event of changes, rather than having to re-zip the application each time and upload these zip files. Go to CodePipeline in AWS and click ‘Create Pipeline’, give the pipeline a name and click ‘new service role’ then click ‘next’



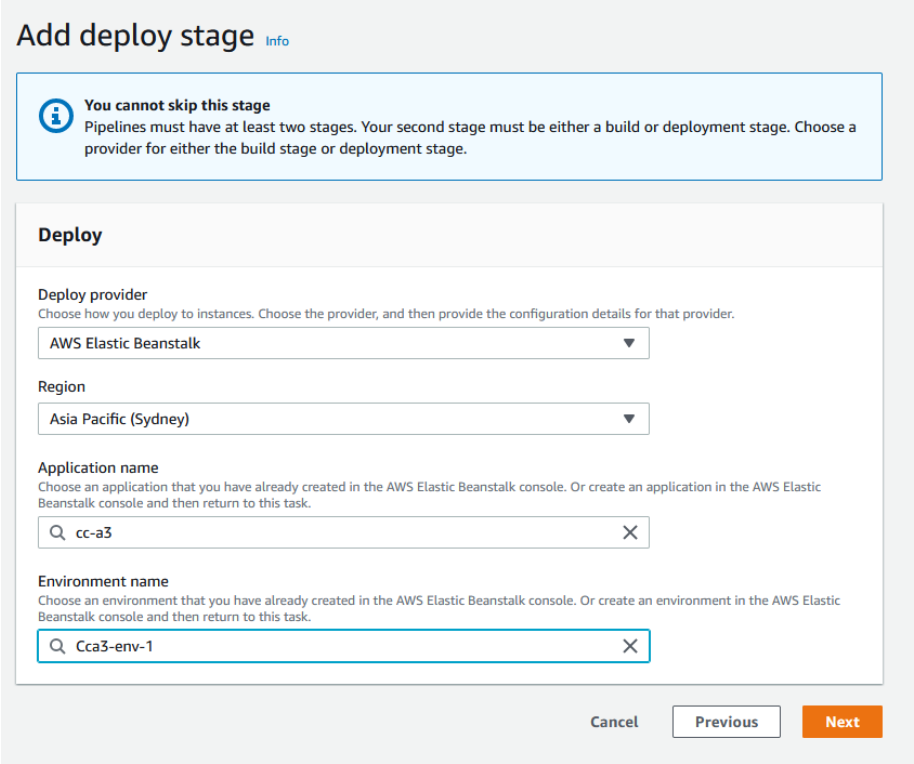
Click ‘GitHub (version 2) as the service provider then connect the repository that holds the code for the flask application in the ‘connection’ and ‘repository name’ area then click ‘master/main’ branch and click ‘next’s



In ‘Add build stage’ click ‘skip build stage’



In ‘add deploy stage’ select ‘AWS Elastic Beanstalk’ as the deploy provider and set the application name and environment name to the one previously created in Elastic Beanstalk



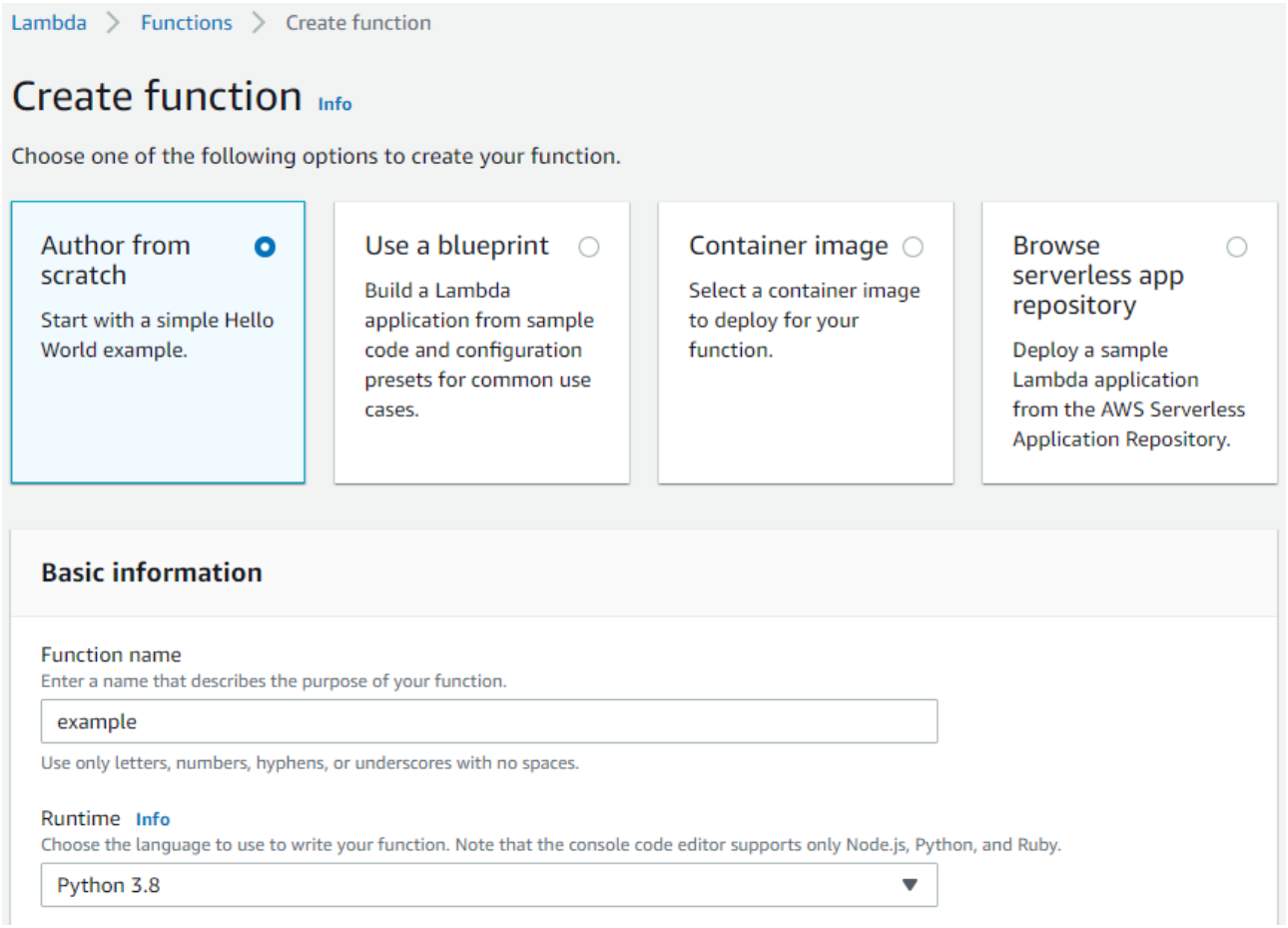
Click ‘next’ then click ‘Create pipeline’ after reviewing the details. This pipeline will now publish to Elastic Beanstalk when GitHub master branch is pushed to or you manually deploy the application.

5. **Connecting Route 53 Domain to the application.**
   See Section n-3b: Route 53 *(Routing traffic – Main site)*.

Section 3: Amazon Lambda

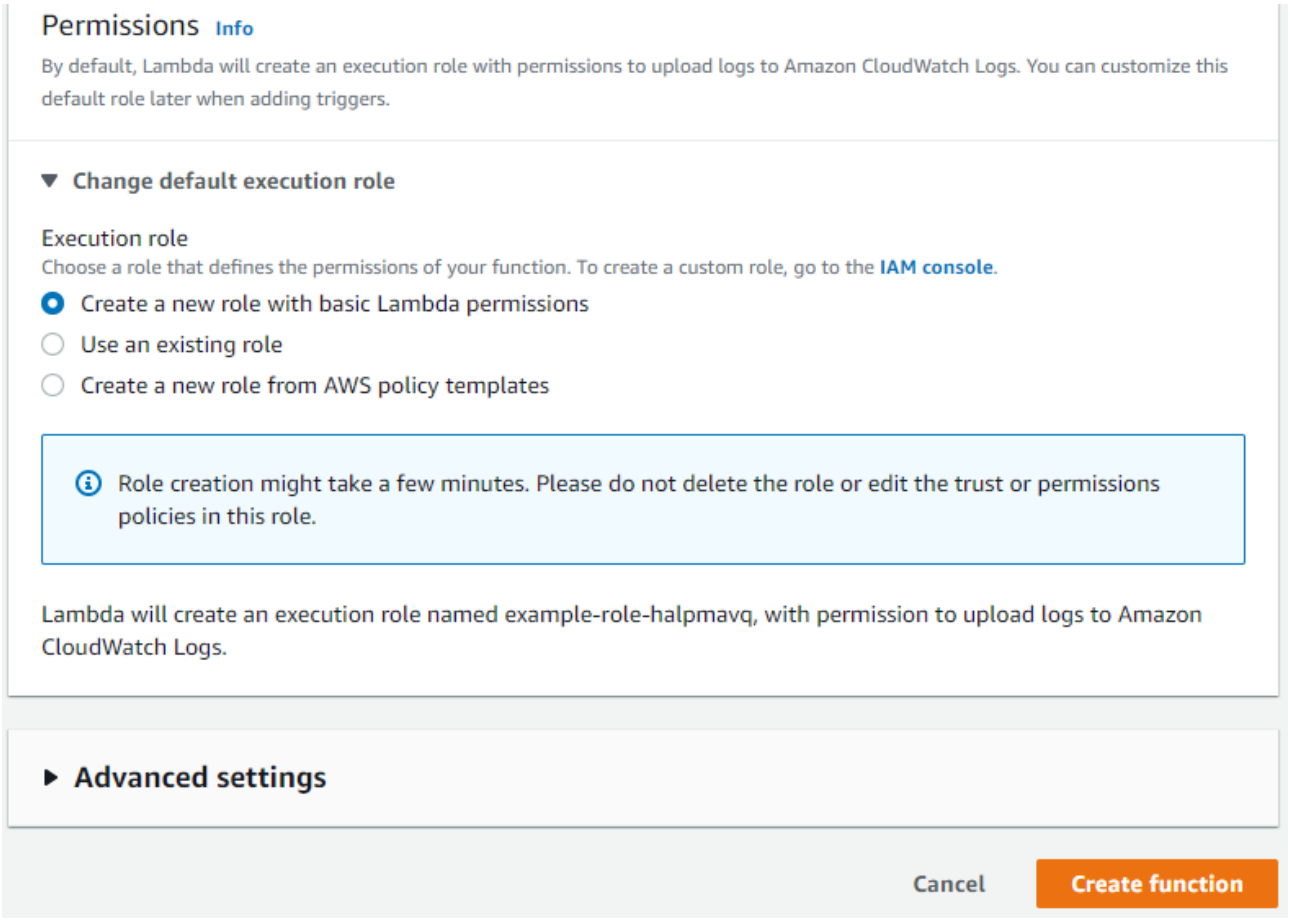
1. Create a lambda function

Navigate to AWS Lamba and click *Create function*. Select "*Author from scratch"* as the function type. Give the function a name and select *Python 3.8* as the runtime.



