Member-only story

Deploying a Docker container with ECS and Fargate.

Including a guide to the necessary permissions.

Ben Bogart · Follow

Published in Towards Data Science · 13 min read · Jul 4, 2021

Photo by Dominik Lückmann on Unsplash

This week I needed to deploy a Docker image on ECS as part of a data ingestion pipeline. I found the process of deploying the Docker image to ECS to be fairly straightforward, but getting the correct permissions from the security team was a bear.

In this article, we will dig into the steps to deploy a simple app to ECS and run it on a Fargate Cluster so you don't have to worry about provisioning or maintaining EC2 instances. More importantly, we'll take a look at the necessary IAM user and IAM role permissions, how to set them up, and what to request from your cyber security team if you need to do this at work.

Let's dig in, starting with terminology.

ECS, ECR, Fargate

The three AWS technologies we are going to use here are Elastic Container Service (ECS), Elastic Container Registry (ECR), and Fargate.

ECS

ECS is the core of our work. In ECS we will create a task and run that task to deploy our Docker image to a container. ECS also handles the scaling of applications that need multiple instances running. ECS Manages the deployment of our application. Learn more.

ECR

ECR is versioned storage for Docker images on AWS. ECS pulls images from ECR when deploying. Learn more.

Fargate

Fargate provisions and manages clusters of compute instances. This is amazing because:

1. You don't have to provision or manage the EC2 instances your application runs on.
2. You are only charged for the time your app is running. In the case of an application that runs a periodic task and exits this can save a lot of money.

Policies, Groups, IAM users, and IAM roles

If you are new to the AWS ecosystem and not doing this tutorial on a root account you will need to know a little about security management on AWS.

Policies

A policy is a collection of permissions for a specified services. For example you could have a policy that only allows some users to view the ECS tasks, but allows other users to run them.

Policies can be attached to Groups or directly to individual IAM users.

Groups

Groups are what they sound like: groups of users that share access policies. When you add a policy to a group, all of the members of that group acquire the permissions in the policy.

IAM user

IAM stands for Identity and Access Management but really its just an excuse to call a service that identifies a user “I am” (Clever right?). If you are not the root user you will be logging into AWS Management Console as an IAM user.

IAM Roles

Roles are a little bit more confusing. A role is a set of permissions for an AWS service. They are used when one service needs permission to access another service. The role is created for the specific type of service it will be attached to and it is attached to an instance of that service. (There are other applications for Roles but they are beyond the scope of this article.)

Setting up Permissions

As I mentioned, this is the most painful part of the process. Amazon has tried to make this easy but access management is hard.

We'll walk through setting up the appropriate policies from a root account. Then we'll translate that to what to ask for from you security team so you can get your Docker container up and running on ECS.

Create an IAM User and assign permissions

This is a good exercise to go through just to get an idea of what is going on behind the scenes. It will help you negotiate the access you need from your organization to do your job.

- Login to your AWS account as a root user. If you don't have an account you can signup for an account here.
- Search for IAM

Image by author

- From the IAM dashboard select `Users` from the left menu.
- Select `Add user` from the top of the page.

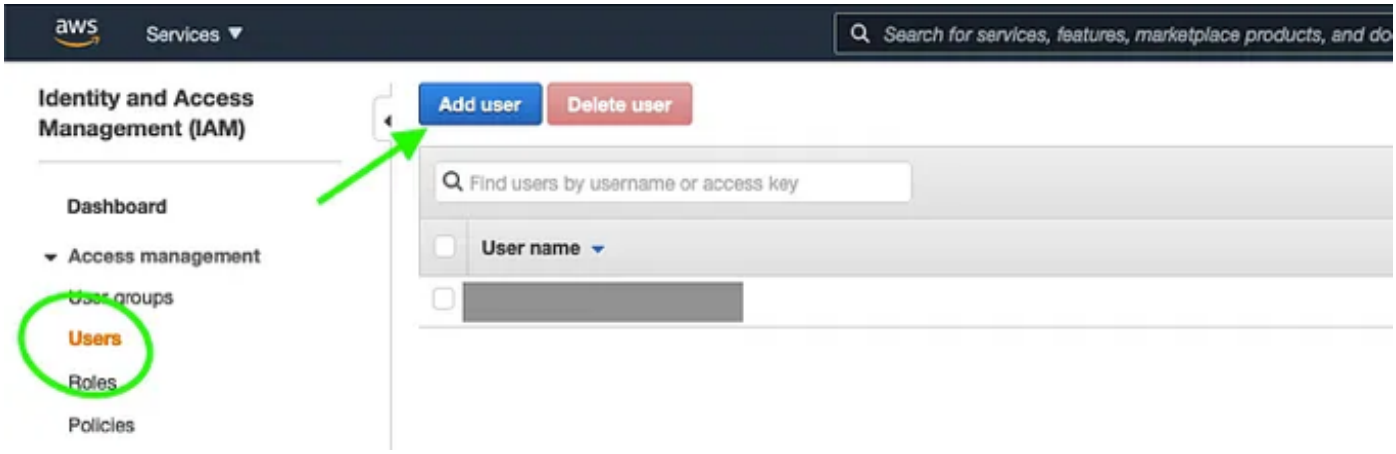


Image by author

- On the Add user screen select a username,
- Check `Programatic access` and `AWS Management Console access`. The rest we can leave they are.

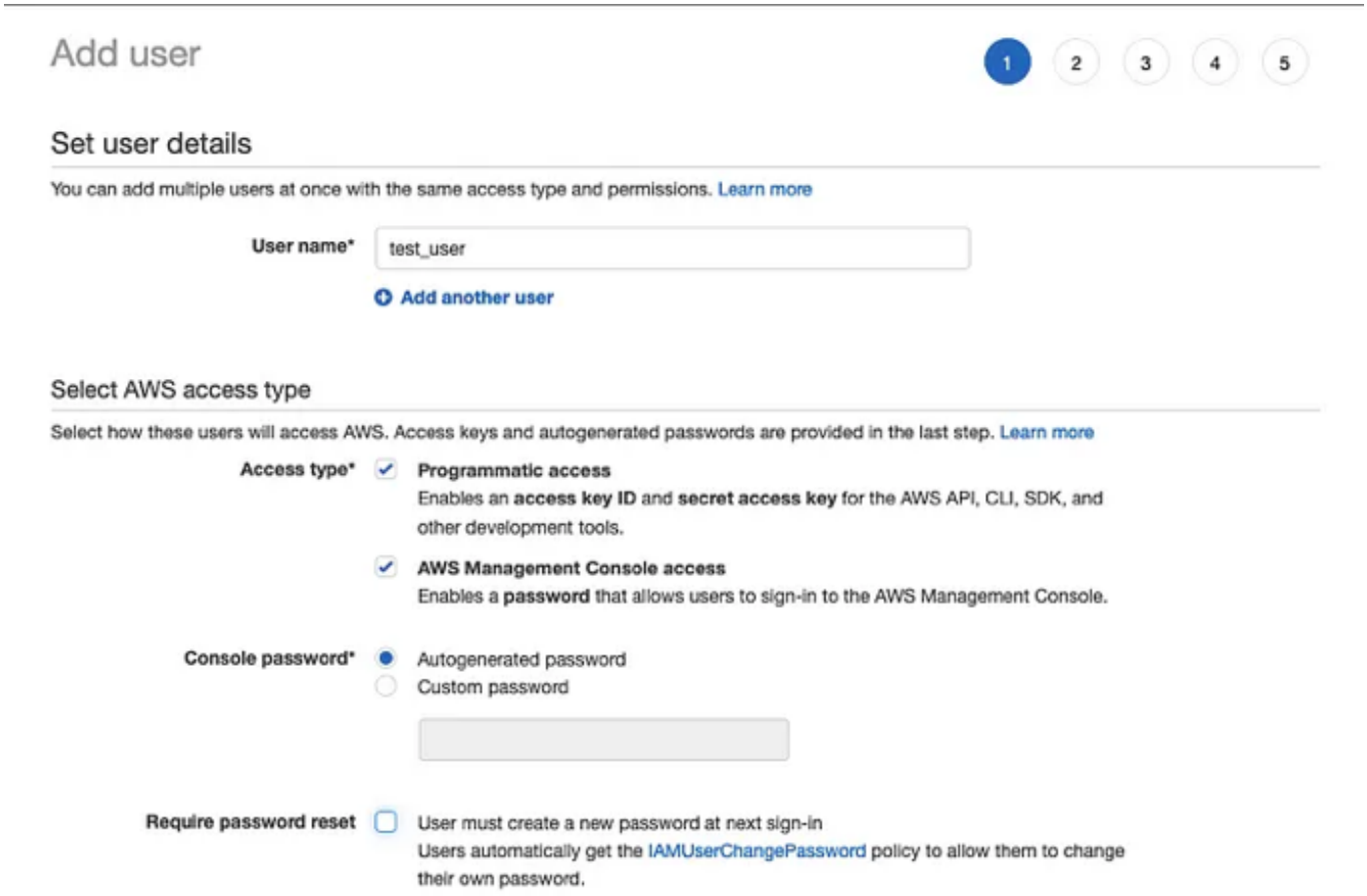


Image by author

Attaching the ECS access policy

To keep our life simple, we are going to attach the access policies directly to this new IAM user. ECS requires permissions for many services such as listing roles and creating clusters in addition to permissions that are explicitly ECS. The best way to add all of these permissions to our new IAM user is to use an Amazon managed policy to grant access to the new user.

- Select `Attach existing policies directly` directly under `Set permissions`.
- Search for `AmazonECS_FullAccess`. (the policies with the cube logo before them are Amazon managed policies).
- Select checkbox next to the policy.

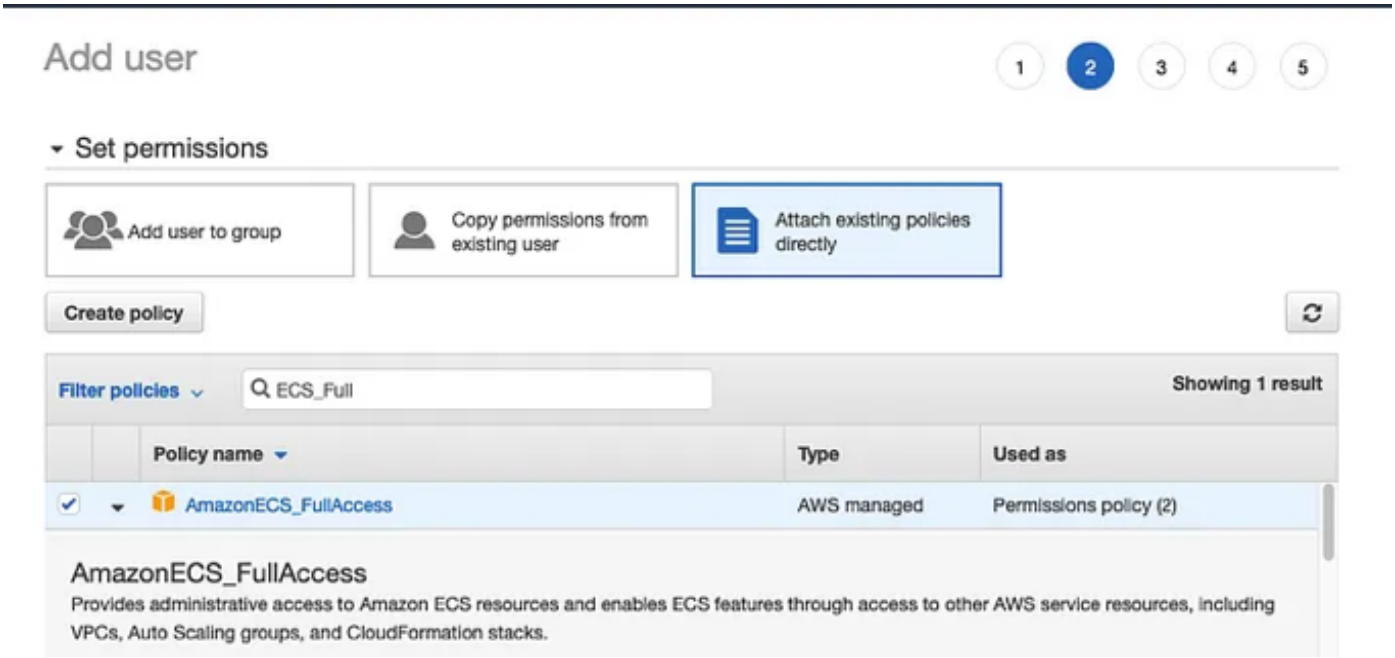


Image by author

Create an ECR policy

We will also need to have access to ECR to store our images. The process is similar except that there is no Amazon managed policy option. We must create a new policy to attach to our IAM user.

