Resources:

- Codebase: https://github.com/dswh/dsml_feb_flask_app
- Lecture: https://www.scaler.com/meetings/i/ml-ops-deploying-ml-applications-on-aws-sagemaker/archive

Table of Content

Introduction

You might have heard about the term "**Cloud**" and "**Cloud Computing**". So, what is it?

- Cloud services are infrastructure, platforms, or software that are hosted by third-party providers and made available to users through the internet.
- There are several cloud service providers like;
    1. AWS
    2. GCP
    3. Azure
    4. DigitalOcean
    5. Oracle Cloud
- These providers set up huge set of data-centers across different regions of the world; all of them having hundreds of thousands of storage and computing machines, which they rent to the users for development.
- Every tech company building products or offering some sort of service needs give access to people across the world.
- For this, companies deploy their products on these cloud platforms.

Local Machine and Cloud Machine

Generally speaking, what things do a developer needs on his local machines?

1. Computing device - Processor
2. Storage - RAM and HDD/SSD
3. Networking Capabilities
4. Security
5. Logs and Monitoring - Tracking files and changes being made.

Now, when the scale of a particular tech product goes up, developers would need to upgrade these specifications to a great extend.

- For instance, a single instance of intel/mac/amd processor would not suffice running Facebook in India.

This is where Cloud Service Providers comes into scenario. They offer high computing powered CPUs/GPUs, more storage, etc.

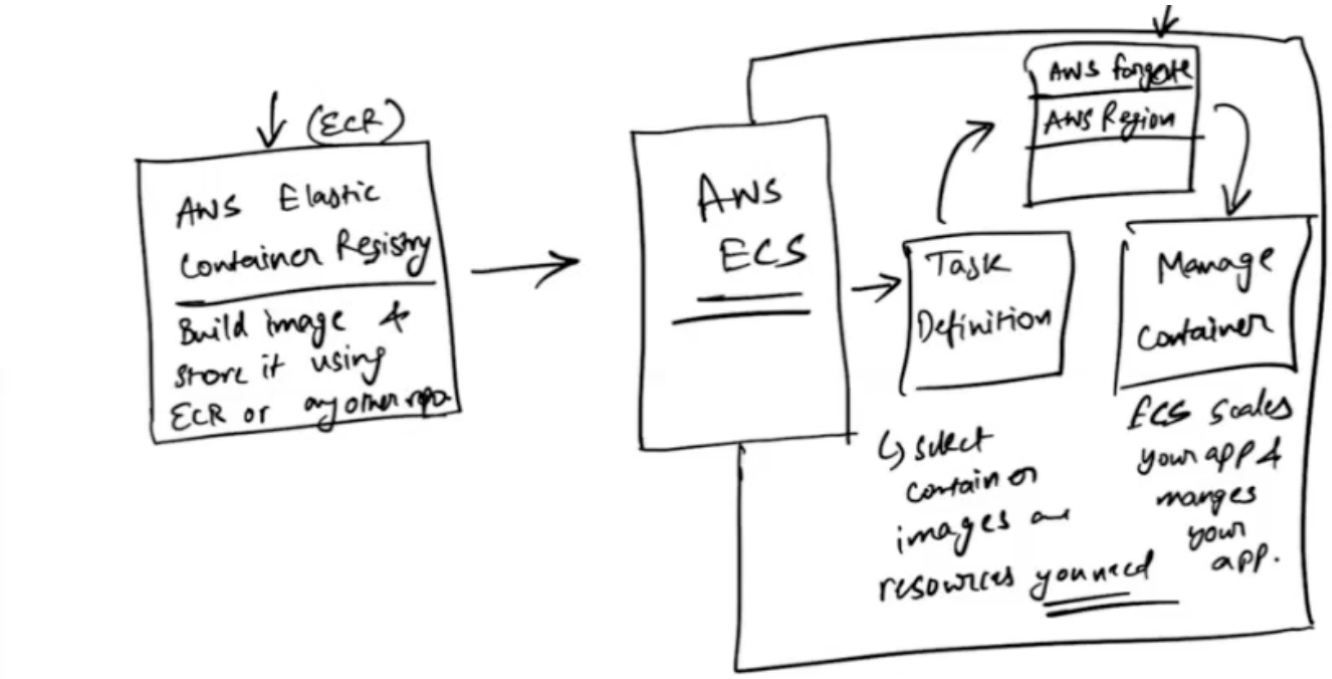
- More computing power/more storage means higher rent.

Problem Statement

Let's say you have a Flask/Streamlit application that is dockerised, the task is to deploy that docker image on AWS.

- In the last lecture, we learnt that any app can be deployed on a docker, regardless of OS, and that docker image can be used to run that application.
- We would use that docker image, and will deploy to a servie that AWS offers; known as ECS which stands for **Elastic Container Service.**

ECS is a cloud computing service offered by AWS that manages containters and let developers run applications without worrying about environment to run the code.