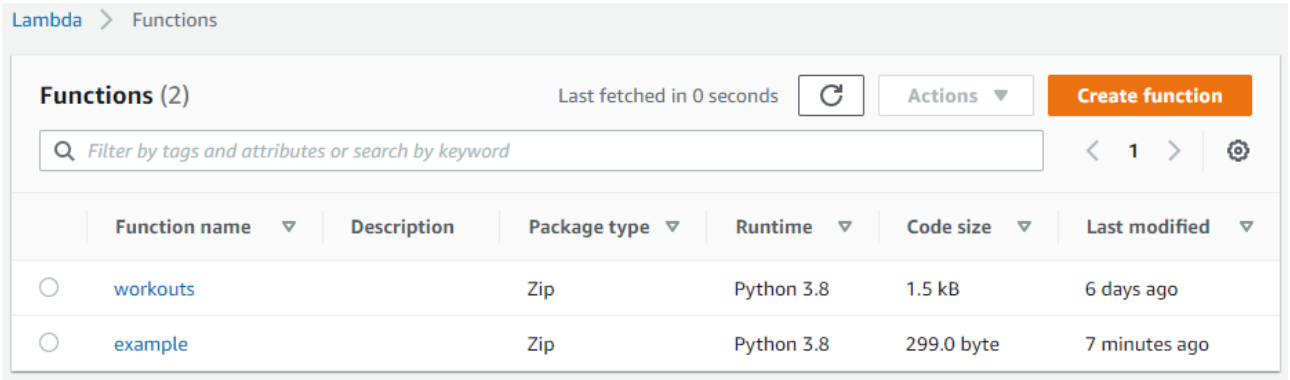
To use AWS Lambda with DynamoDB, the IAM policy *AmazonDynamoDBFullAccess* needs to be attached to the function.

For now, under permissions, select *Create a new role with basic Lambda permissions*. This will create a new IAM role with only the *AWSLambdaBasicExecution* policy attached. Leave all other options as they are and click *Create function*.

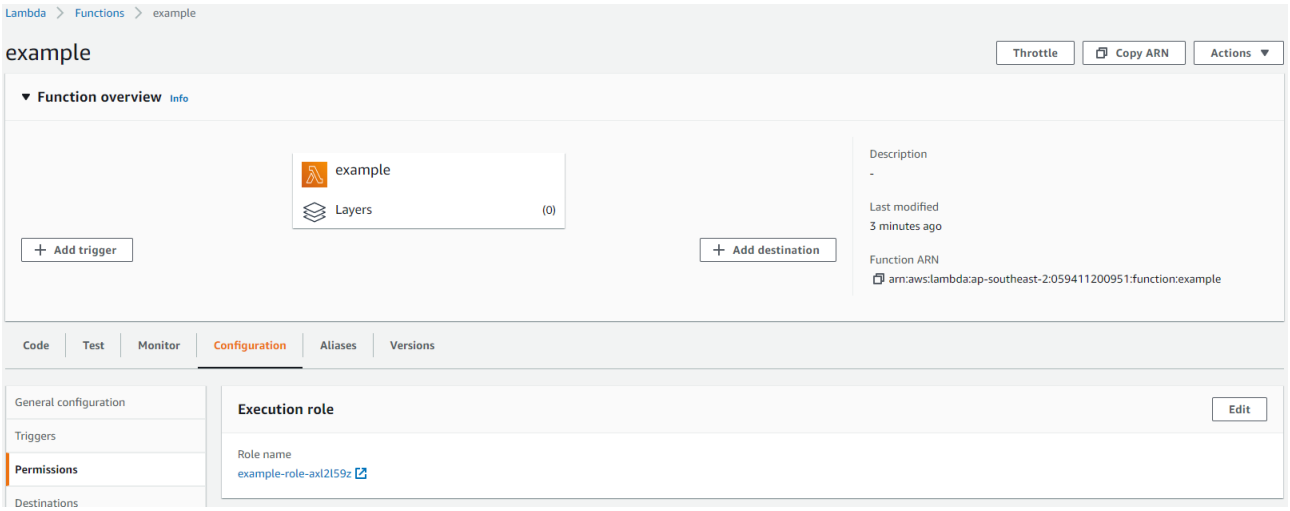


2. Add full DynamoDB access to the lambda's permissions

Select the newly created lambda function from the *Functions* section of AWS Lambda:

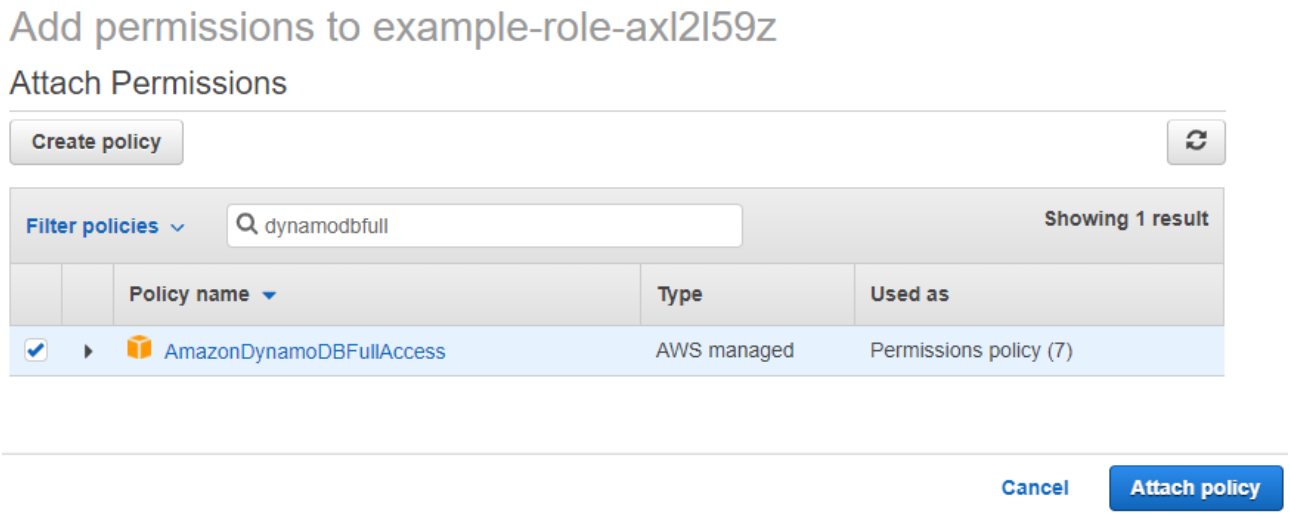


Navigate to *Configuration* → *Permissions* to access the role attached to the function.



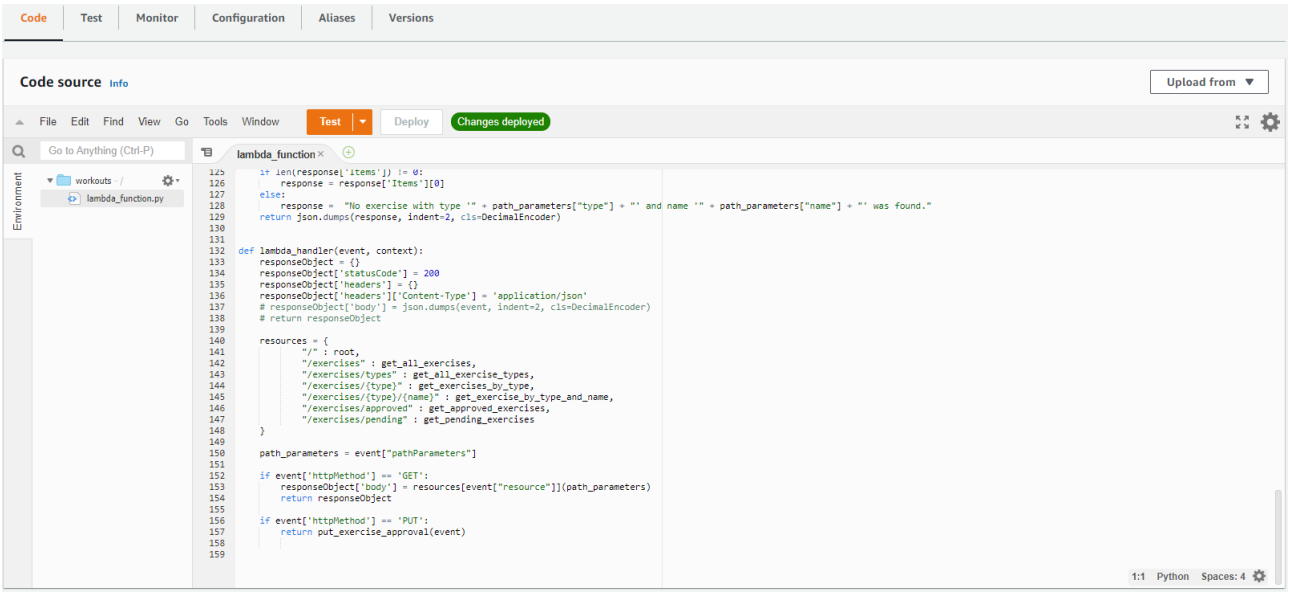
Select the role to be redirected to IAM. Click *Attach policies*:

And search for *AmazonDynamoDBFullAccess*. Click the checkbox on the left and then click *Attach policy*. The Lambda function now has full access to DynamoDB.