- Once again, select `Create policy`.
- Under Service select `Elastic Container Registry`.
- Under Actions select `All Elastic Container Registry actions (ecr:*)`
- Under Resources select `specific` and `Add ARN`. Here we will select the region, leave our account number and select `Any` for Repository name.

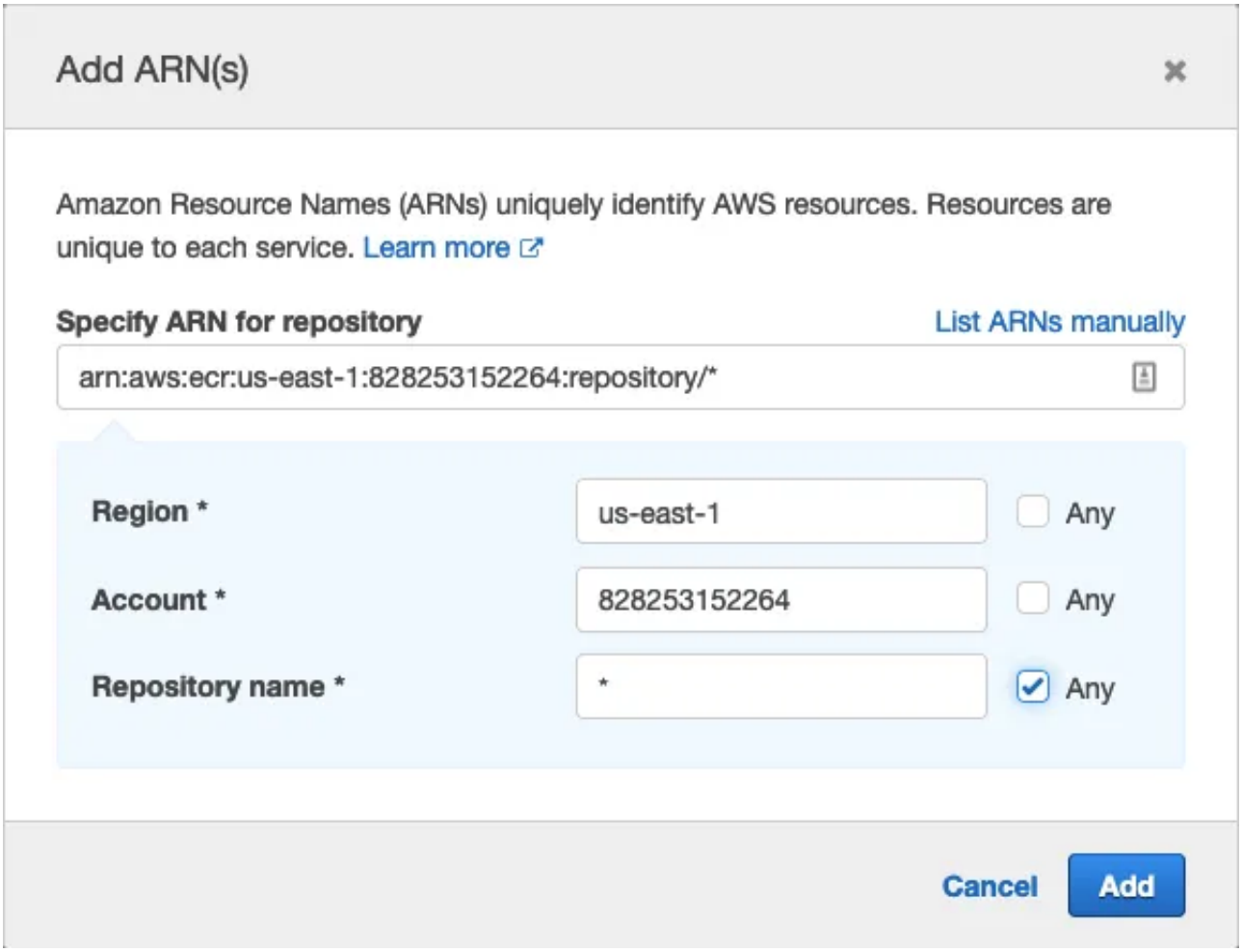


Image by author

- Click `Add`.
- Skip the tags by clicking `Next: Review`.
- Fill in an appropriate policy name. We will use `ECR_FullAccess`
- Select `Create policy`

Attach the new policies to the IAM user

- After creating the policies go back to the browser tab where we were creating the IAM user.
- Refresh the policies by clicking on the refresh symbol to the top right of the policy table.

- Search for `ECR_FullAccess ECS_FullAccess` and select the radio button to the left of each policy we created to attach it to our IAM user.

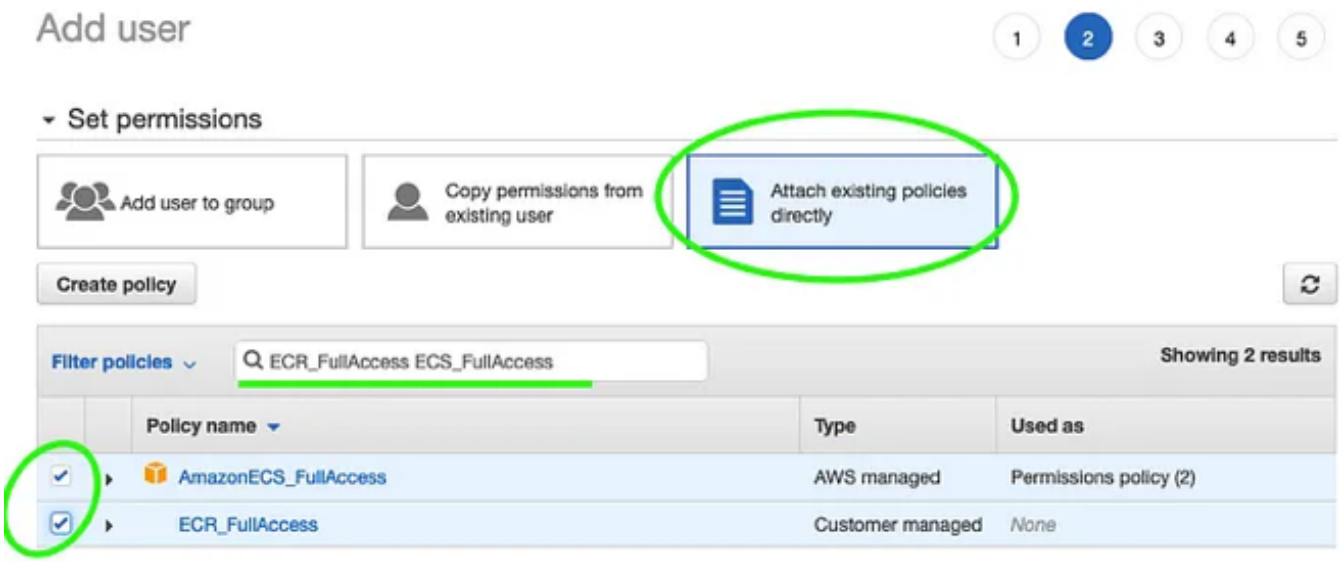


Image by author

- Select `Next:Tags`.
- Leave tags blank.
- Select `Next:Review`
- Finally, review our work and create the user.

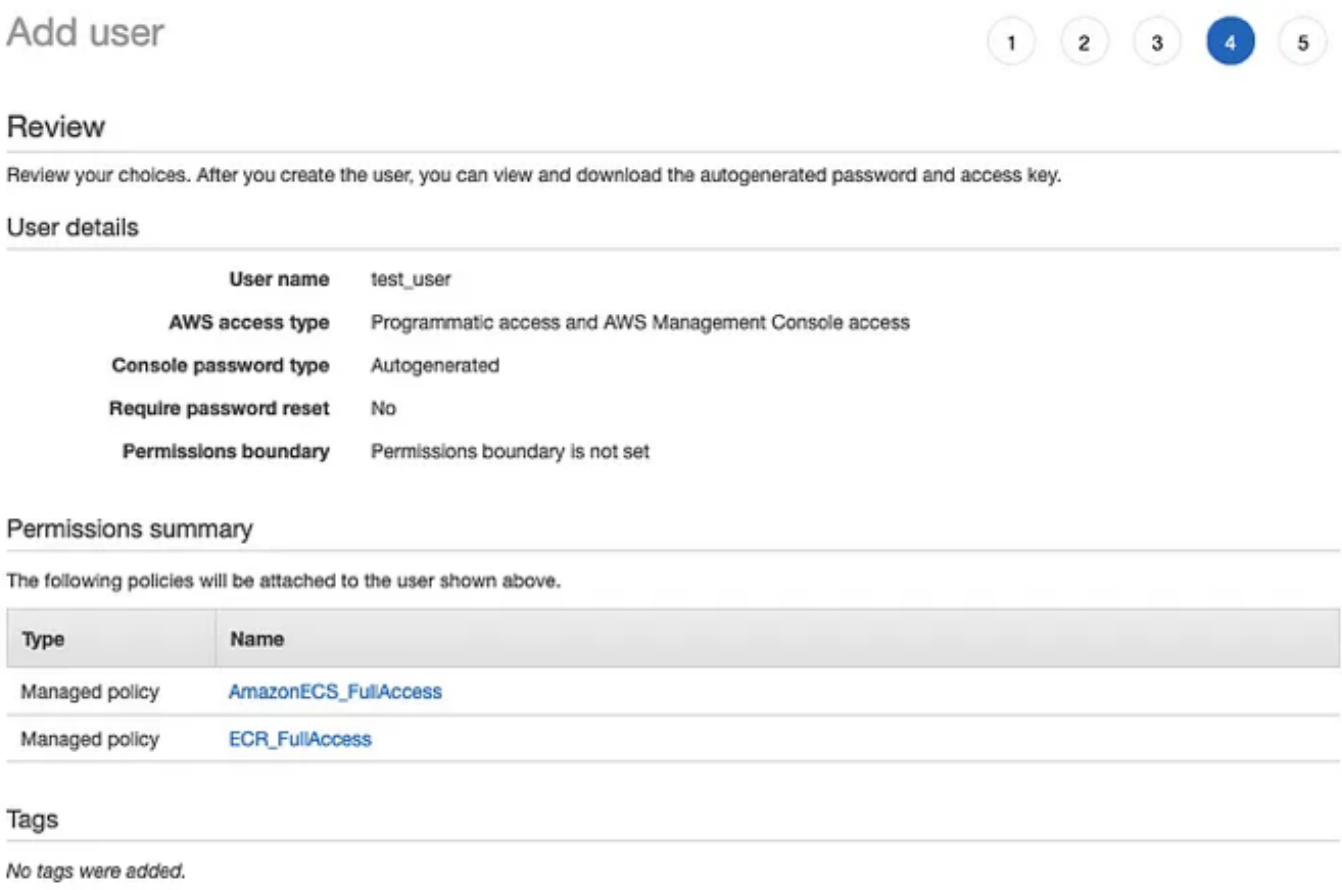


Image by author

When you submit this page you will get a confirmation screen. **Save** all of the information there in safe place **we will need all of it when we deploy our container.**

In the real world it is unlikely that you would need to create these permissions for yourself. It's much more likely that you will need to request them from someone, perhaps a security team, at your organization. Now that you know a little about what is involved you are better prepared to make that request.