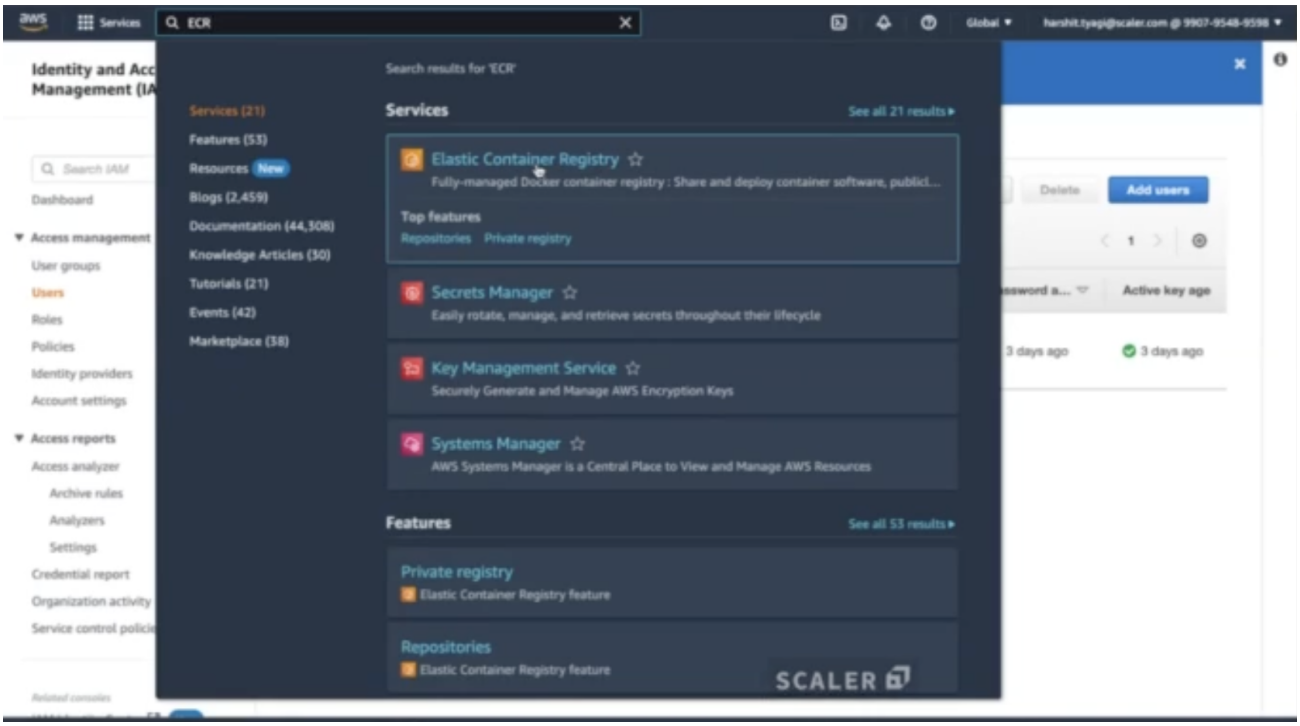
- Following are the steps that are followed when working with ECS:
    1. **Building Elastic Container Registry (ECR)**
        - Similar to Dockerhub, it is a registry where we can run the ECS service on the image that we push on AWS.

1. Creating Elastic Container Registry

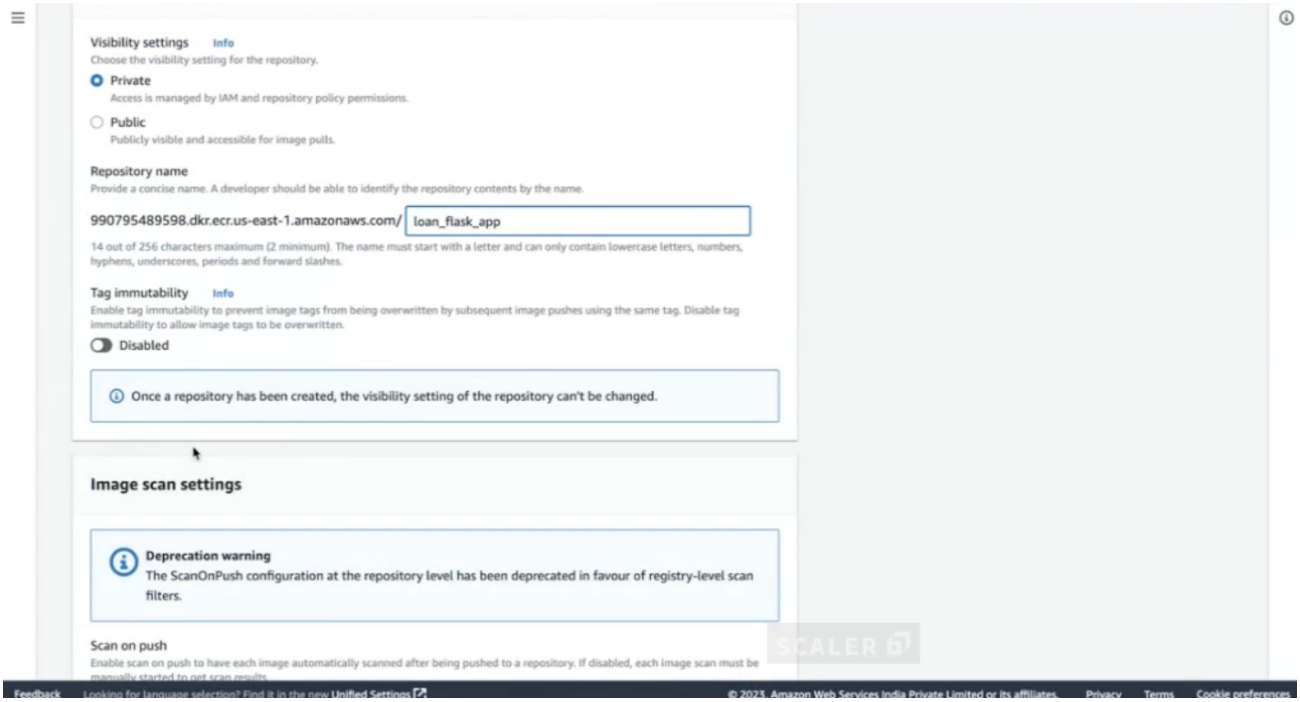
Here, we define some tasks that would select the appropriate container and need and traffic.