3. Edit the lambda function

Select the newly created function again and edit the code as needed. A sample of the lambda function used to control our REST API is shown below:



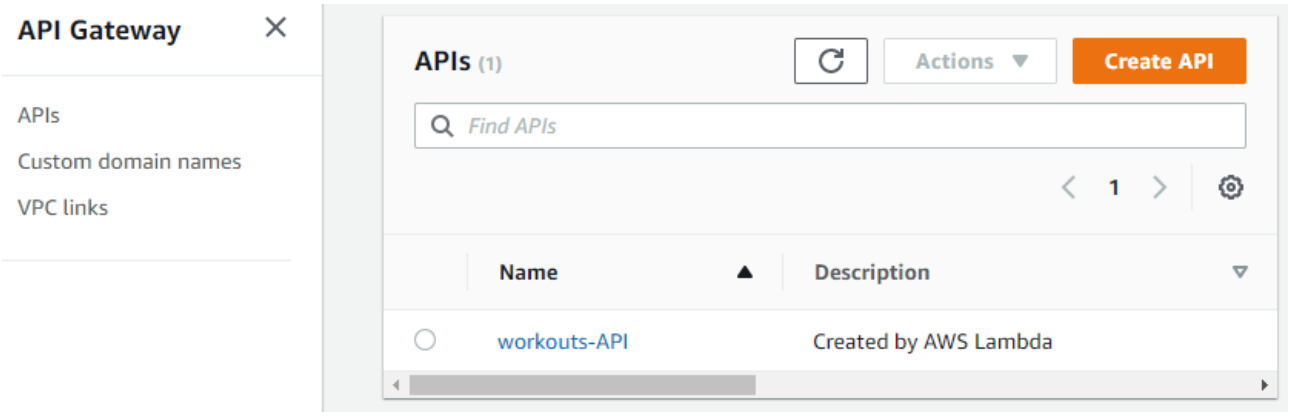
Whenever a change is made to the lambda function, click *Deploy* above the code editing section.

Note that the request information accessed by the lambda function is provided by API Gateway, which is handled in the next section.

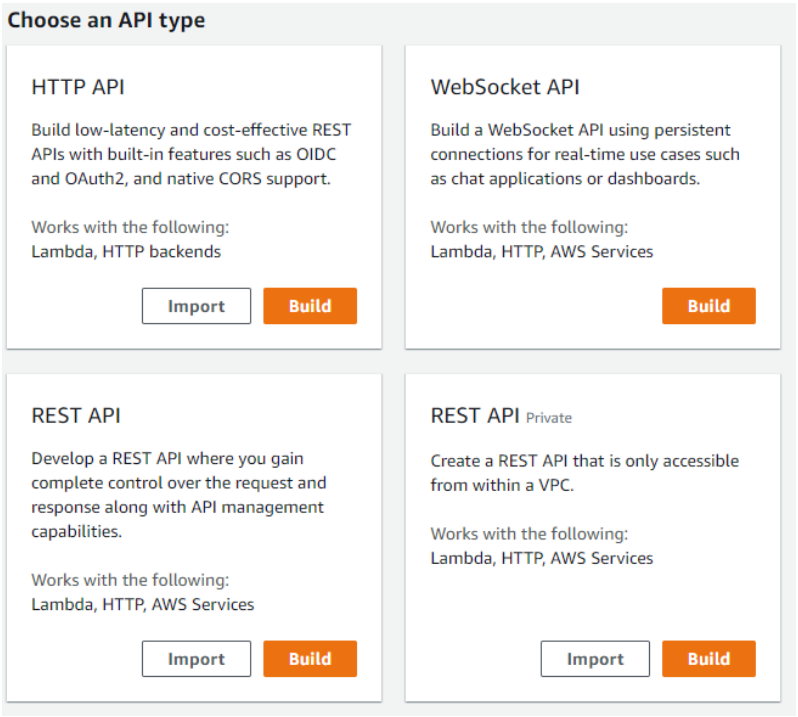
Section 4: API Gateway[6]

1. Create a new API

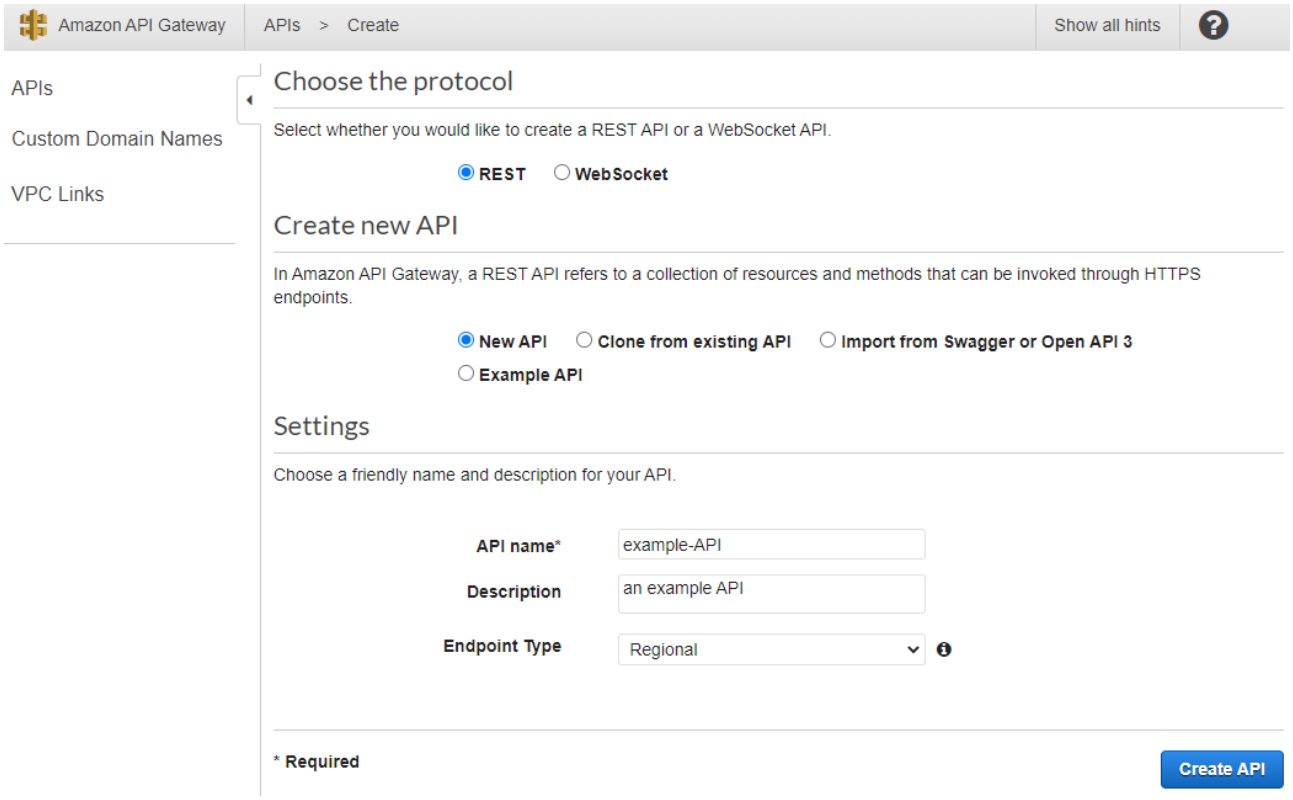
Navigate to AWS API Gateway. To create a new API, click the orange *Create API* button.



Select REST API (*not* private) and click *Build*.



Enter an API name and description. For our purposes, a regional endpoint is used, but an edge optimized API is more suitable for websites accessed globally. Click *Create API*.

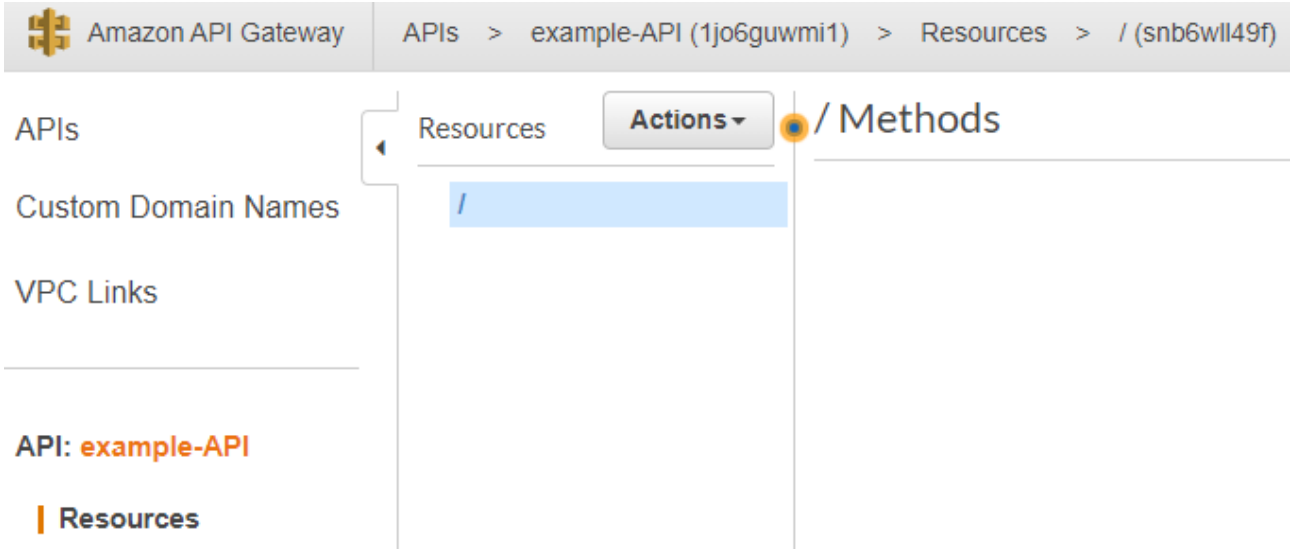


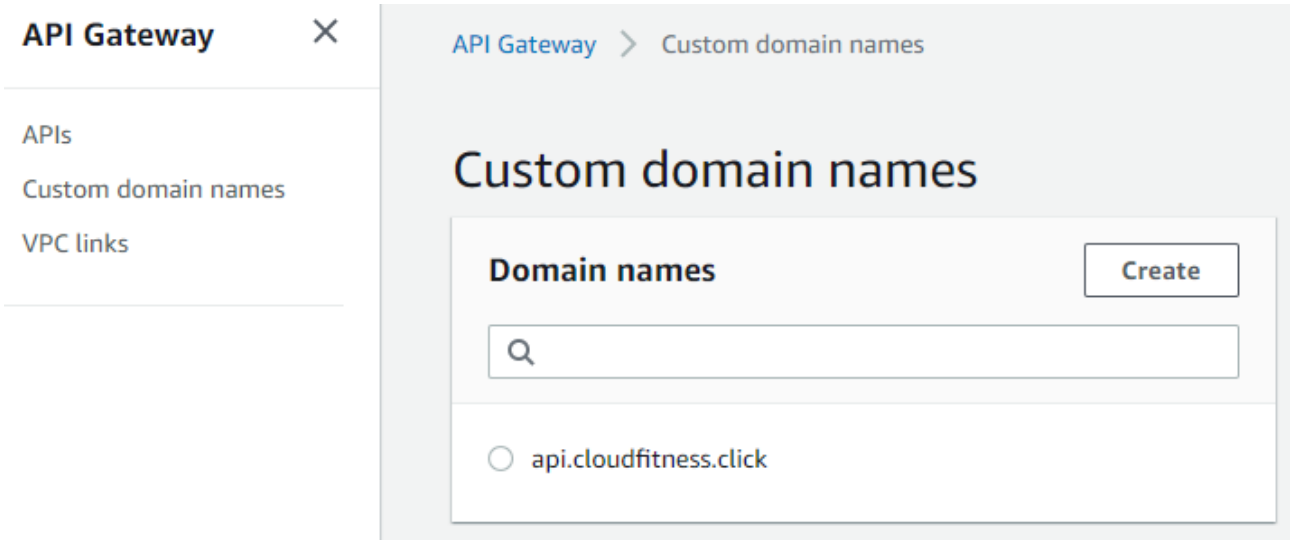
You will be redirected to the newly created API. If not, you can access the API from the AWS API Gateway landing page.