Your request should contain

- a very brief explanation of what you need to accomplish.

- a requested list of permissions.

The second is arguably unnecessary, but it will save everyone the time and pain of many back and forth emails as they try to work out exactly which permissions you need.

They may grant the permissions you request, or they may grant you a subset of them. They are the cyber security experts so if you get less than you ask for proceed in good faith. If you hit a wall, send them the error so they can grant the necessary permissions for you to move forward.

Your request could look something like this:

```
Hi Joe,

I need to deploy a Docker container on ECS.  I will also need
access to ECR for this.

Please add the following to my IAM user privileges:
- AmazonECS_FullAcces managed policy
- The following policy for ECR access:
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "ecr:*",
            "Resource": "*"
        }
    ]
}

Thanks! You are the best!
```

Ok on to the main event

Deploying a Docker Container to ECS

The steps here are:

1. Create the Docker image
2. Create an ECR registry
3. Tag the image
4. Give the Docker CLI permission to access your Amazon account
5. Upload your docker image to ECR
6. Create a Fargate Cluster for ECS to use for the deployment of your container.
7. Create an ECS Task.
8. Run the ECS Task!

Create the Docker image

For the purpose of this demo I am going to use an a simple flask app that shows gifs of cats from this GitHub repository. The app is part of docker-curriculum.com which is a great Docker primer if you are just getting started.

Lets begin

- Clone the source files form GitHub and cd into the `flask-app` directory.

```
$ git clone https://github.com/prakhar1989/docker-curriculum.git
$ cd docker-curriculum/flask-app
```

- Create the Docker image:

```
docker build -t myapp .
```

Test the app to make sure everything is working. The flask app we downloaded listens on port 5000 so we will use the same port to test.

```
docker run --publish 5000:5000 myapp
```

Now you should be able to go to `localhost:5000` and see a random cat gif

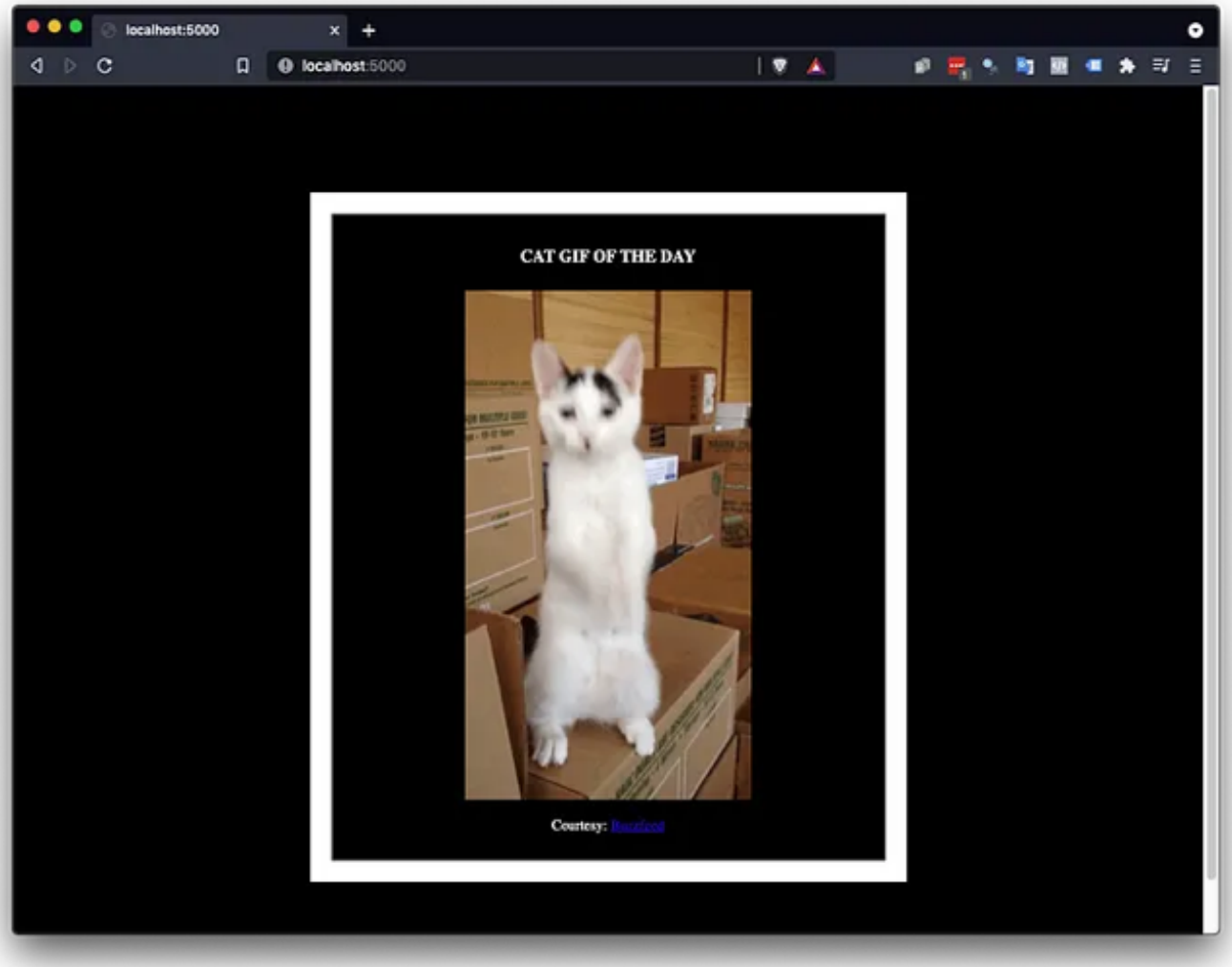


Image by author

Yay!

Create an ECR registry.

In this step we are going to create the repository in ECR to store our image. We will need the ARN (Amazon Resource Name — a unique identifier for all AWS resources) of this repository to properly tag and upload our image.

First login to the AWS console with the test_user credentials we created earlier. Amazon will ask for your account id, username, and password.

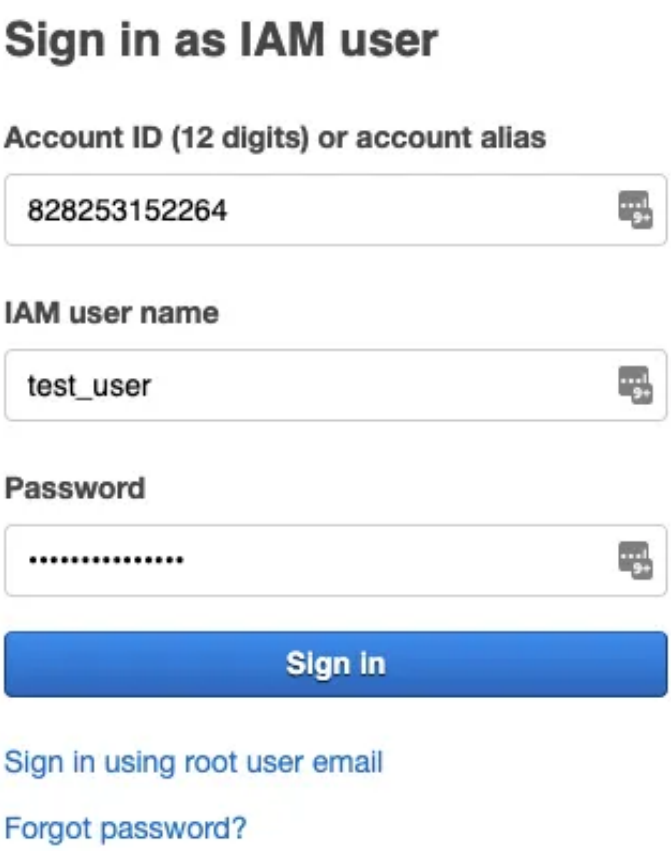


Image by author

- Once you are in, search for `Elastic Container Registry` and select it.

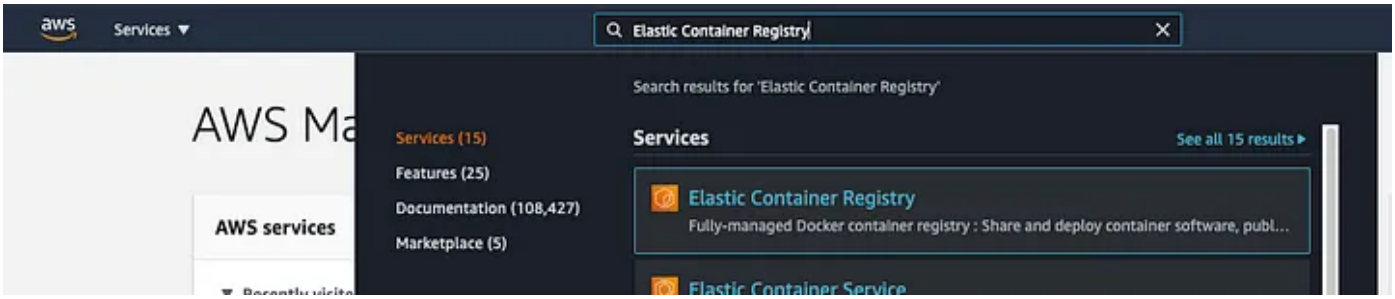


Image by author

- From there fill in the name of the repository as `myapp` and leave everything else default.