> **Instructor's Note:**
>
> Please ask students to check the region on which you're creating ECR. If the region is different, chances are you'll get an error message because of how services on AWS works based on regions.

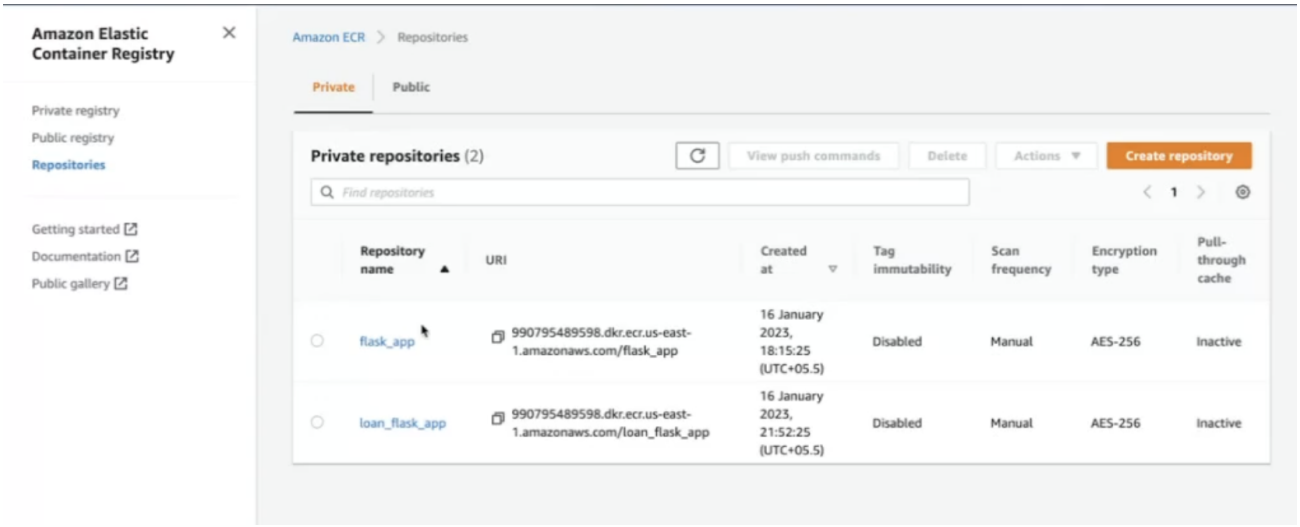
- When you log into AWS, search for ECR in the search box.



- Click on `Elastic Container Registry` and then click on `Get Started.`
- Click on `Create Repository.`
- We'll keep visibility as `private` as of now.
- Give a repository a name, and leave everything else as it is and click on `Create Repository` at the bottom of the page.



- You would now able to see your repositories like this:



2. Pushing Image

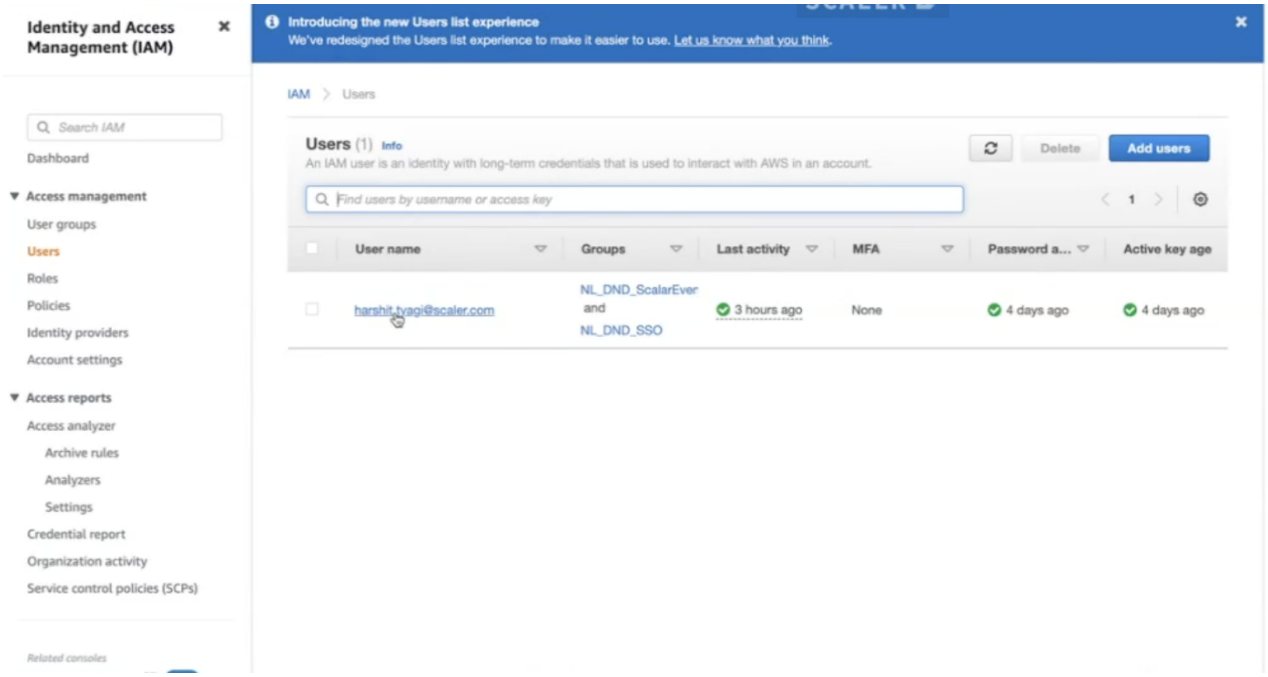
- Go to AWS CLI, and open the folder where your `dockerfile` is present.
- We'll now run `docker build` command. Since, we're operating on `mac machine`, the command would look something like this:

  ```
  docker build --platform=linux/amd64 --tag=loan_flask_app .
  ```