2. Registering a custom domain name

From herein, we will assume that a custom domain has been registered for the API Gateway following the instructions in Section n-1 (Route 53, *Domain name registration*).

To assign a custom domain name to the API, click *Custom Domain Names* in the left menu:





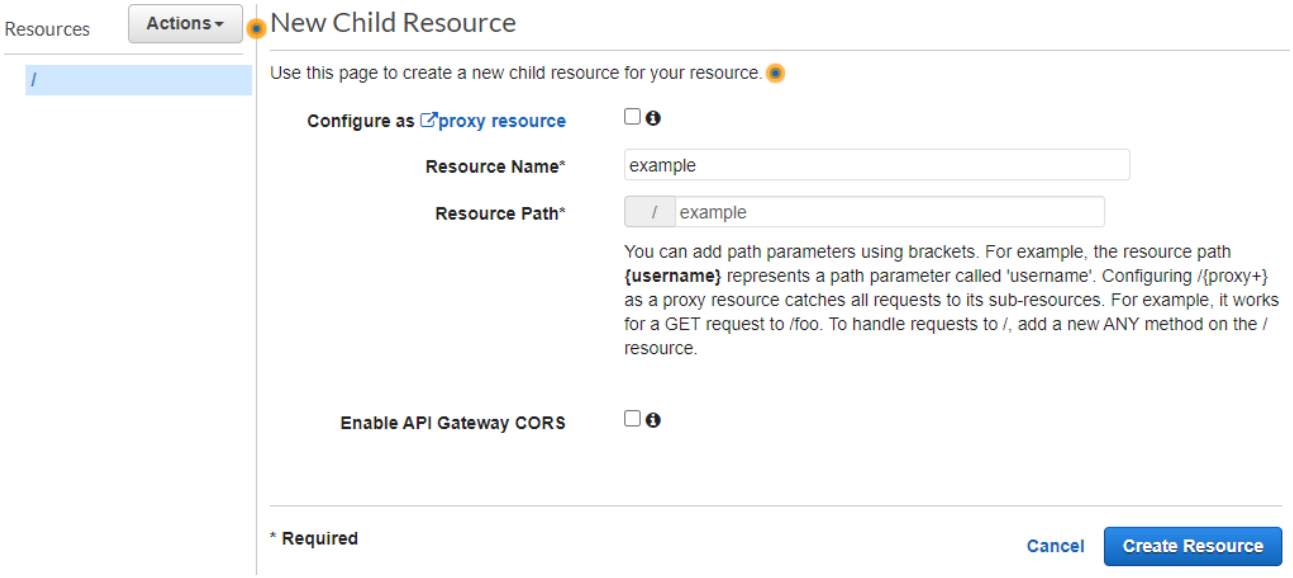
Register the domain name using the search box provided. The domain name entered here should correspond to the subdomain you wish to be routed to as configured in Route 53 (see Section n-3c, Route 53: Routing to the REST API). For our website, the custom domain name given is *api.cloudfitness.click*. This will forward HTTPS requests from the given domain to API Gateway, and that information will be forwarded to the appropriate lambda function as described in the next section.

3. Adding resources

Resources correspond to the URLs visited by users. The root resource "/" corresponds to the base subdomain registered as a custom domain name, such as *api.cloudfitness.com/*. By default, no methods are available at this resource for newly created APIs. Resources are added in a tree structure.

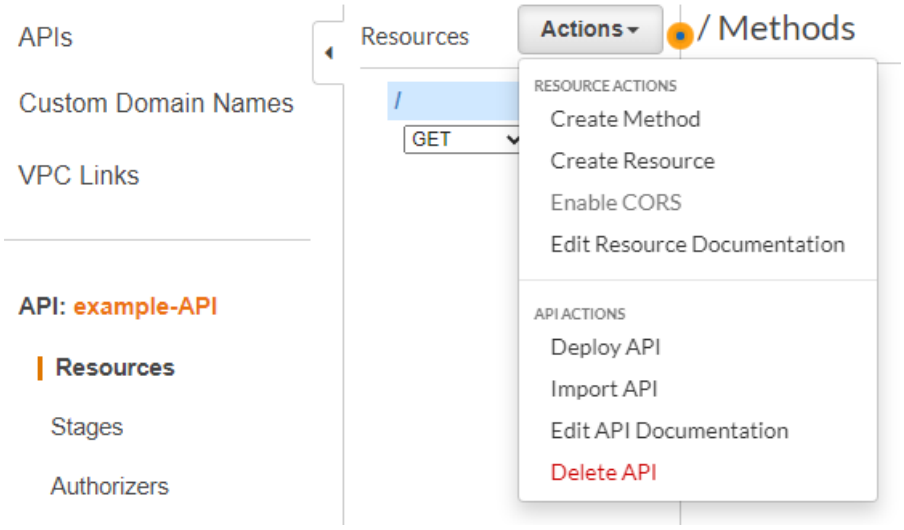
Resources can either be named with an absolute name or a variable name. Variable names are enclosed in { curly braces } and allow users to visit endpoints with variable parameters. For example, our application has a variable resource under /exercises/{type}, which will list all exercises of a certain type if the user replaces {type} with a valid workout type, e.g., /exercises/compound/.

To add a resource, click the *Actions* dropdown menu and select *Create Resource*. Do not configure the resource as a proxy resource. Give the resource a name, which typically matches the value entered as a resource path. If the resource path should be a variable resource path, enclose the value in curly braces, i.e., "{example}" rather than "example". Leave *Enable API Gateway CORS* unchecked and click *Create Resource.*

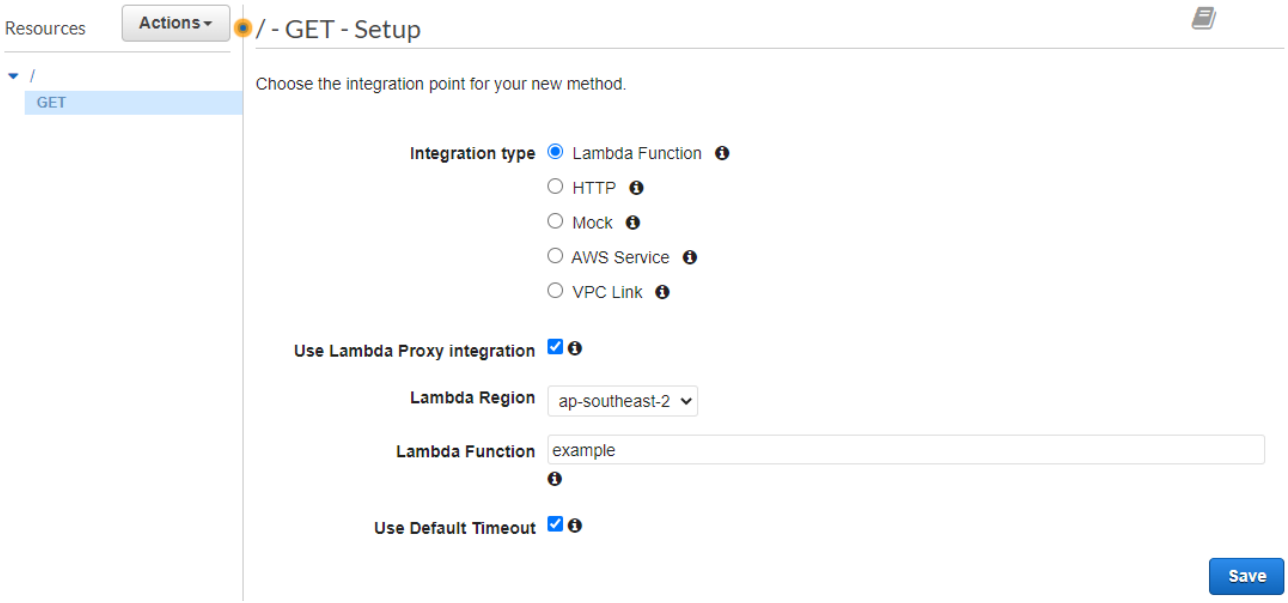


4. Adding methods to resources

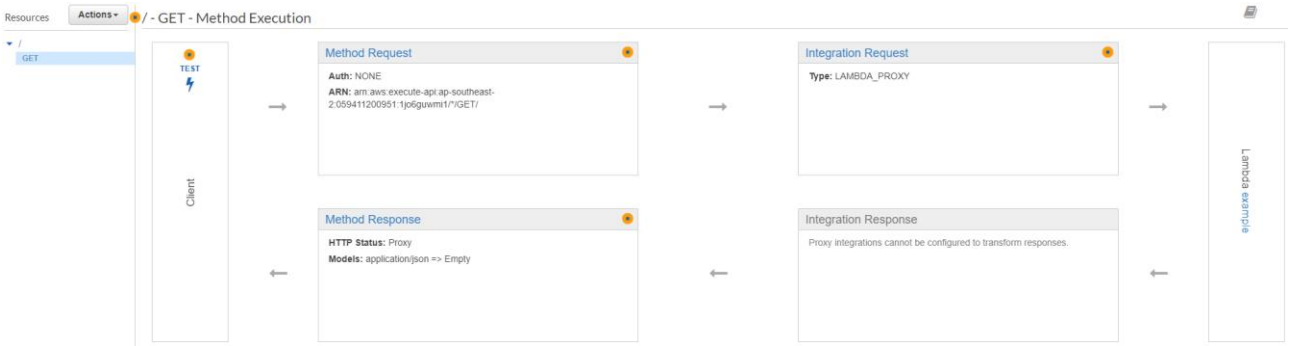
To add a method to a resource, click the *Actions* dropdown and select *Create Method*. A dropdown will appear underneath the resource name, where you can select HTTP methods such as GET, POST, PUT, and so on. Select GET.