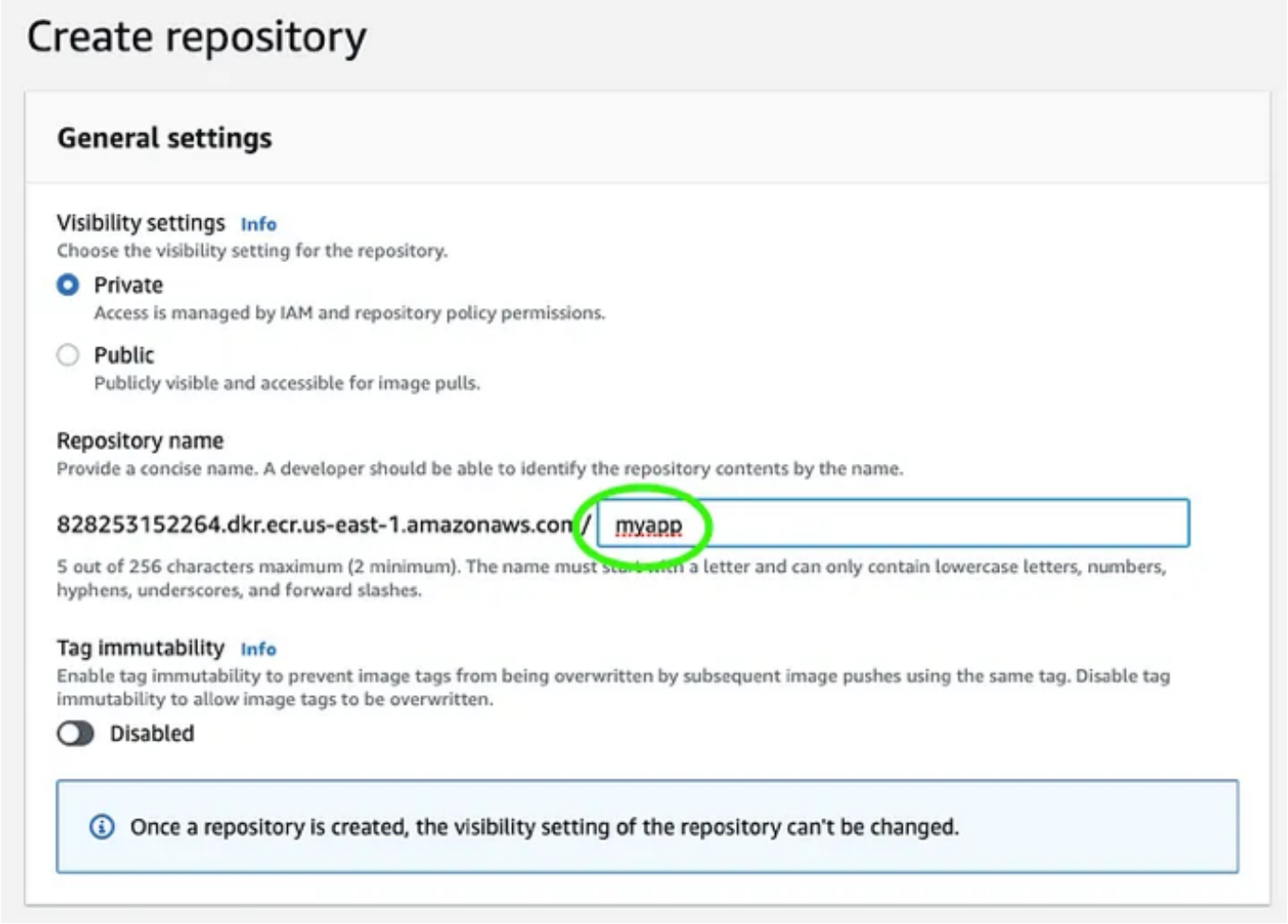


Image by author

- Select `Create Repository` in the lower left of the page and your repository is created. You will see your repository in the repository list, and most importantly the ARN(here called a URI) which we will need to push up our image. Copy the URI for the next step.

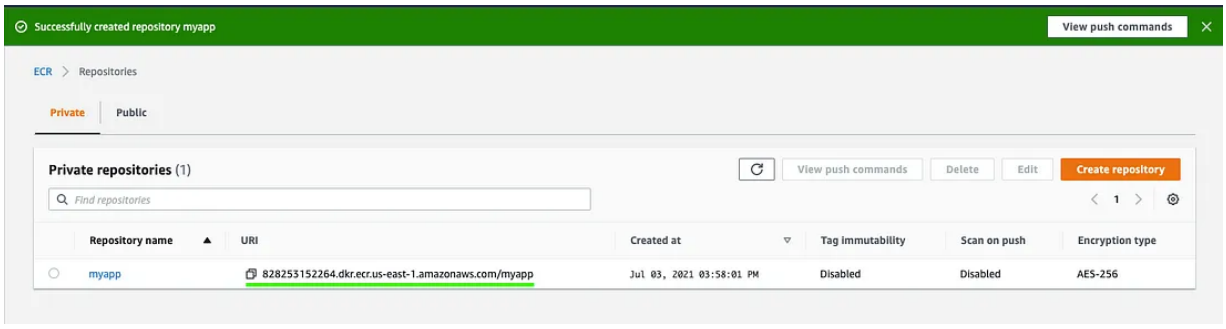


Image by author

If you prefer you can also do the above step from the command line like so:

```
$ aws ecr create-repository \
    --repository-name myapp \
    --region us-east-1
```

Tag the image

In order for ECR to know which repository we are pushing our image to we must tag the image with that URI.

```
$ docker tag myapp [use your uri here]
```

The full command for my ECR registry looks like this:

```
docker tag myapp 828253152264.dkr.ecr.us-east-
1.amazonaws.com/myapp
```

Give the Docker CLI permission to access your Amazon account

I'll admit this step is a little convoluted. We need to login to aws to get a key, that we pass to docker so it can upload our image to ECR. You will need the aws cli for the rest of our work.

- First we'll login to our aws account.

```
# aws configure
```

AWS will ask us for our credentials which you saved from way back when we created the AIM user (right?). Use those credentials to authenticate.

- Next, we need to generate a ECR login token for docker. This step is best combined with the following step but its good to take a deeper look to see what is going on. When you run the followign command it spits out an ugly token. Docker needs that token to push to your repository.

```
# aws ecr get-login-password --region us-east-1
```

- We can pipe that token straight into Docker like this. Make sure to replace `[your account number]` with your account number. The ARN at the end is the same as the one we used earlier without the name of the repository at the end.

```
# aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin [your account number].dkr.ecr.us-
east-1.amazonaws.com
```

If all goes well the response will be `Login Succeeded`.

Upload your docker image to ECR

We've done the hard part now. It should be smooth sailing from here.

- Use docker to push the image to the ECR repository.

```
docker push 828253152264.dkr.ecr.us-east-1.amazonaws.com/myapp
```

Create a Fargate Cluster.

Let's return to the AWS management console for this step.

- Search for `Elastic Container Service` and select `Elastic Container Service`.
- From the left menu select `Clusters`
- Select `Create cluster`

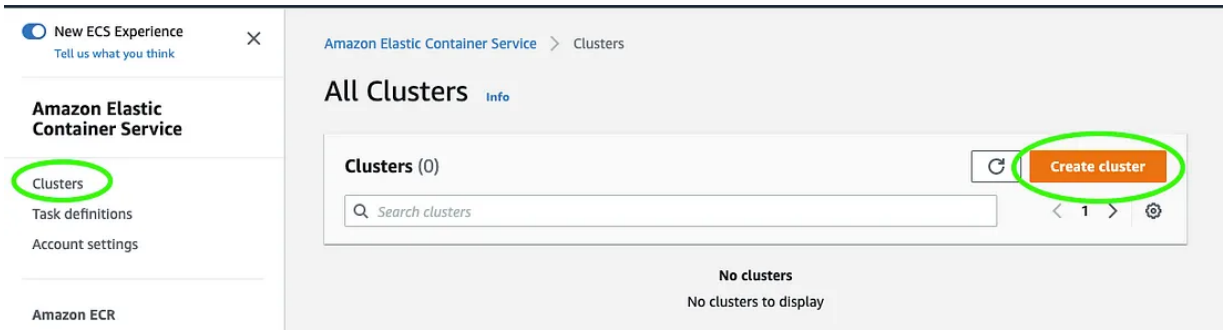


Image by author

- Under `Select cluster template` we are going to select networking only. We don't need ec2 instances in our cluster because Fargate will take care of spinning up compute resources when we start our task and spinning them down when we stop our task.

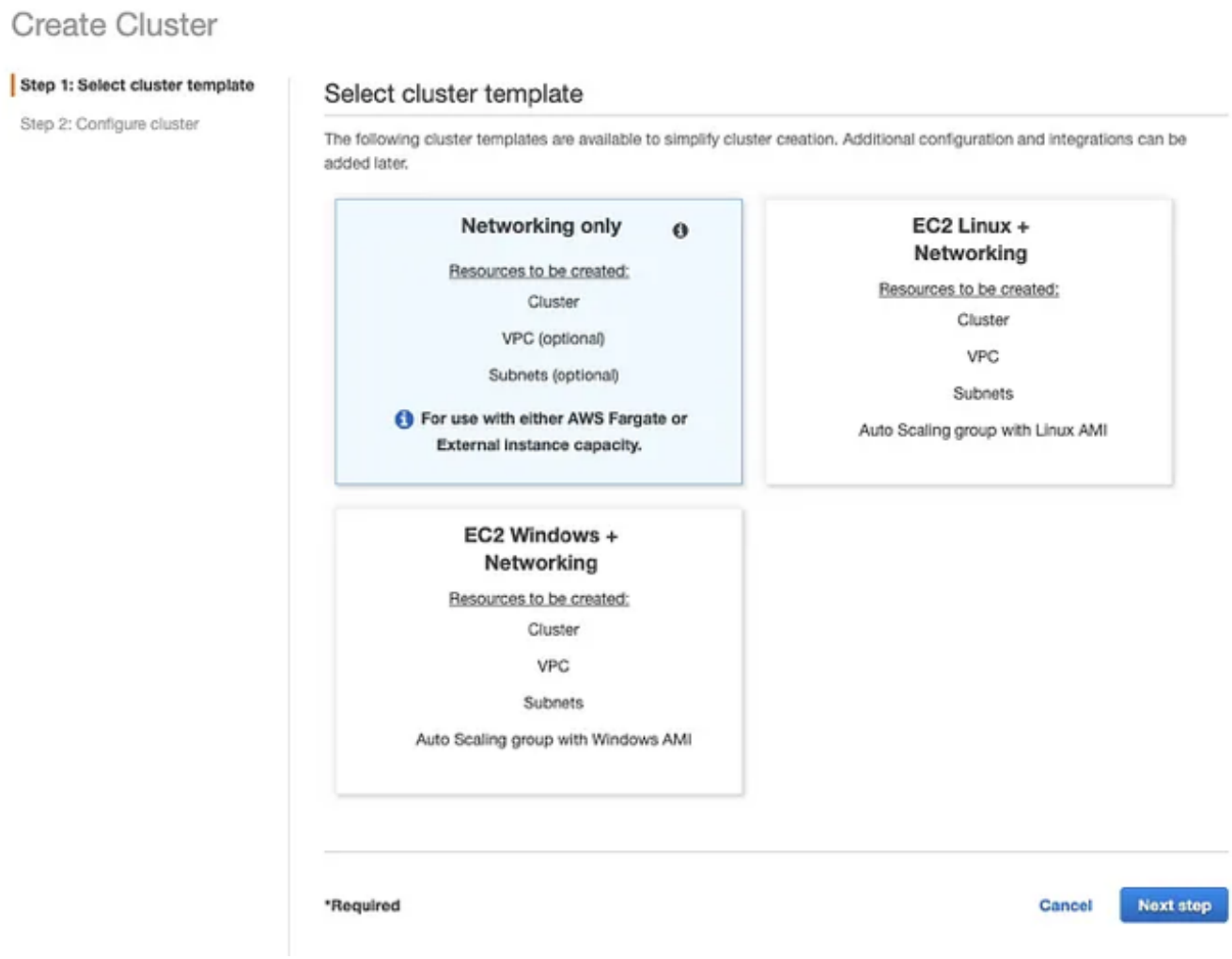
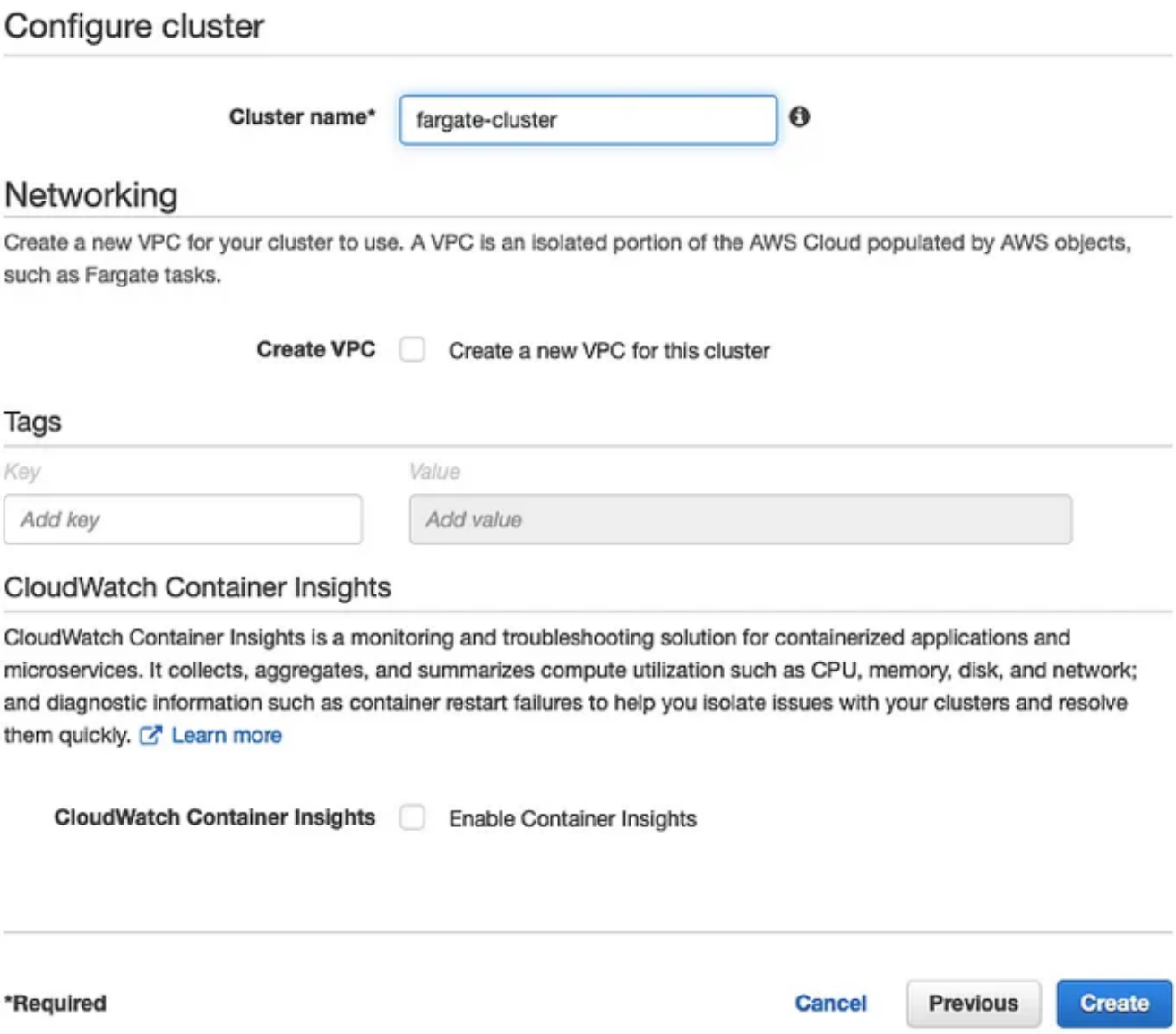