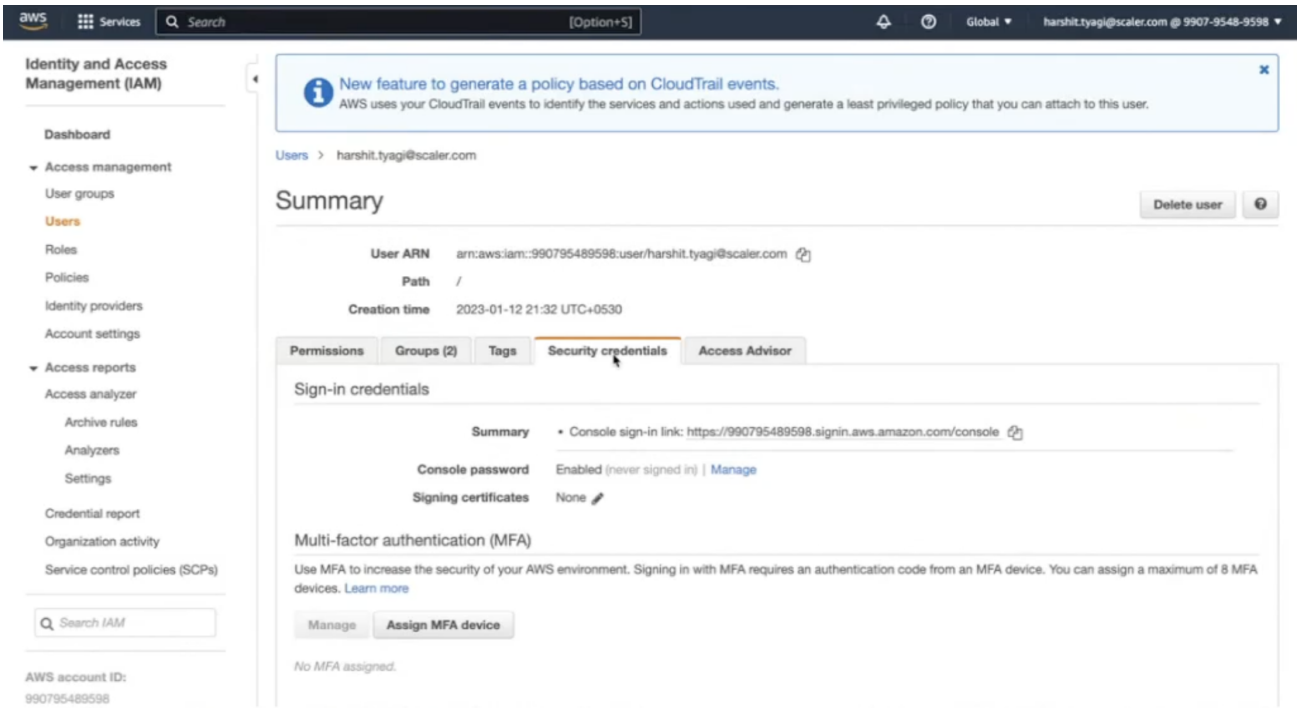
- In the above command, you would specify platform based on what machine you're using, and tag as your ECR repository name.
- Once the image has been built, run the command, and check if the container is working properly for the flask app that we built:

  ```
  docker run -p 8000:5000 -d loan_flask_app
  ```

- All this is working on your local machine and we need to push the image that we built on the ECR.
- For this you'll need to login to your AWS account using AWS CLI. For this use the command `aws configure` command.
- Next, search for `IAM` in the search bar on AWS page, and click on `Users` on left hand side. You'll see your user name like this:

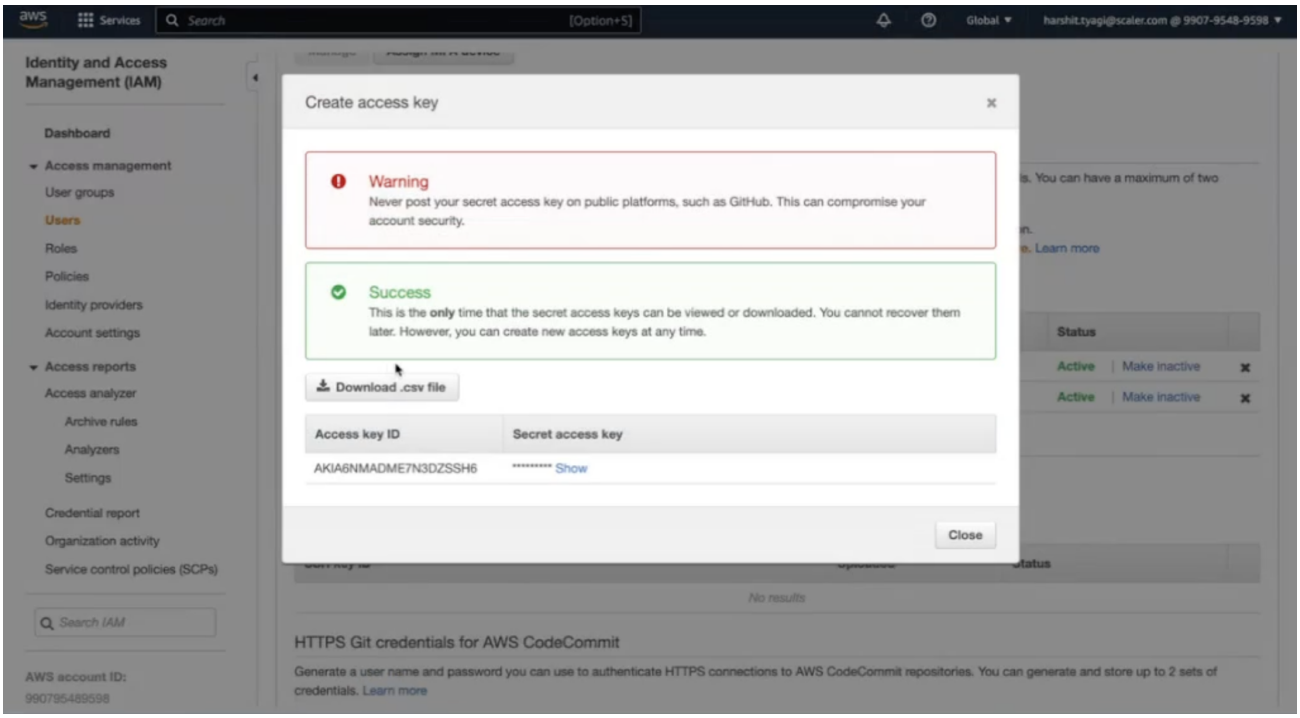


- Click on your user name. You would see a `Summary page`, where you should click on `Security Credentials`.



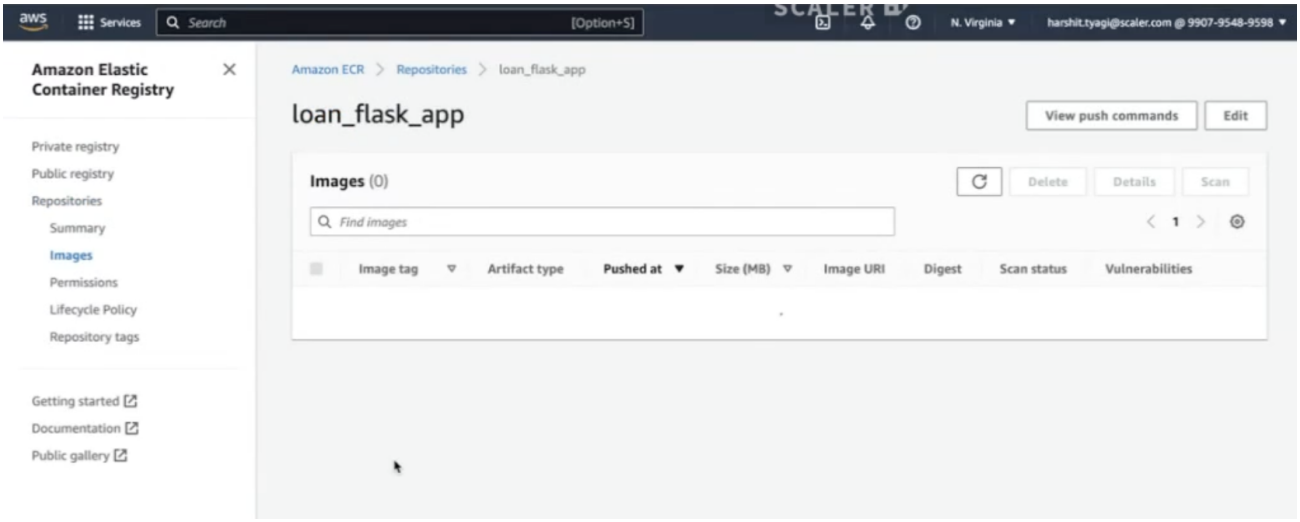
- Under `Access Keys` section, you'll see a single access key id.

- Click on `Create Access Key`, and you'll see a pop-up window something like this.