Select *Lamba* function as the integration type, and tick *Use Lambda Proxy integration*. By selecting *Use Lambda Proxy Integration*, the entire HTTP request is passed to the lambda function, allowing all headers to be accessed in the lambda (see Section 3-3 (Amazon Lambda: Edit the lambda function) for an example usage). Select the appropriate region and the lambda function that should receive the request information and leave *Use Default Timeout* checked. Click *Save and* click *OK* when prompted to *Add Permission to Lambda Function*.

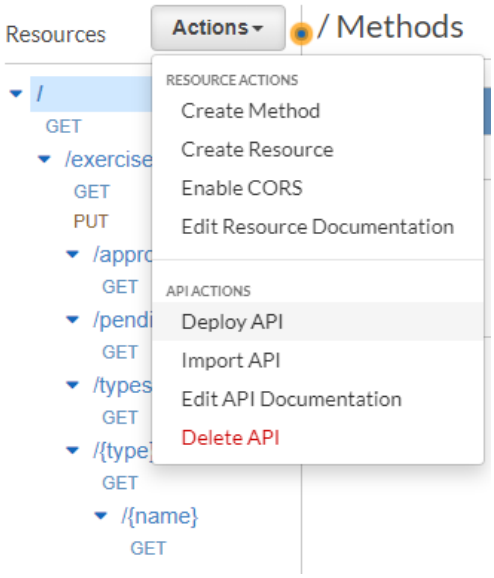


The GET method under / will now contain a method execution pathway. Note the lambda we selected on the right side of the diagram.

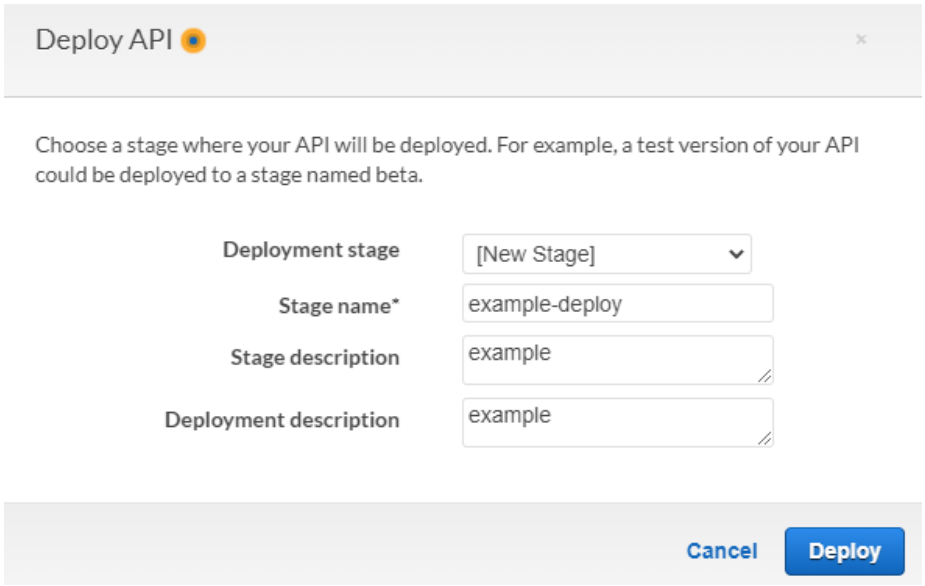


5. Create and deploy the final API

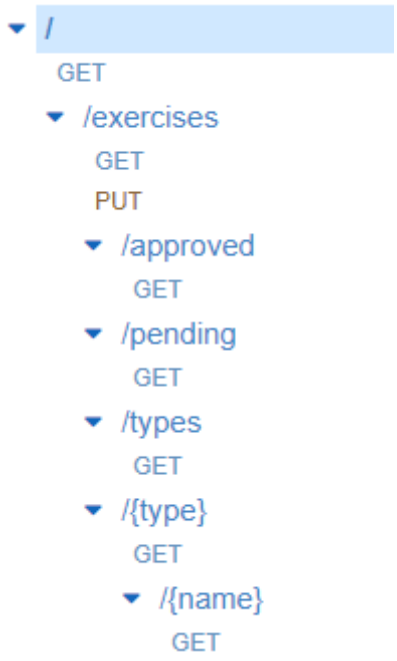
To deploy the API, click the *Actions* dropdown and select *Deploy API*.



On the first deployment, a new deployment stage must be selected. On future deployments, this stage can be reused. Whenever a change is made to the API, ensure that the API is deployed.



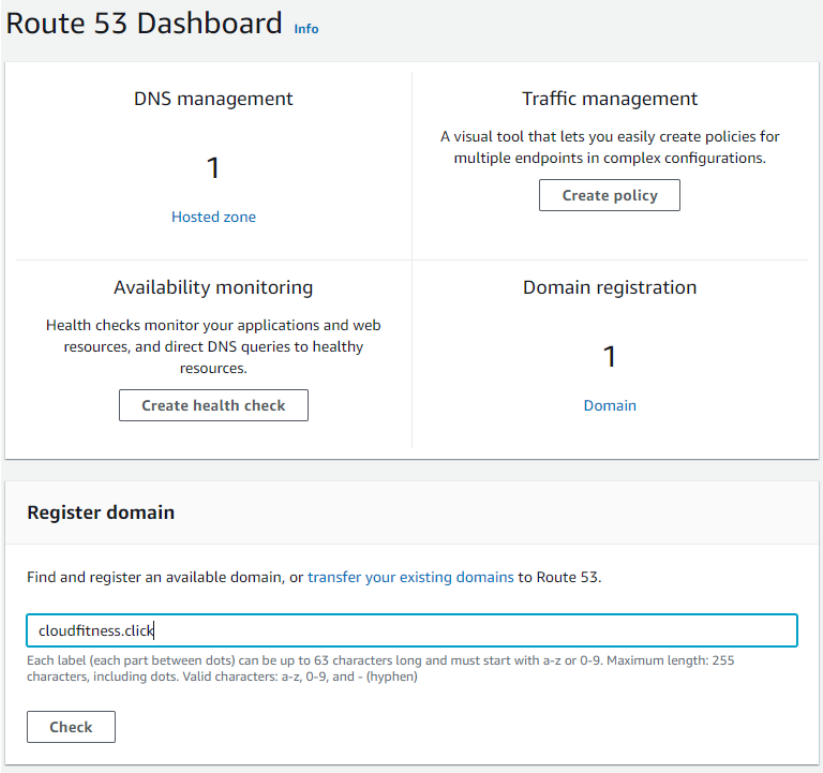
The final API Gateway setup for our application is shown below. Each resource and method are configured as described above. Note that the /exercises resource also contains a PUT method used via the admin site.


/
GET
/exercises
GET
PUT
/approved
GET
/pending
GET
/types
GET
/{type}
GET
/{name}
GET

Section 5: Route 53

1. Domain name registration

Navigate to the Route 53 Dashboard and search for a domain name using the search bar contained under the section “Register domain”:



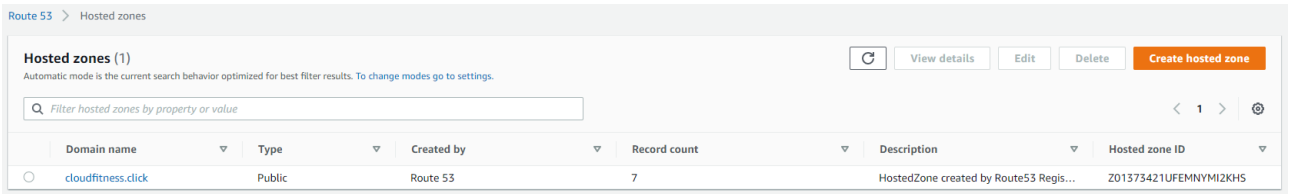
If the domain is available, purchase the domain and wait for DNS propagation to indicate that the purchased domain is useable. We recommend purchasing *.click* domains since they are the cheapest at $5 AUD.

2. Setup HTTPS certification

For use with API gateway, the domain name must be usable under HTTPS. See **Step 10** (*Add SSL / HTTPS to the admin site*) of *Section 1: Admin Site* to see how this is done.

3. Routing traffic

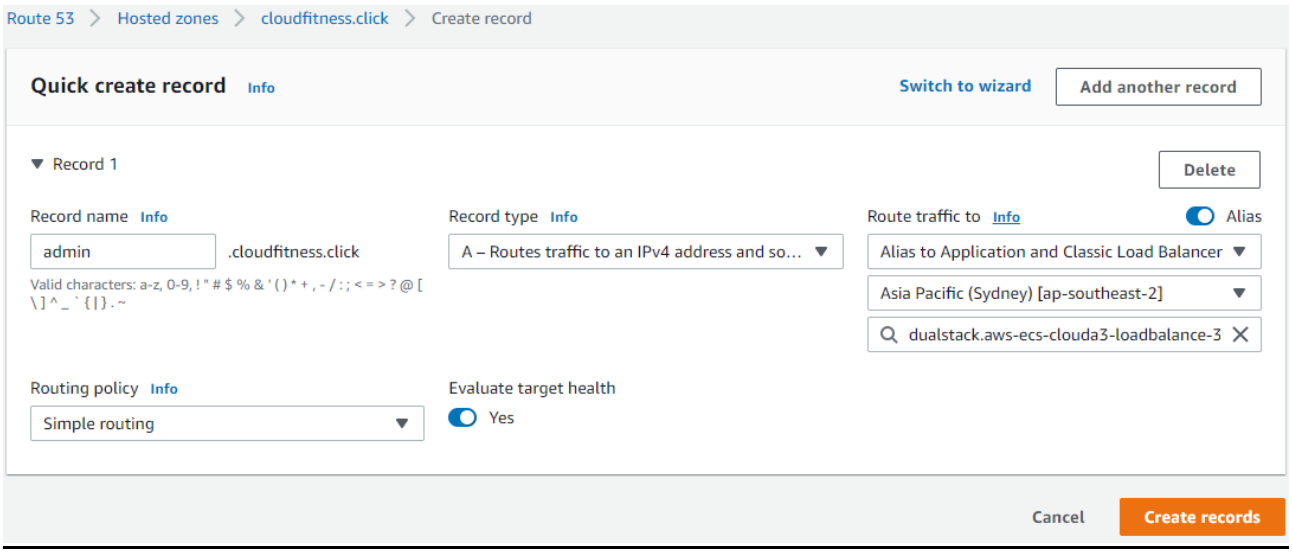
Using the side panel, navigate to “Hosted zones”, which should list the domain registered above:



Select the domain name, and then click *Create record.* A separate record is created for the admin site, the main site, and the REST API handled by API Gateway.