Image by author

- I'll name the cluster `fargate-cluster`, and the rest we can leave as is.



- Select `Create`

Create an ECS Task

The ECS Task is the action that takes our image and deploys it to a container. To create an ECS Task lets go back to the ECS page and do the following:

- Select `Task Definitions` from the left menu. Then select `Create new Task Definition`

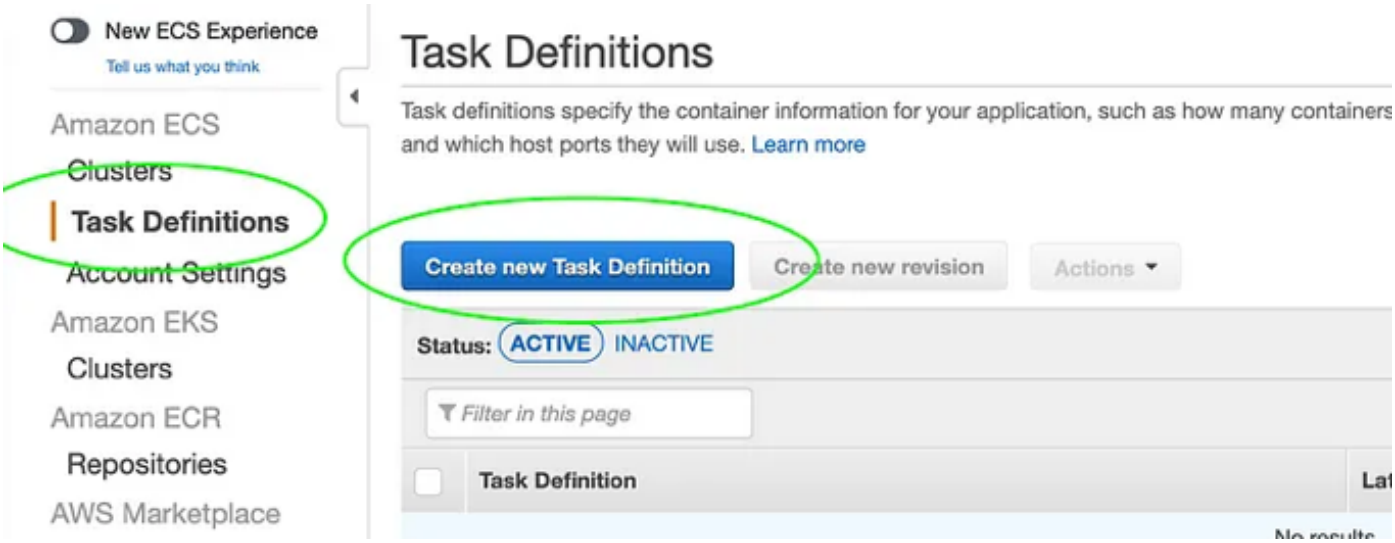


Image by author

- Select `Fargate`
- Select `Next Step`

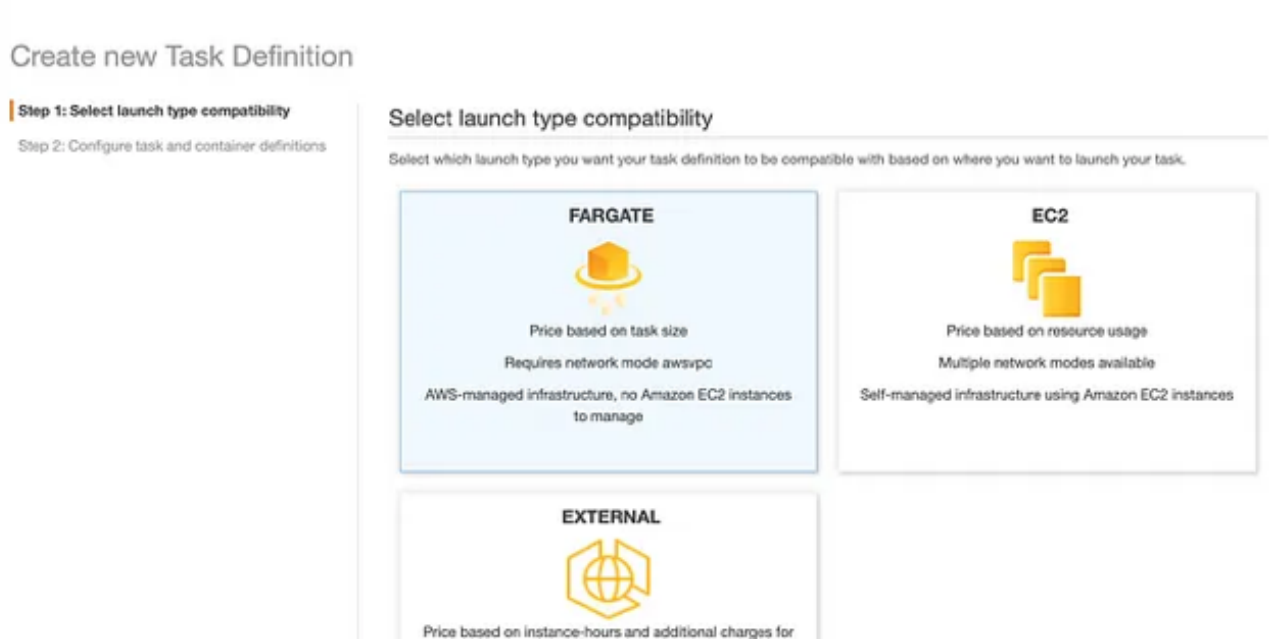


Image by author

- Enter a name for the task. I am going to use `myapp`.
- Leave `Task Role` and `Network Mode` set to their default values.

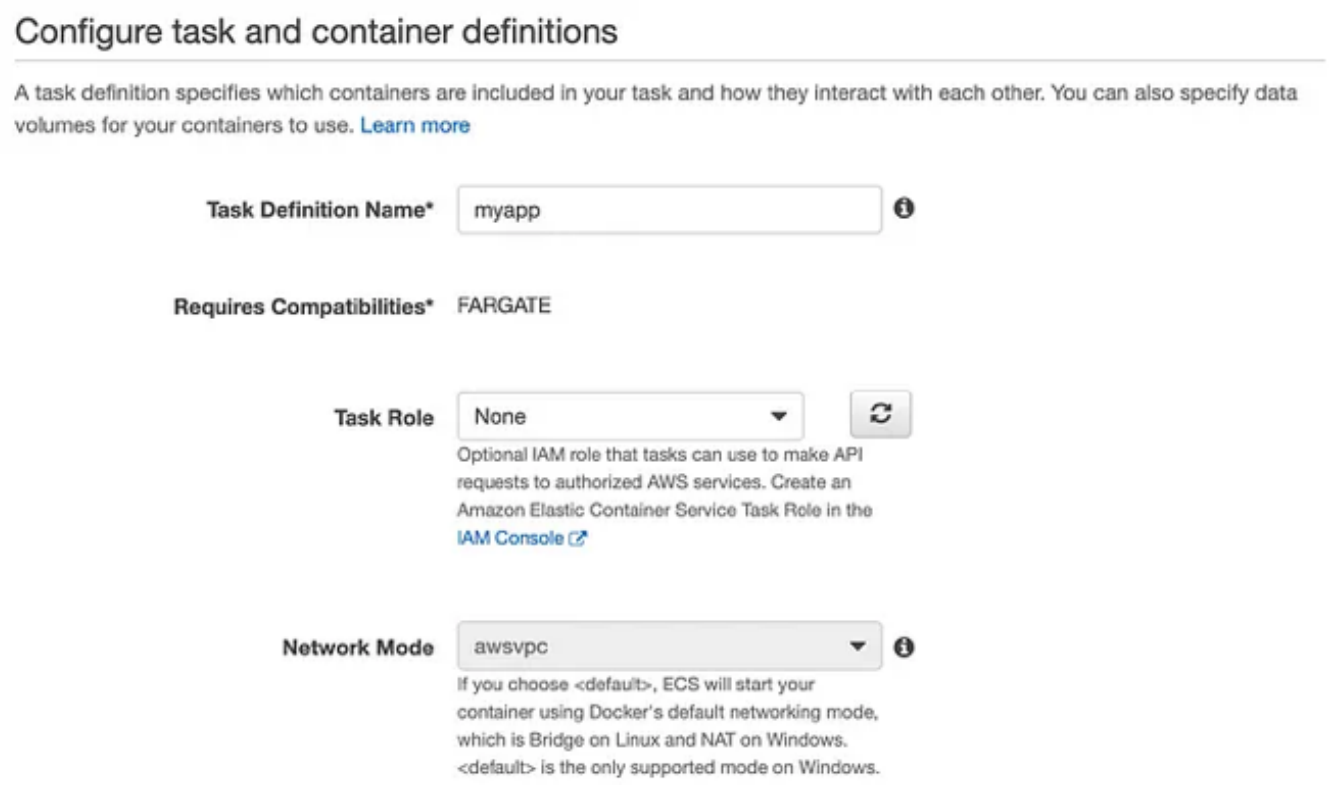


Image by author

- Leave `Task Execution Role` set to its default.
- For Task memory and Task CPU select the minimum values. We only need minimal resources for this test.