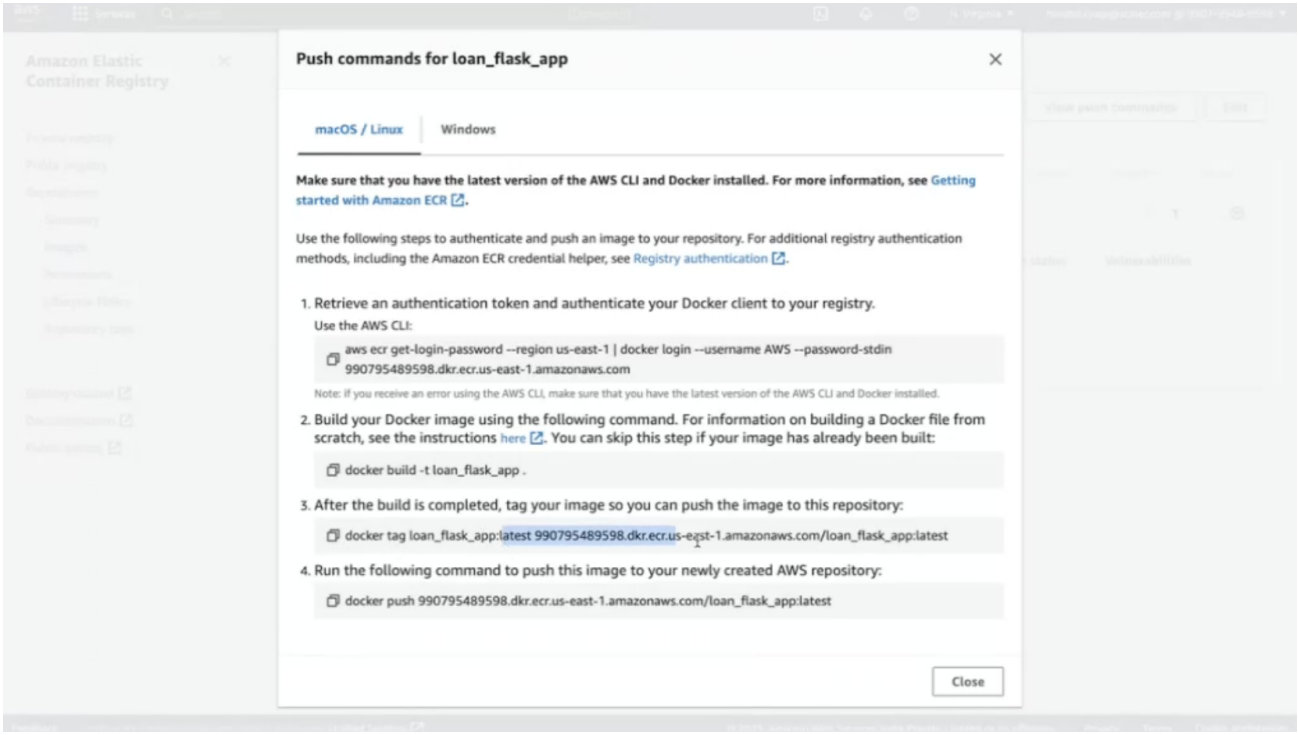


- Copy your Access Key ID, and paste it on AWS CLI which is asking for the same.
- Next, it will ask for secret access key, and you can again cpoy paste from the pop-up window, and paste on CLI.
- Set your **default region** as `us-east-1` if it is different, and **default output format** as `None`.

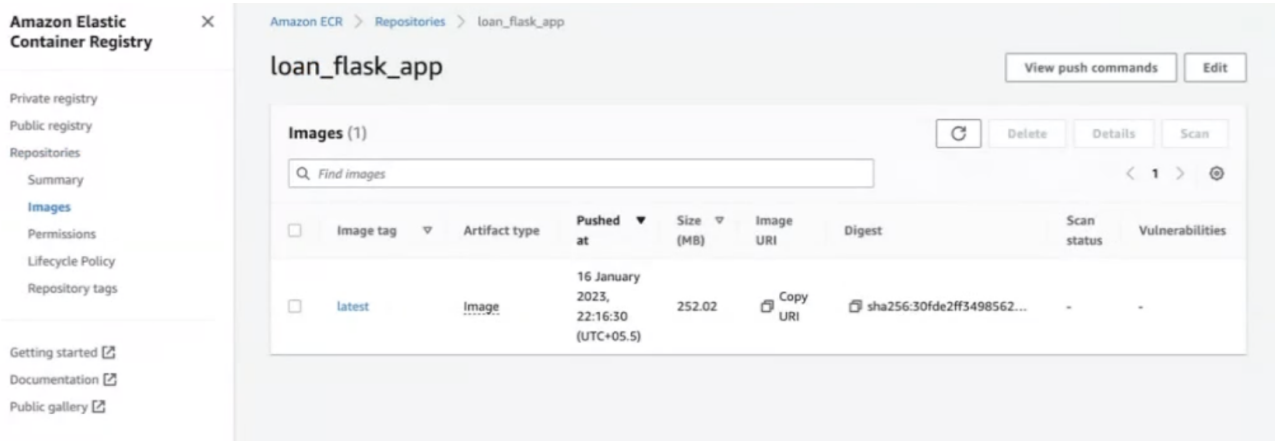
- Now, go to your ECR page where you can see your created repositories, and click on the one that we just built: `loan_flask_app`.
- You'll see a page like this:



- On top rigt side, you'll see `View Push Commands`. Click on that, and you'll see a set of commands for `Linux/Mac` and `Windows` machines.



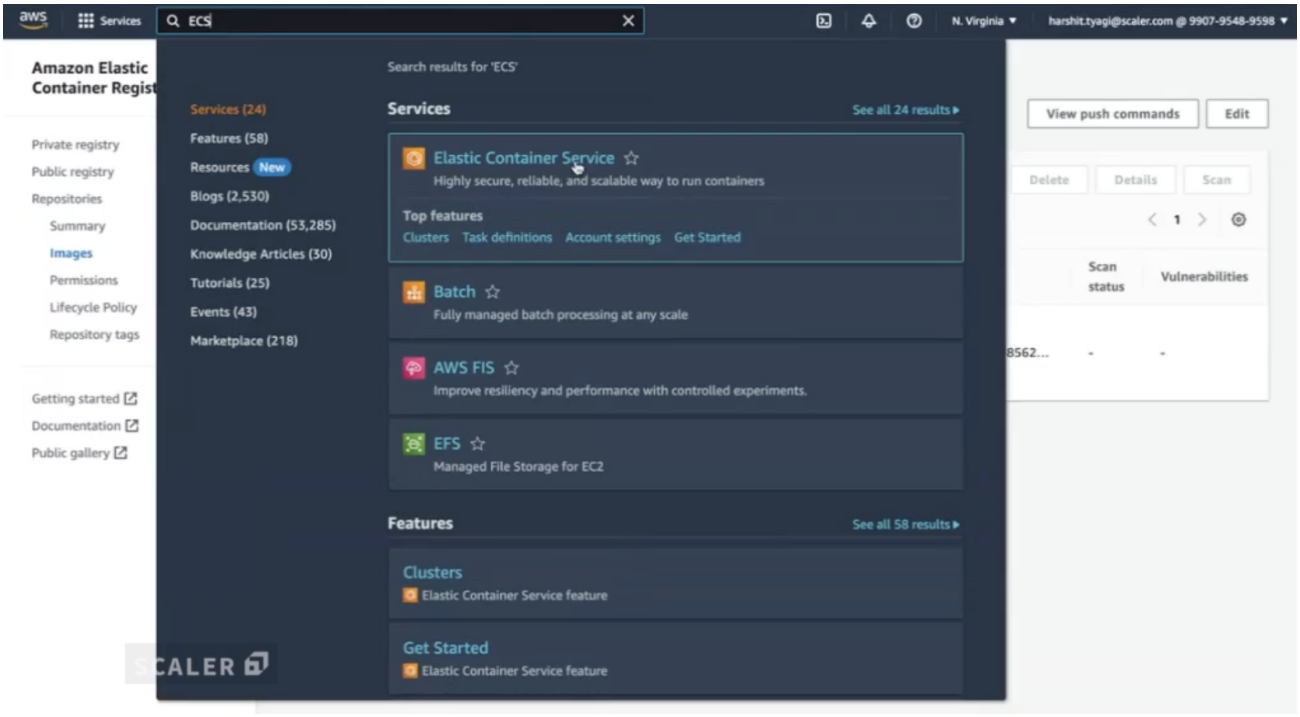
- Based on your machine, you'll need to simply copy paste these commands on to your AWS CLI.
- The first command will log into your AWS account. After running it, you should see a message saying `Login Succeeded`.
- Second command is asking for building docker image. Since, we've already done that, we'll skip this.
- The third command is simply tagging your image.
- The fourth command is pushing docker image to ECR repository. After running this, you would see AWS pushing your files.
- To verify, if image has been pushed, you can go to your repository page, and you would see an image there.