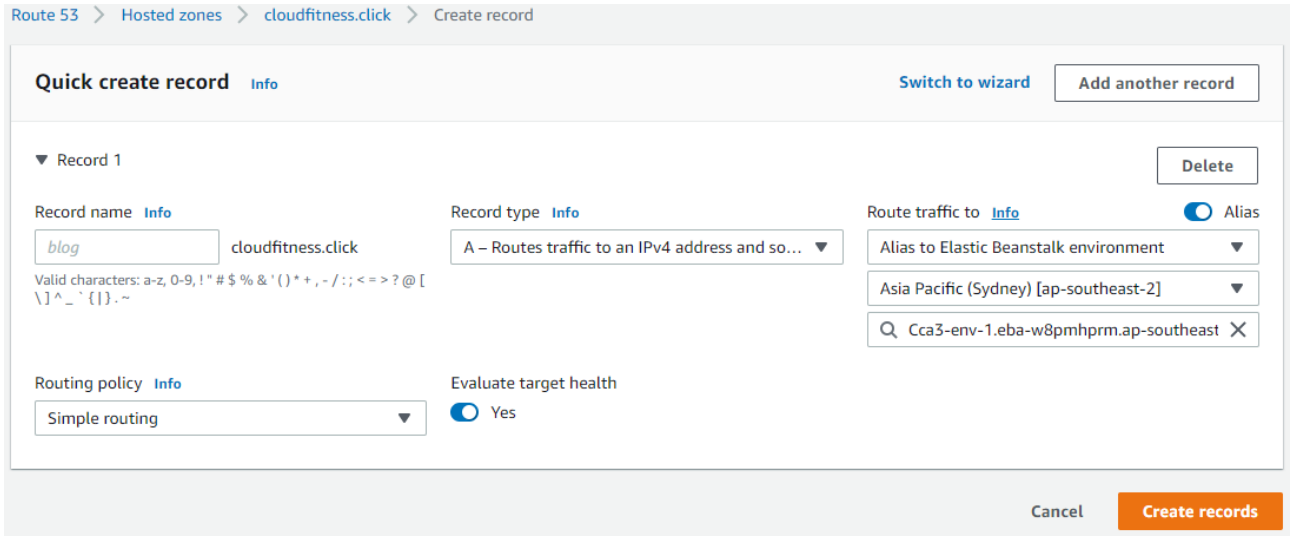
a. Routing the admin site

Enter “admin” under *Record name* to create a record for *admin.cloudfitness.click*. Select “A” as the record type. Check the *Alias* slider next to ”Route traffic to” and select “Alias to Application and Classic Load Balancer”. Select the appropriate region and the application load balancer created in section 1. Leave all other options as they are.



b. Routing the main site

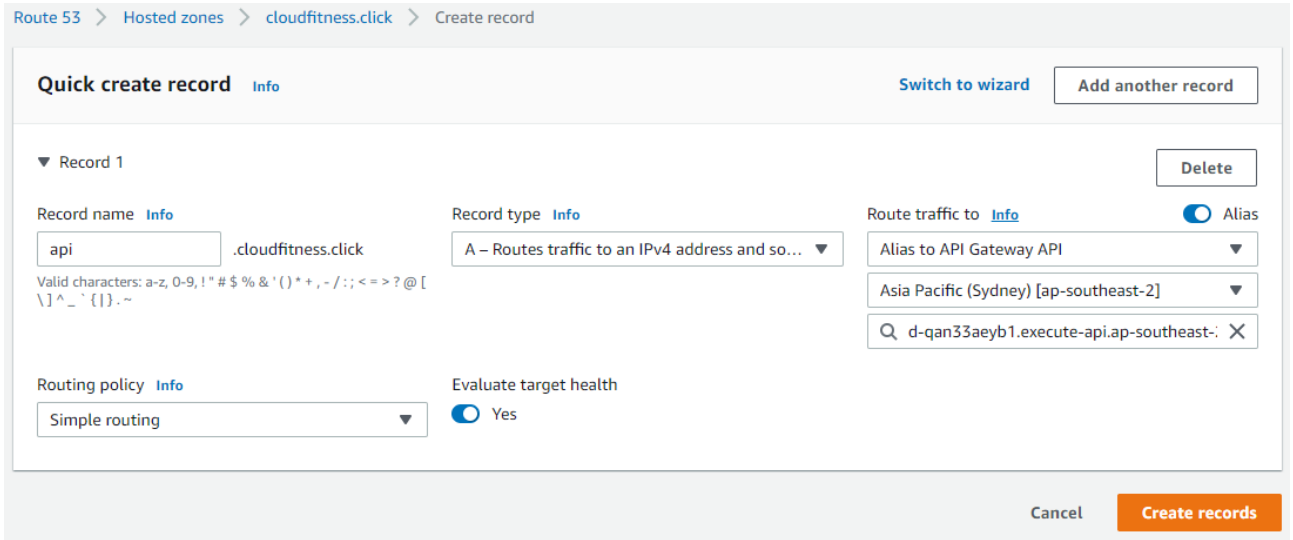
Leave the record name blank to create a record for *www.cloudfitness.click*. Select “A” as the record type. Check the *Alias* slider next to ”Route traffic to” and select “Alias to Elastic Beanstalk Environment”. Select the appropriate region and the application load balancer created in section 1. Leave all other options as they are.



c. Routing to the REST API

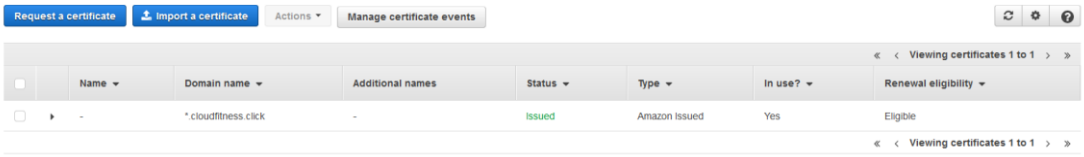
Enter “api” under *Record name* to create a record for *api.cloudfitness.click*. **Ensure this is the same name as the custom domain name specified for the corresponding API in API gateway.** Otherwise, the API will not appear as a selectable endpoint.

Select “A” as the record type. Check the *Alias* slider next to “Route traffic to” and select “Alias to API Gateway API”.

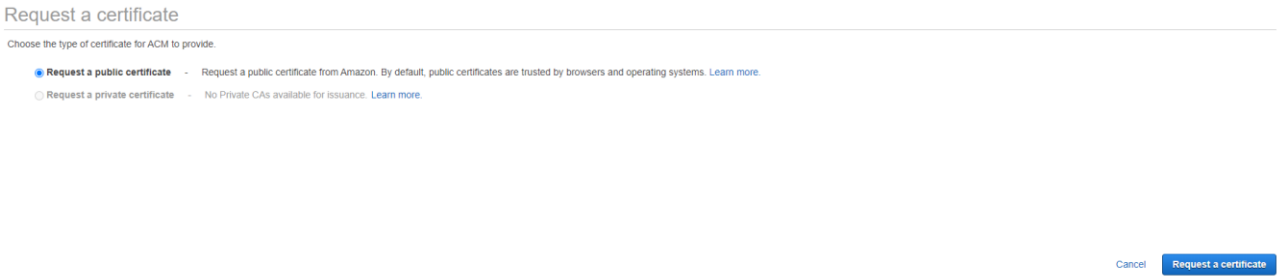


Section 6: Certificate Manager

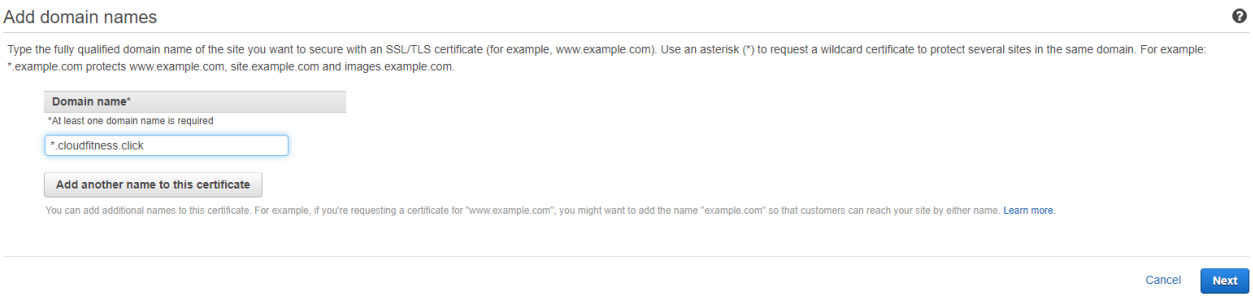
To create a certificate that will allow the use of HTTPS/SSL on the sites go to ‘Certificate Manager’ in the aws console. Click ‘request a certificate’.



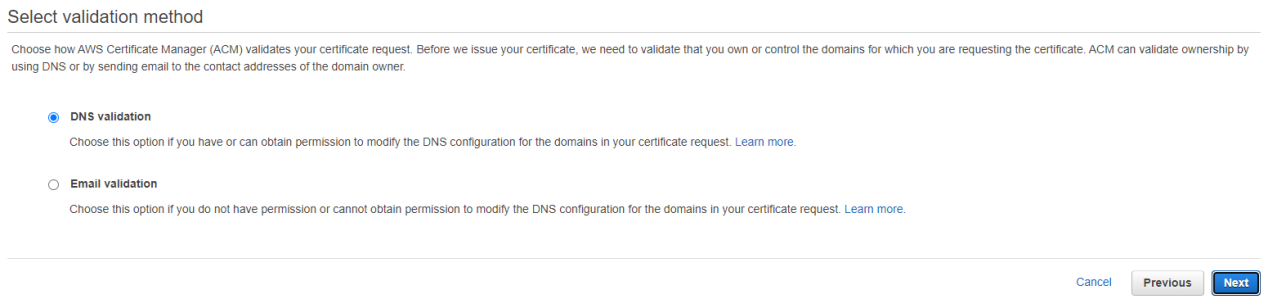
Then click ‘request a public certificate and click ‘Request a certificate’



Fill in the domain name for the certificate as *.cloudfitness.click and click ‘Next’. (this * before the domain name signifies that the certificate can be for any type of doman name [name].cloudfitness.click)



Then click ‘DNS’ validation. Click ‘Next’.

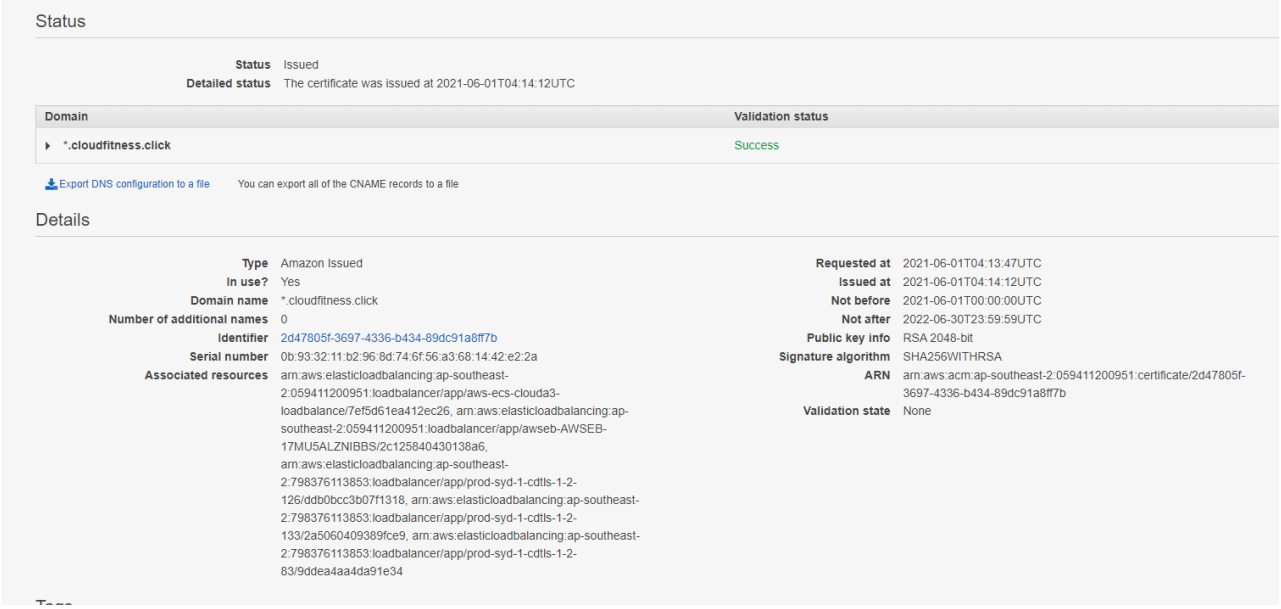


We will not add any tags so skip the ‘Add tags’ section and click ‘Review’. Then in the validation section press ‘Create a record in Route 53’. This adds the Certificate to route 53

automatically and validates the certificate. It should appear as a record of type CNAME in Route 53, so you can verify this.

☐	_cbf3026d53e06343...	CNAME	Simple	-	_a5da253cb1f7ea090e88b7c70617f42b.xrchbtpdjs.acm-validations.aws.

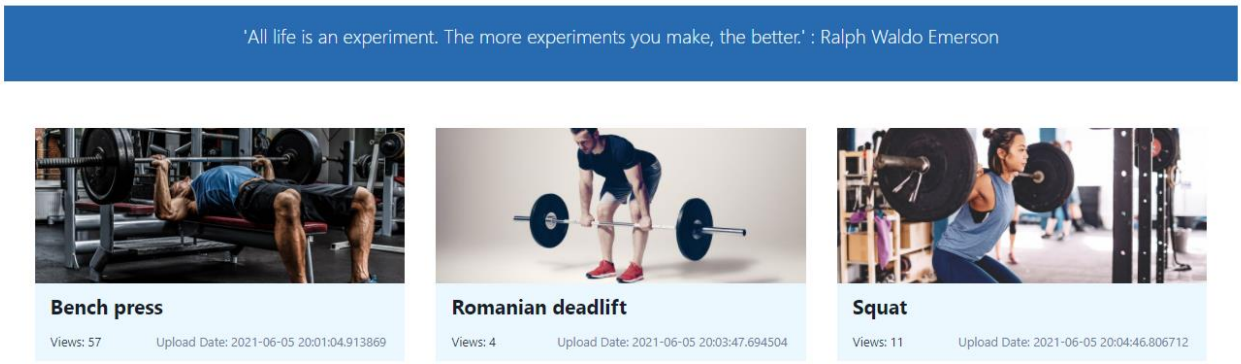
When adding HTTPS/SSL to applications you will need to copy the ARN from the certificate in the certificate manager to enable the use of the certificate.



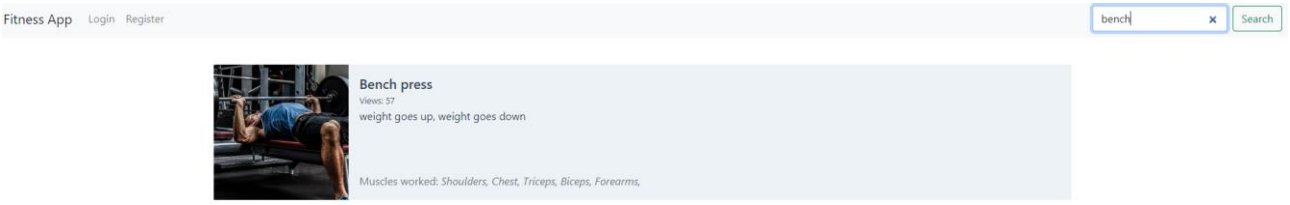
User Manual

Main site

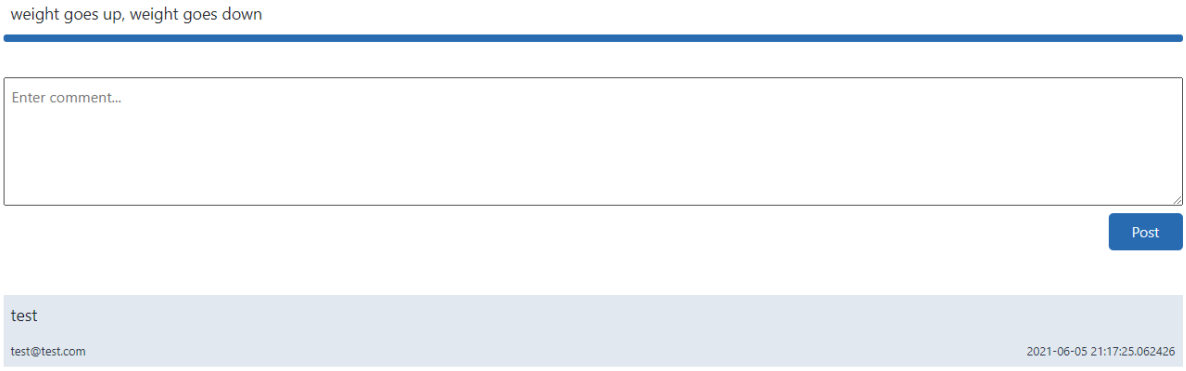
Viewing exercises: Users can view exercises without being a registered user or by being registered. They can see a brief overview of each exercises via the home page:



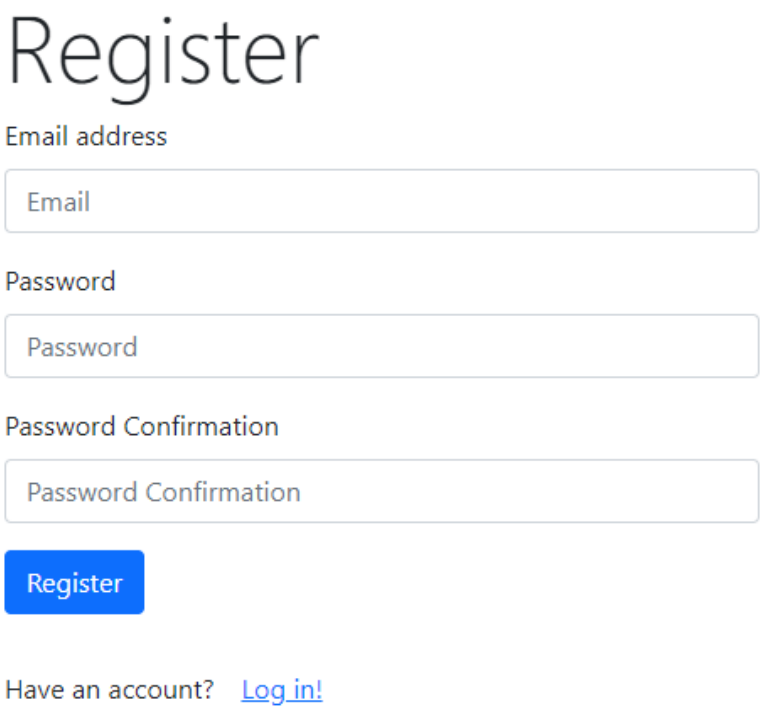
Or via the search function:



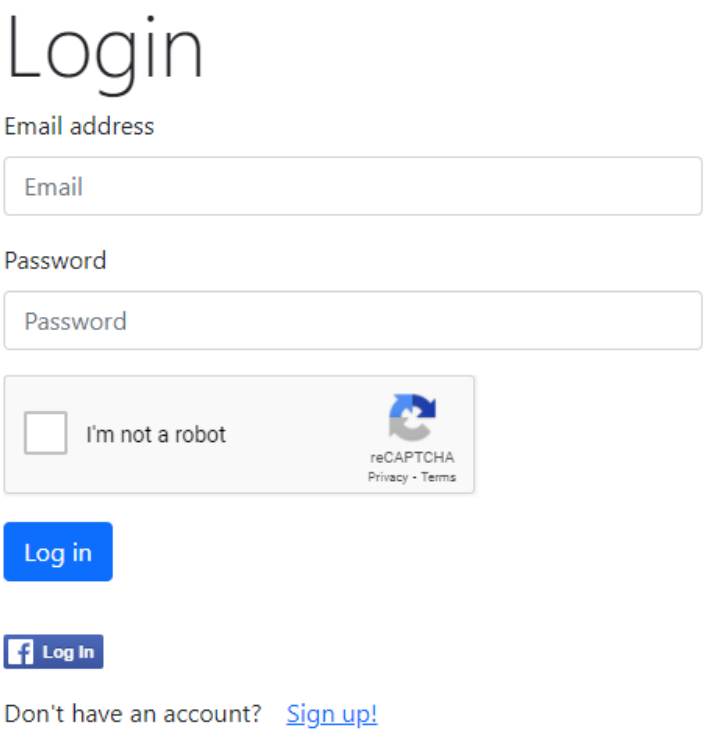
Viewing exercise detail: Users can click the exercises from the front page or search to be redirected to the main exercise information page with comments, likes and views all viewable by the user. Logged in users can add exercises to their profiles from this page.



User Register: To register, users must enter an email address and password with password confirmation.



User login: Users can login via their facebook accounts if they do not wish to register a new account or login with an account they registered with. They will need to complete the reCAPTCHA before being allowed to login.



User exercise upload after login: Users can upload exercises under two main types, ones which will only be viewable by the user on the personal page, and others which can be uploaded to the main site but need to be approved by an admin first.

Upload Exercise

Name

Type of exercise

Compound

Exercise Level

Beginner

Muscle groups (check all that apply)

Neck	Traps	Shoulders	Chest	Triceps	Biceps	Forearms	Abdominals	Middle back	Lats	Lower back
Glutes		Quadriceps		Hamstrings			Calves			

Description

Video URL (optional)

Exercise Image (optional)

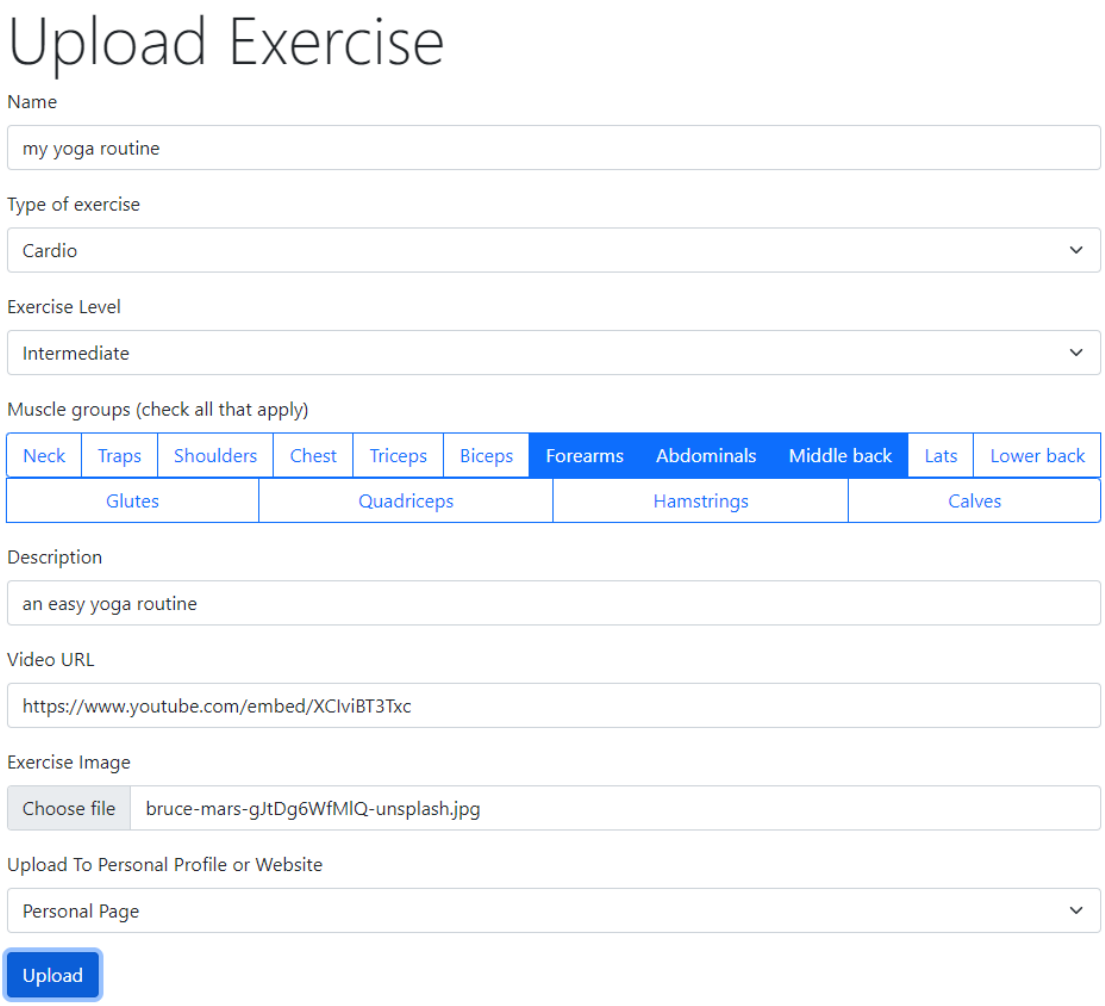
Choose file | No file chosen

Upload To Personal Profile or Website

Upload To Website

Personal Page

Upload To Website



User personal profile: Users can view personally uploaded exercises or exercises which have been added to the main site and then to their profile. To distinguish these different exercise categories, users a prompted by a bin icon for their own personal exercises if they choose to delete them they will be permanently removed. However, exercises on the main site will still stay on the site itself.

Personal profile exercise: Personal profile exercises do not have views/likes or comments.

Logout: users can easily logout by clicking the logout button on the top of the page.

API Access

Outside users can access the public api via api.cloudfitness.click and use the data for their own personal use.

GET: https://api.cloudfitness.click/exercises to get list of all exercises

GET: https://api.cloudfitness.click/exercises/approved to get list of all approved exercises.

GET: https://api.cloudfitness.click/exercises/pending to get list of all pending exercises.

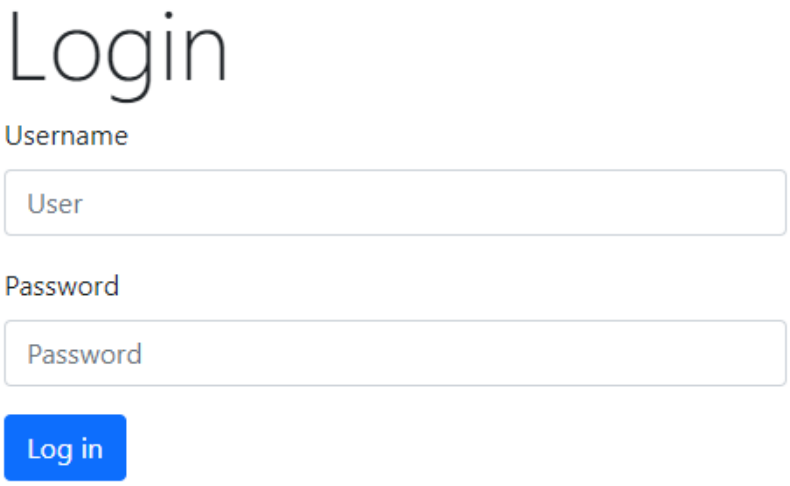
GET: https://api.cloudfitness.click/exercises/types to get a list of types you can pass through to get specific exercises.

GET: https://api.cloudfitness.click/exercises?type={type}&name={name} to get a specific exercise

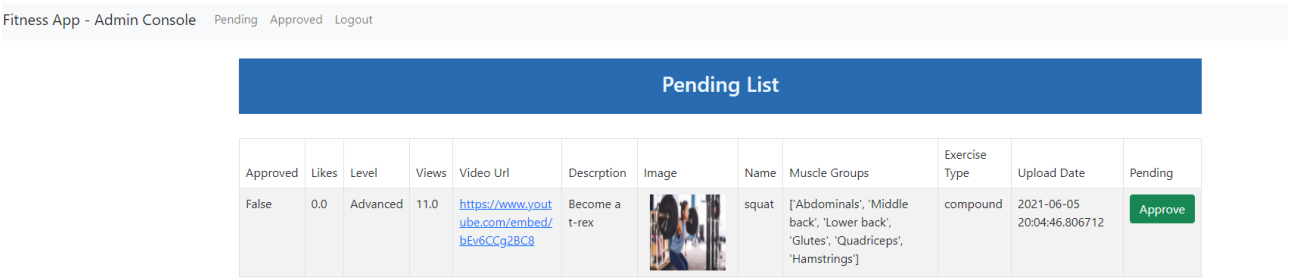
Normal users cannot access PUT requests which is only available to admin with the use of an API key.

Admin Access

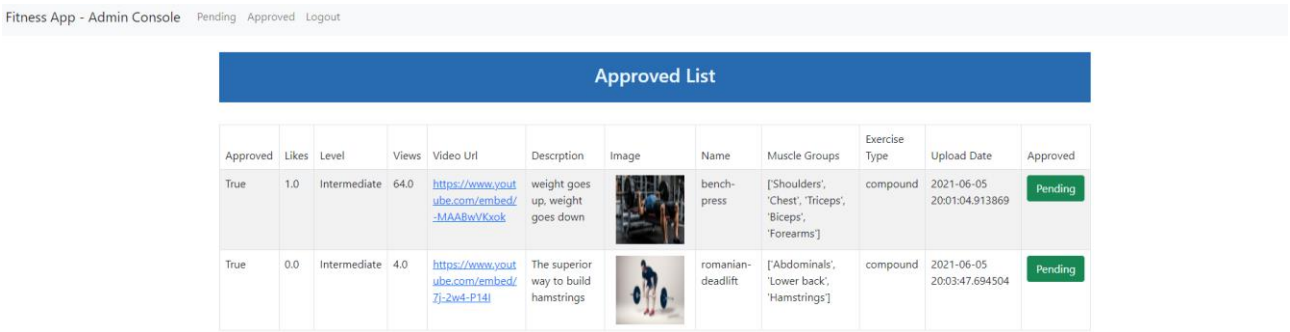
Login: to use the admin functions of the admin site (user: admin, password: password) all admins must login.



View Pending exercises: To approve pending exercises, admins must visit the pending tab and view the exercise information in the table. They can click approve if they wish the exercise to be displayed on the site.



View Approved exercises: If admins wish to change approved exercises back to pending to remove them from the main site, they can visit the approved tab and select the ‘Pending’ button on the appropriate exercise.



References

[1] Healey, "'Sweat tech' is on the rise, with 39% of Australians turning to digital platforms, apps and at-home fitness technology in the last year", *Business Insider Australia*, 2021. [Online]. Available: https://www.businessinsider.com.au/sweat-tech-fitness-apps-australia-2021-3. [Accessed: 05- Jun- 2021].

[2] Alex Damiani. *Flask to AWS ECS Series.* (Jan. 3, 2021). Accessed: June. 1, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=kqa_cchAMLY&t=24s

[3] Cloud Path. *AWS CodePipeline tutorial | Build a CI/CD Pipeline on AWS.* (Apr. 29, 2019). Accessed: May. 10, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=NwzJCSPSPZs

[4] AWS. *Getting started using Elastic Beanstalk.* (2021). Accessed: May. 2, 2021. [Documentation]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html

[5] AWS. *Configuring an Application Load Balancer.* (2021). Accessed: May. 5, 2021. [Documentation]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environments-cfg-alb.html

[6] Implementing a REST API with AWS (API Gateway, Lambda, and DynamoDB) (2020). Accessed: May 31. 2021. [Online]. Available: https://levelup.gitconnected.com/implementing-a-rest-api-with-aws-api-gateway-lambda-and-dynamodb-c62b8a1f6182