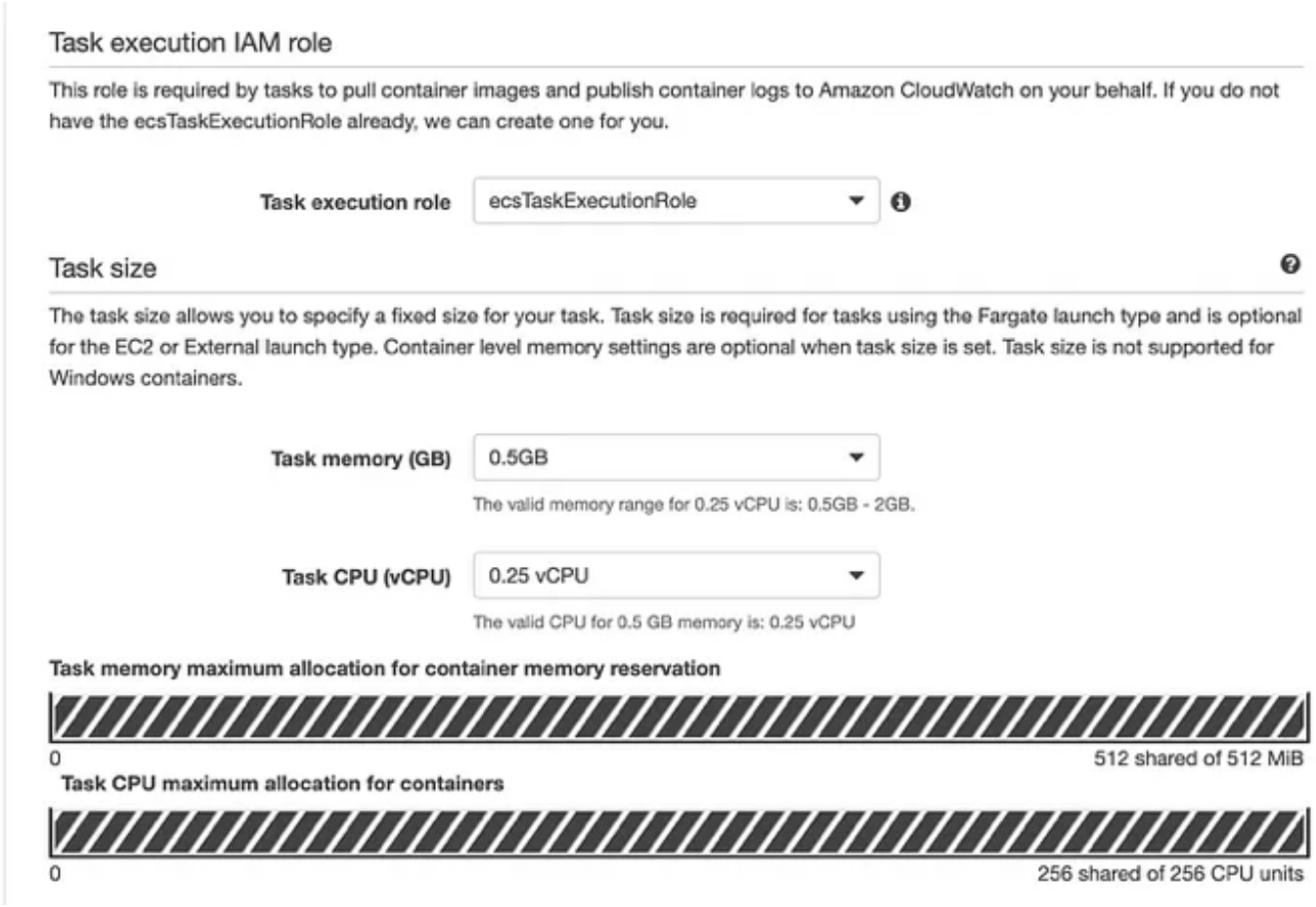


Image by author

- Under Container definition select `Add Container`.
- Enter a Container name. I will use `myapp` again.
- In the Image box enter the ARN of our image. You will want to copy and paste this from the ECR dashboard if you haven't already.
- We can keep the Memory Limit to 128Mb
- In port mappings you will notice that we can't actually map anything. Whatever port we enter here will be opened on the instance and will map to the same port on container. We will use 5000 because that is where our flask app listens.

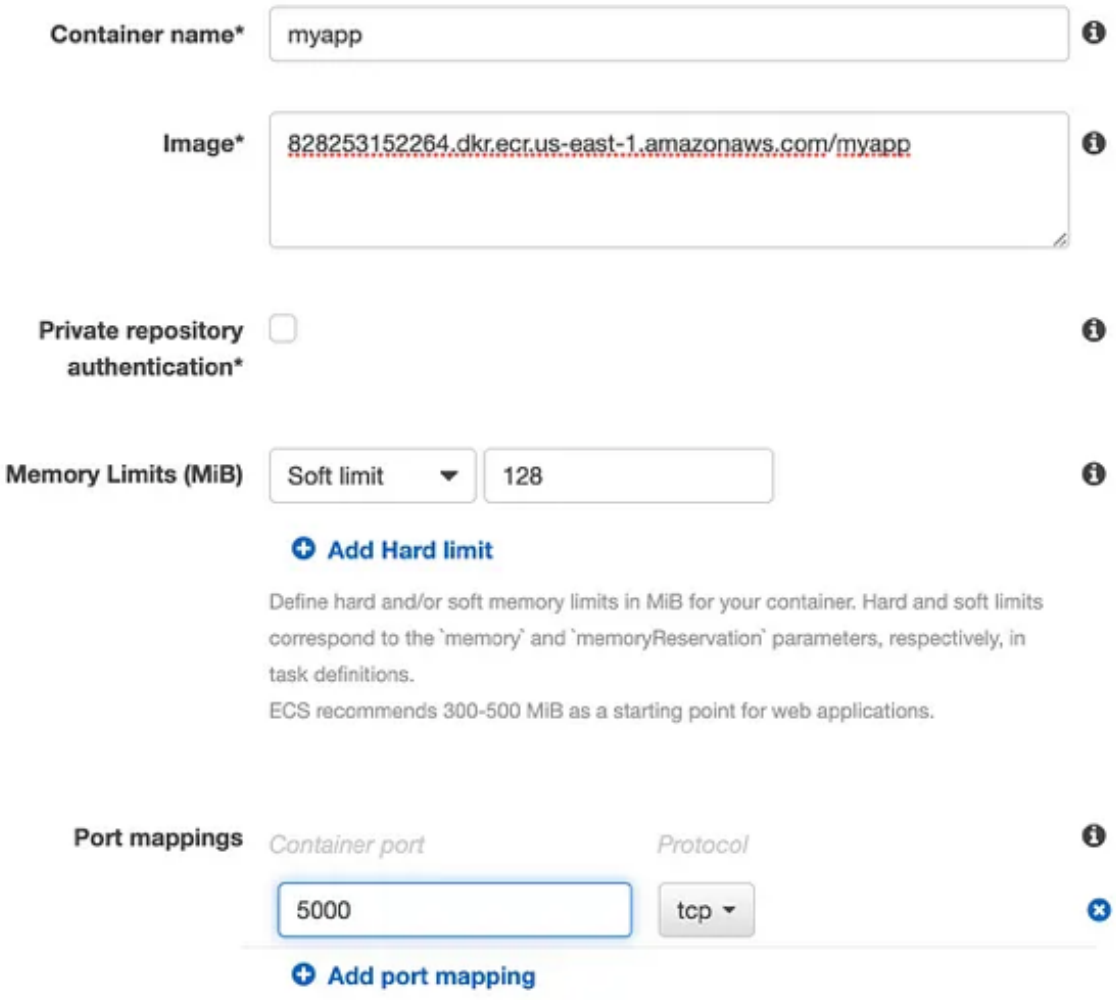


Image by author

- Leave everything else set to its default value and click `Add` in the lower left corner of the dialog.
- Leave everything else in the Configure task and container definitions page as is and select `Create` in the lower left corner of the page.
- Go back to the ECS page, select `Task Definitions` and we should see our new task with a status of ACTIVE.

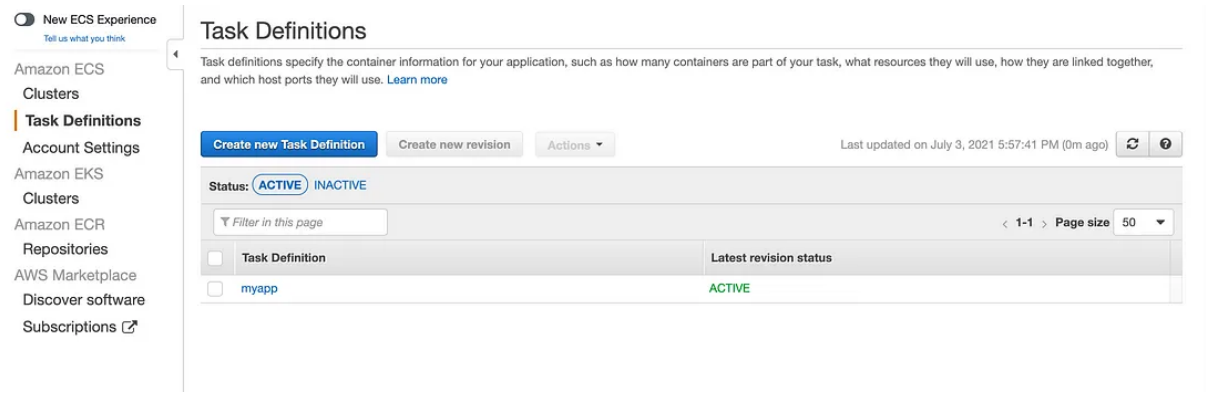


Image by author

Run the ECS Task!

This is the moment we have all been waiting for.

- Select the task in the Task definition list
- Click `Actions` and select `Run Task`

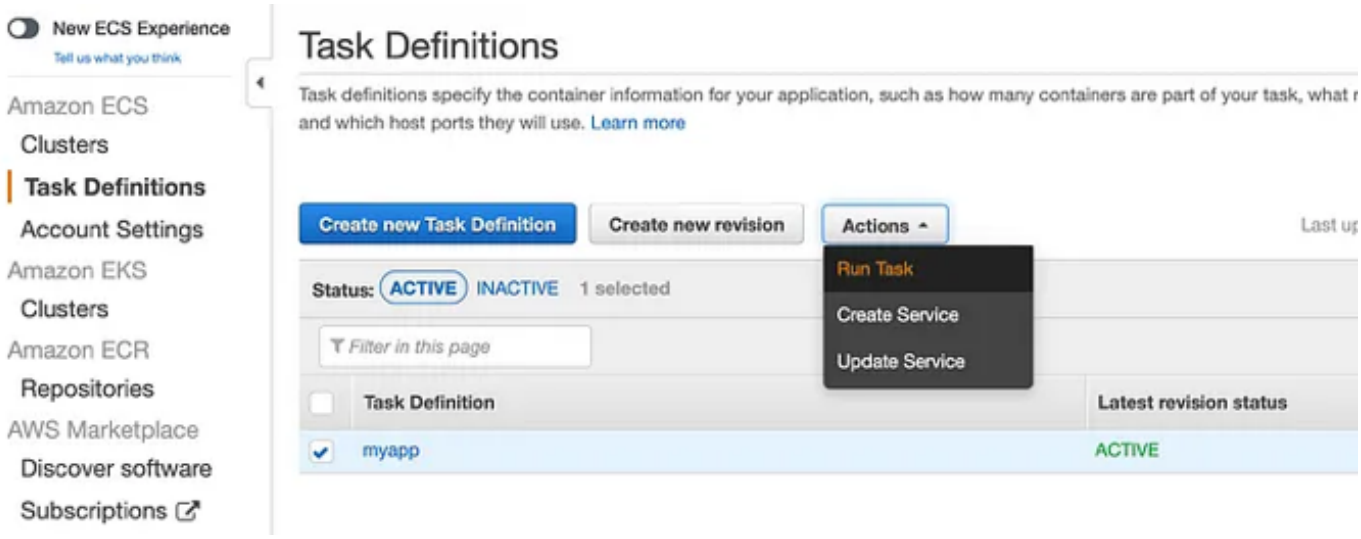


Image by author

- For `Launch type`: select `Fargate`

- Make sure `Cluseter` : is set to the `fargate-cluster` we created earlier.

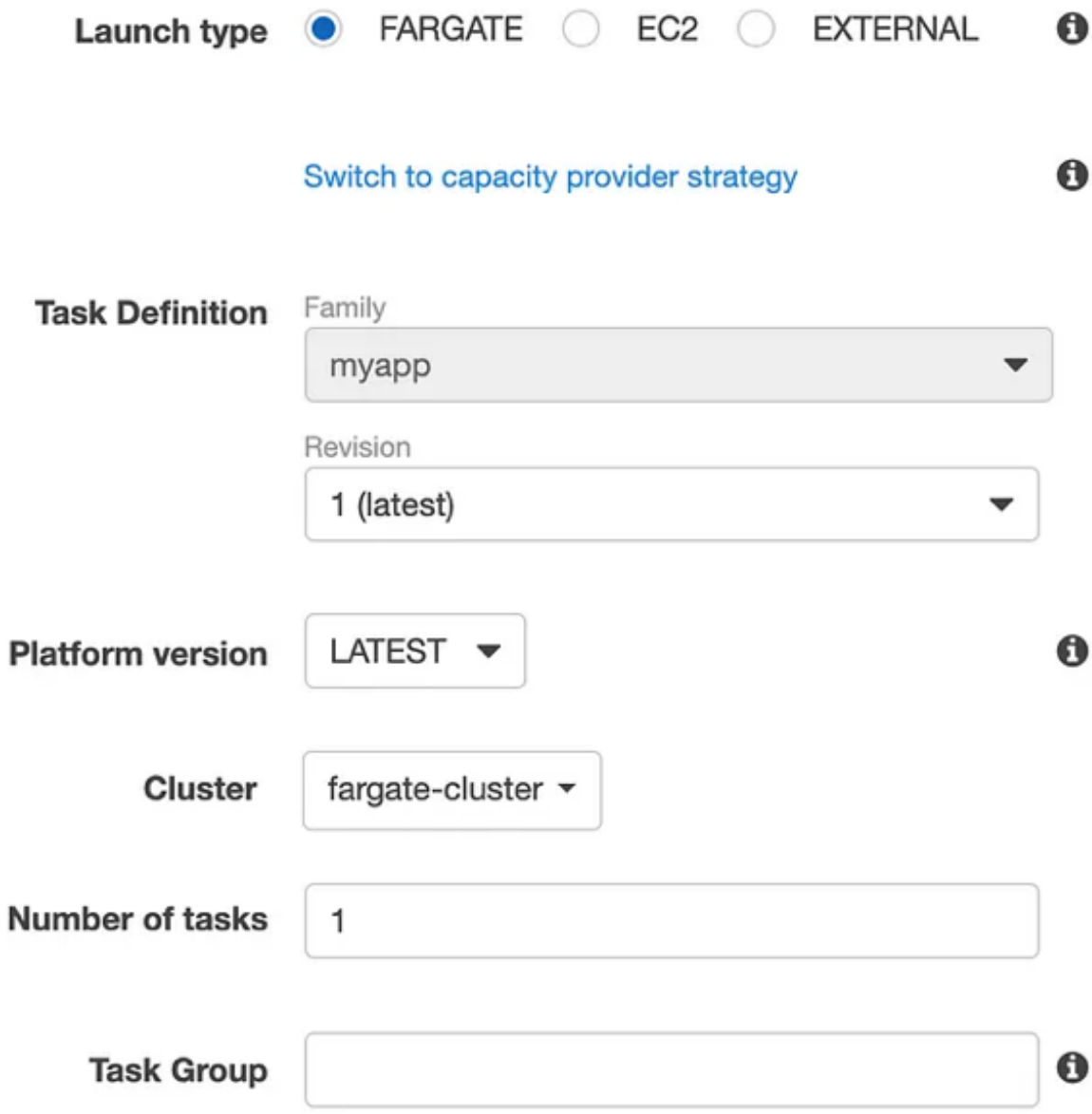


Image by author

- Cluster VPC select a vpc from the list. If you are building a custom app this should be the vpc assigned to any other AWS services you will need to access from your instance. For our app, any will do.

- Add at least one subnet.
- `Auto-assign public IP` should be set to `ENBABLED`

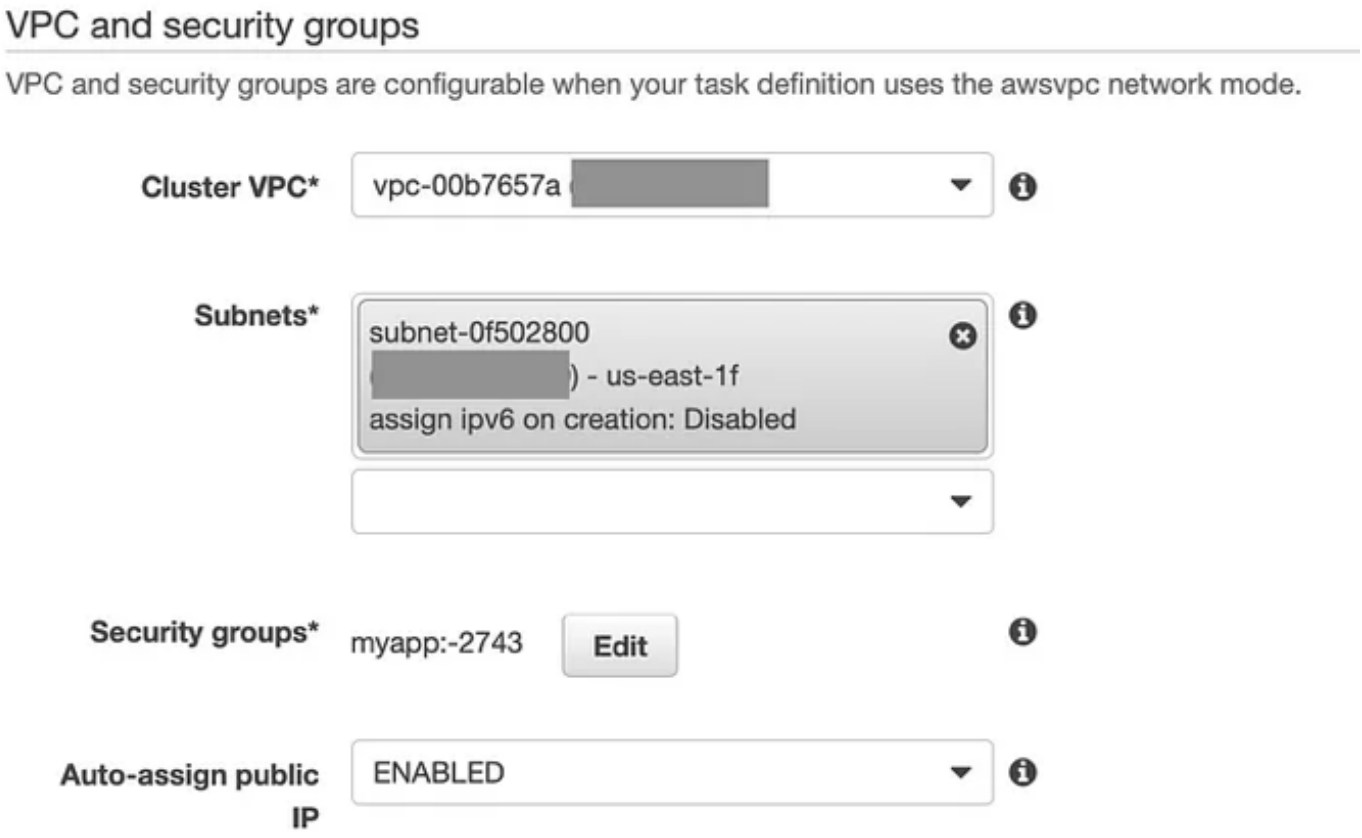


Image by author

- Edit the security group.
  Because our app listens on port 5000, and we opened port 5000 on our container, we also need to open port 5000 in the security group. By default the security group created by Run Task only allows incoming connections on port 80. Click on `Edit` next to the security group name and add a Custom TCP rule that opens port 5000.

Image by author

And finally, run the task by clicking `Run Task` in the lower left corner of the page.

Check to see if our app is running

After you run the Task, you will be forwarded to the `fargate-cluster` page. When the Last Status for your cluster changes to `RUNNING`, your app is up and running. You may have to refresh the table a couple of times before the status is `RUNNING`. This can take a few minutes.

Image by author

- Click on the link in the Task column.
- Find the Public IP address in the Network section of the Task page.

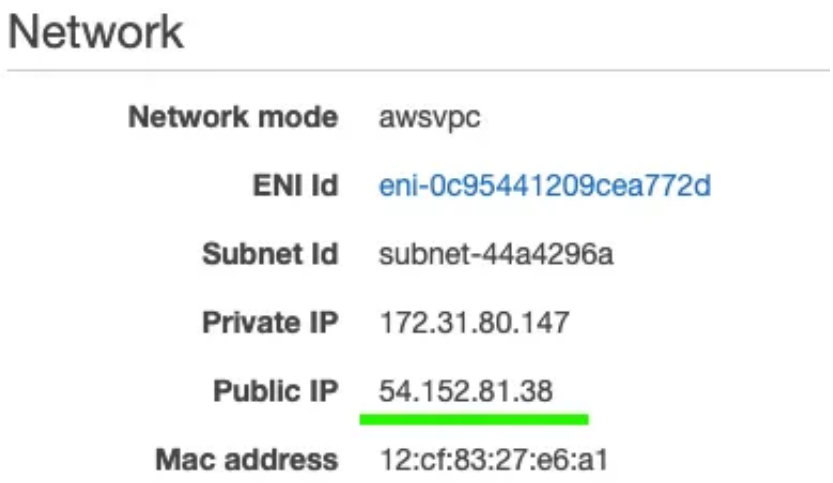


Image by author

- Enter the public IP address **followed by :5000** in your browser to see your app in action.

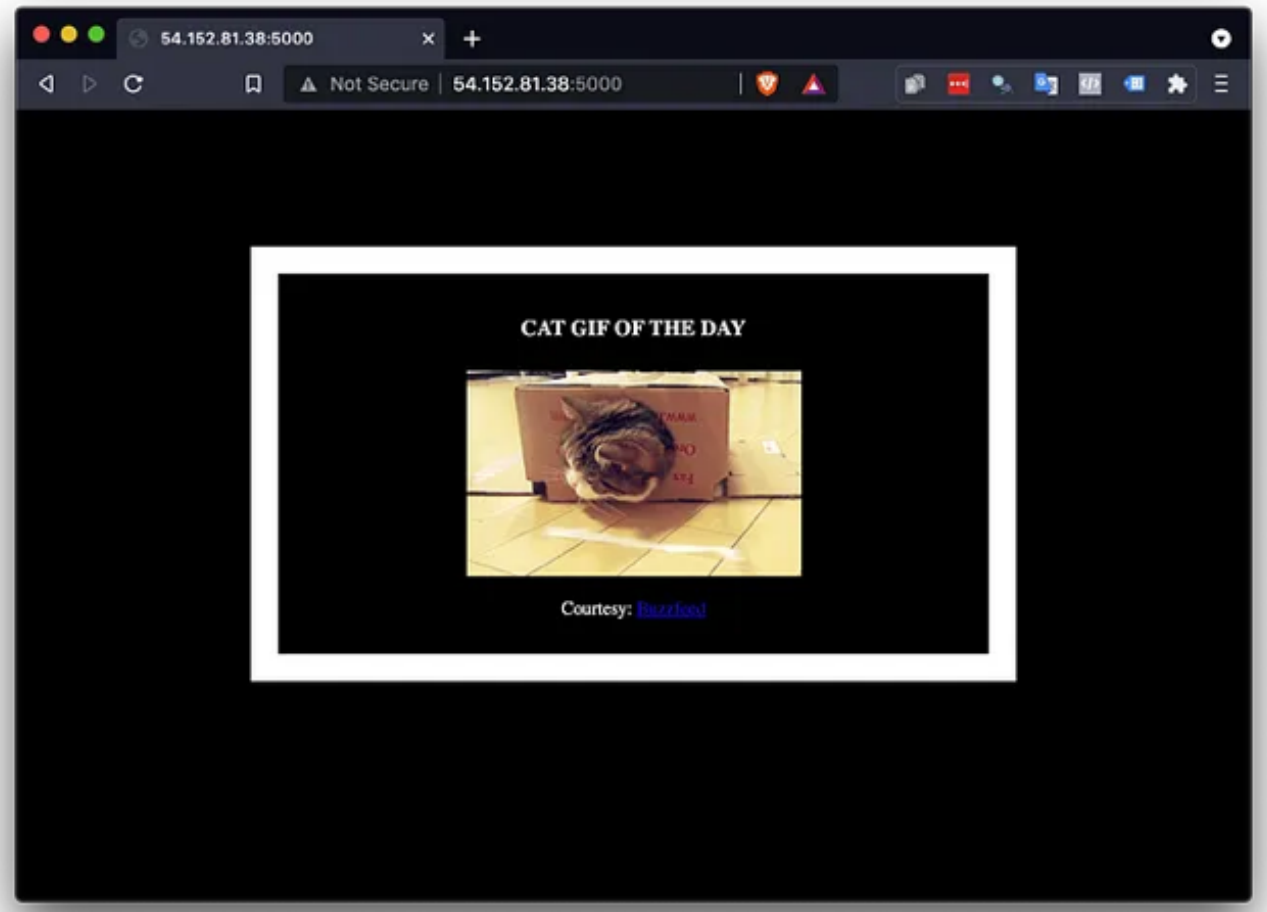


Image by author

Shut down the app

When you are done looking at cat gifs, you'll want to shut down your app to avoid charges.

- From the ECS page select Clusters from the left menu, and select the `fargate-cluster` from the list of clusters.

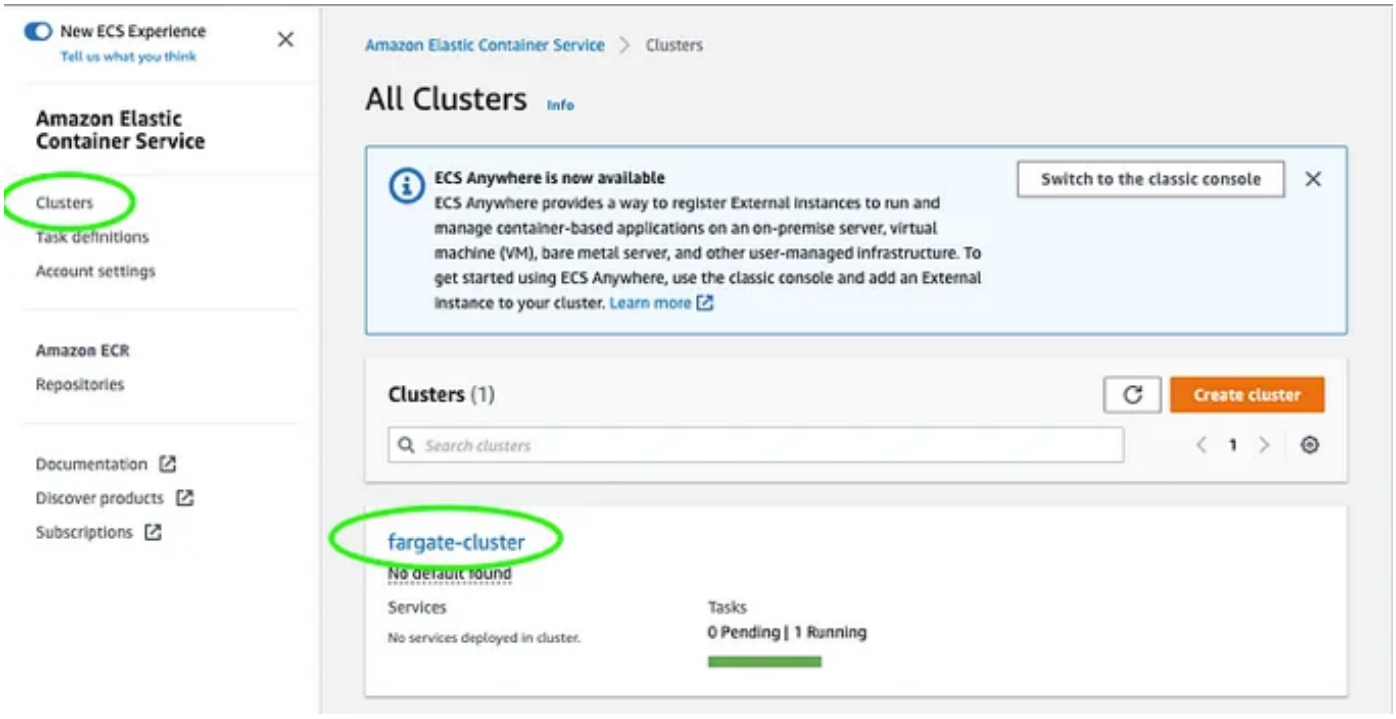


Image by author

- From the table at the bottom of the page select tasks.
- Check the box next to the running task
- Select stop from the dropdown menu at the top of the table

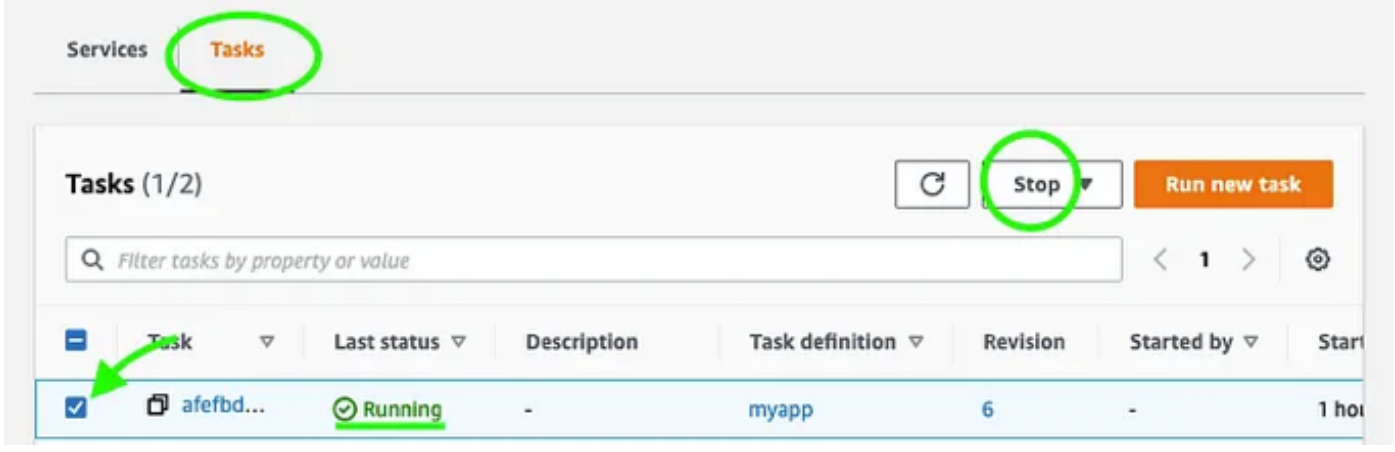


Image by author

Conclusion

Now that you know how to deploy a Docker image to ECS the world is your oyster. You can deploy a scraping app that runs until it completes then shuts down so you are only billed for the time it runs. You can scale a web service. You can spread cat gifs around the internet with multiple cat gif servers. It's all up to you.

Resources

- If your permissions do not allow your Task to create an ECS task execution IAM role you can create one with these directions.

Amazon ECS task execution IAM role

The task execution role grants the Amazon ECS container and Fargate agents permission to make AWS API calls on your...

docs.aws.amazon.com

- docker-curriculum.com is a good place to start if you are new to Docker.

A Docker Tutorial for Beginners

Learn to build and deploy your distributed applications easily to the cloud with Docker Written and developed by...

docker-curriculum.com

- The Amazon tutorial for deploying a Docker image to ECS.

Docker basics for Amazon ECS

Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications that...

docs.aws.amazon.com

Now go do good.

Python Docker AWS Containers Fargate

Written by Ben Bogart

Follow

245 Followers · Writer for Towards Data Science

Bandoneonista and Data Scientist at Komaza. linkedin.com/in/benbogart/

More from Ben Bogart and Towards Data Science