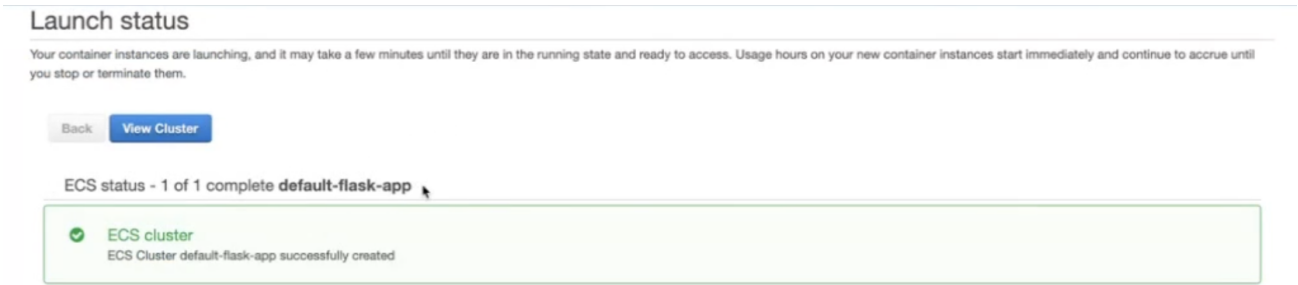
- Next thing to setup would be AWS Fargate clusters. Note that, as a data scientist, you would not require to set this up. All these things would be taken care by DevOps Engineers.

3. AWS Fargate

- On to your AWS console, search for `ECS`, and you'll see a service named as `Elastic Container Service`.



- Then, click on `Create Cluster`.
- Select your cluster template as `Networking only` and go to next step.
- Give your cluster a name, and leave rest everything as it is. Then, click on `Create` at the bottom of your page.
- You will see ECS Status like this:

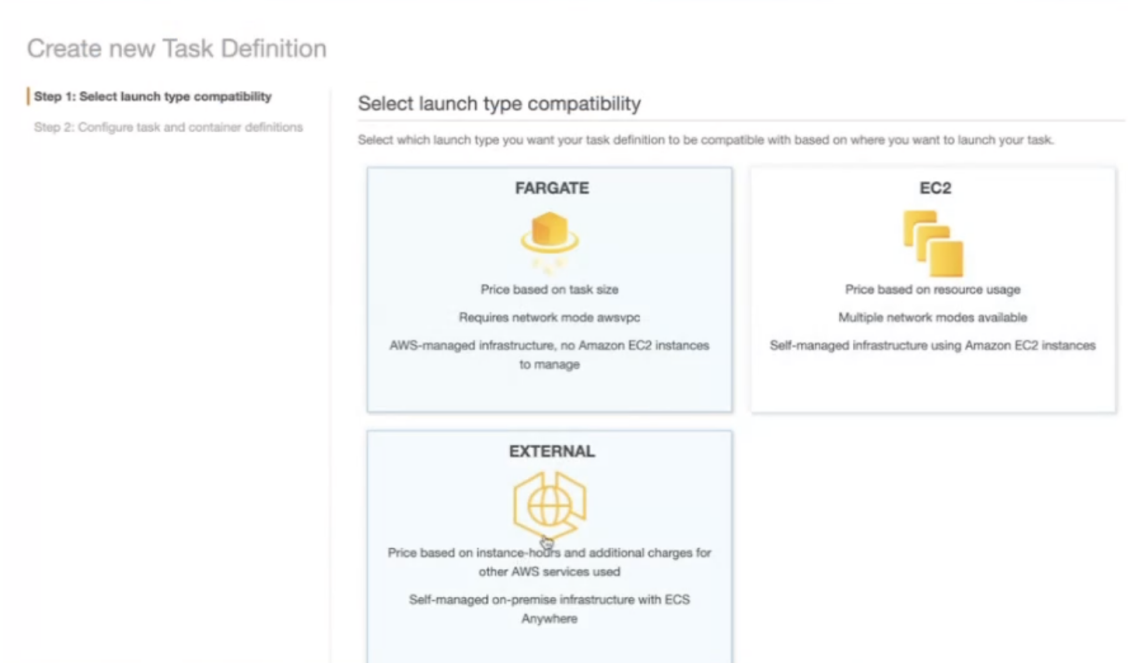


- Then, click on `View Cluster`.

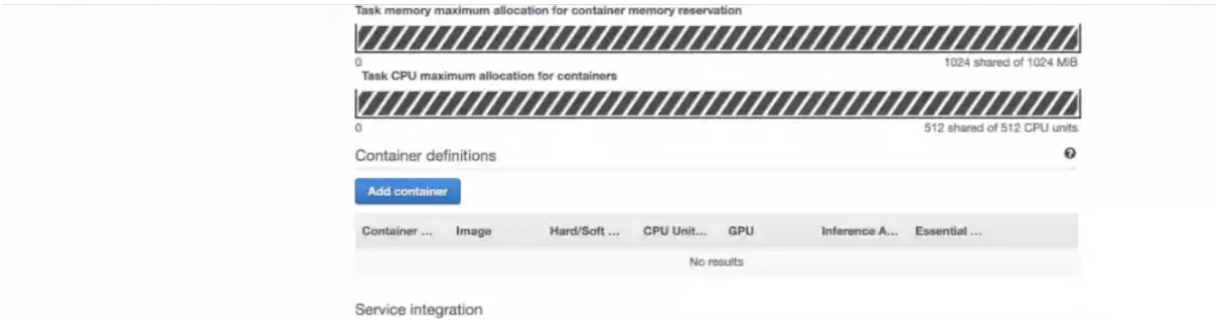
Now, in order to run flask application on this created cluster, you'll need to define tasks with the help of `ECS tasks`.

- On the left hand side of `ECS Console`, you'll see `Tasks Definitions`. Click on that.
- Then click on `Create new Task Definition`.

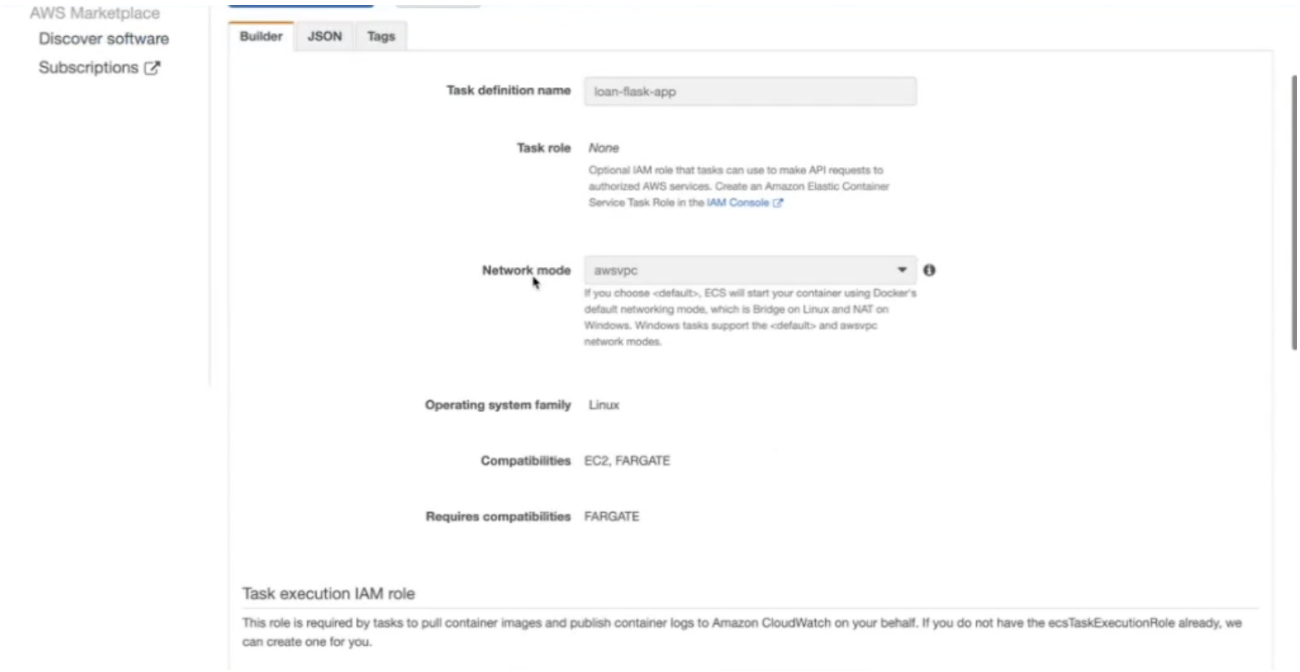
- Select `FARGATE` and go to next step.



- Enter a name for Task Definition, and leave `Task role` and `Network mode` as it is.
  - In task size, enter `Total memory` as **1GB**, and `Task CPU` as **0.5 vCPU** since our app is very lightweight.
  - Next is adding Container. On the same page, you'll see `Container Definitions`



- Click on `Add Container`.
  - A pop-up window will appear. Specify the name of the container under Container name.
  - For `image`, go to your ECR page, and copy `URI` of the corresponding repository, where we pushed the image of the container.
  - Paste that URI and add `:latest` to the last.
  - Next, under `Port Mapping` section, click on `Add port mapping`, and enter value **5000** with protocol as `tcp` only.
  - Leave everything else as it is and click on `Add` button.
- Then click on `Create` at the bottom. Once created, click on `View task definitions`. You'll see a page similar to this:



- Under `Actions` drop down, select `Run Task`.
  - Select `Launch type` as **Fargate**.
  - Under `Task Definition` check if correct one is being selected.
  - Select `Cluster VPC` as whatever is being displayed.
  - Under `Subnets`, select the first one in the drop down menu.
  - Under `Security Groups` click on `edit`.
    - Click on `Add Rule`.
      - Select `type` as `Custome TCP`, and `port range` as 5000 and hit `save`.
    - We'er doing this because our flask app runs on 5000 port.
  - Enable `Auto Assign public IP`. This will auto assign an IP address for the application to be accessed publicly.
- Click on `Run Task`. You'll see a page like this: