

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

| Topic |
|---------------------------------|
| Introduction |
| Problem Statement |
| Creating ECR |
| Pushing Image to ECR |
| Running tasks using AWS Fargate |

▼ 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?

2. Computing Device - Application Processor

3. Storage - RAM and SSD
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.

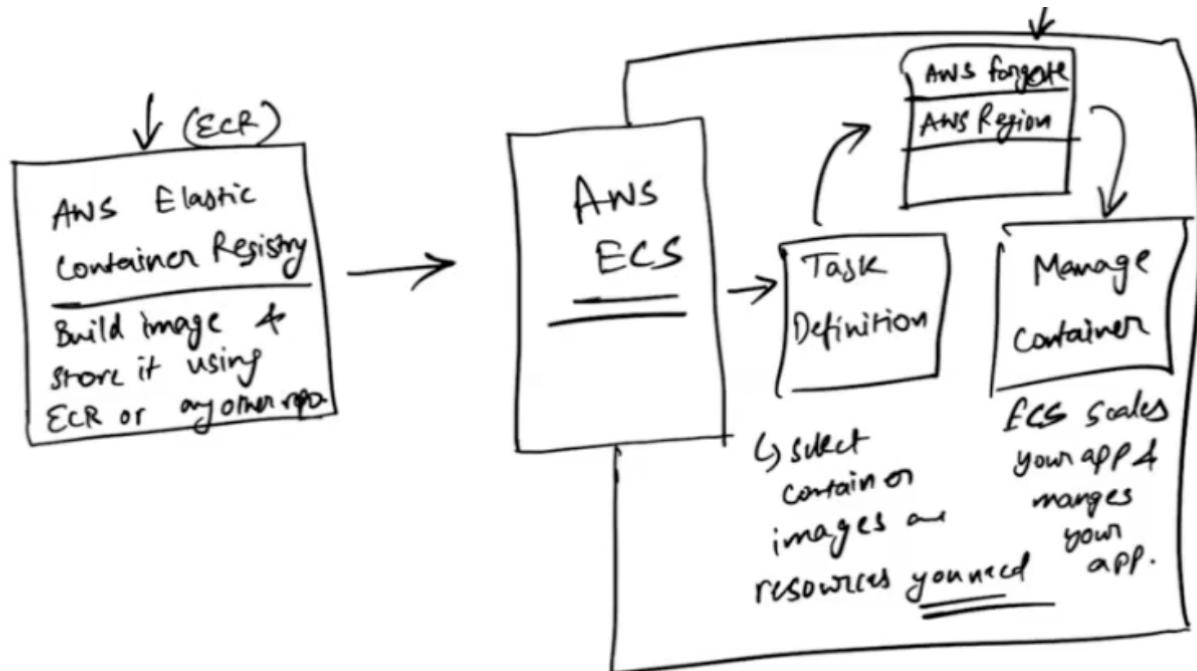
- More computing power/more storage means higher rent.

▼ 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 containers 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

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.

need and usage.

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

The screenshot shows the AWS IAM service interface with a search bar at the top. The search term 'ECR' has been entered. Below the search bar, there are two main sections: 'Services' and 'Features'. The 'Services' section lists 'Elastic Container Registry', 'Secrets Manager', 'Key Management Service', and 'Systems Manager'. The 'Features' section lists 'Private registry' and 'Repositories'. The 'Elastic Container Registry' card is expanded, showing its description: 'Fully-managed Docker container registry : Share and deploy container software, publicly or privately'. Below the card, there are buttons for 'Delete' and 'Add users'.

- 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.

The screenshot shows the 'Create Repository' wizard in the AWS ECR service. Step 1: 'Visibility settings' is set to 'Private'. Step 2: 'Repository name' is set to 'loan_flask_app'. Step 3: 'Tag immutability' is set to 'Disabled'. Step 4: 'Image scan settings' shows a deprecation warning: 'The ScanOnPush configuration at the repository level has been deprecated in favour of registry-level scan filters.' At the bottom, there is a note: 'Once a repository has been created, the visibility setting of the repository can't be changed.'

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

The screenshot shows the Amazon ECR interface. On the left, there's a sidebar with links like 'Private registry', 'Public registry', and 'Repositories'. The main area is titled 'Amazon ECR > Repositories' and shows 'Private repositories (2)'. There are tabs for 'Private' and 'Public'. Below is a table with columns: Repository name, URI, Created at, Tag immutability, Scan frequency, Encryption type, and Pull-through cache. Two rows are listed:

| Repository name | URI | Created at | Tag immutability | Scan frequency | Encryption type | Pull-through cache |
|-----------------|---|--------------------------------------|------------------|----------------|-----------------|--------------------|
| flask_app | 990795489598.dkr.ecr.us-east-1.amazonaws.com/flask_app | 16 January 2023, 18:15:25 (UTC+05:5) | Disabled | Manual | AES-256 | Inactive |
| loan_flask_app | 990795489598.dkr.ecr.us-east-1.amazonaws.com/loan_flask_app | 16 January 2023, 21:52:25 (UTC+05:5) | Disabled | Manual | AES-256 | Inactive |

▼ 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:

The screenshot shows the AWS IAM 'Users' page. On the left, there's a sidebar with navigation links like 'Dashboard', 'User groups', 'Users' (which is selected and highlighted in orange), 'Roles', 'Policies', 'Identity providers', and 'Account settings'. Below that is a 'Related consoles' section. The main content area has a header 'Users (1) Info' with a note: 'An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.' There's a search bar 'Find users by username or access key'. A table lists one user: 'User name' is 'harshit.tyagi@scaler.com', 'Groups' are 'NL_DND_ScalarEver' and 'NL_DND_SSO', 'Last activity' was '3 hours ago', 'MFA' status is 'None', and 'Password last used' and 'Active key age' were both '4 days ago'. There are buttons for 'Delete' and 'Add users'.

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

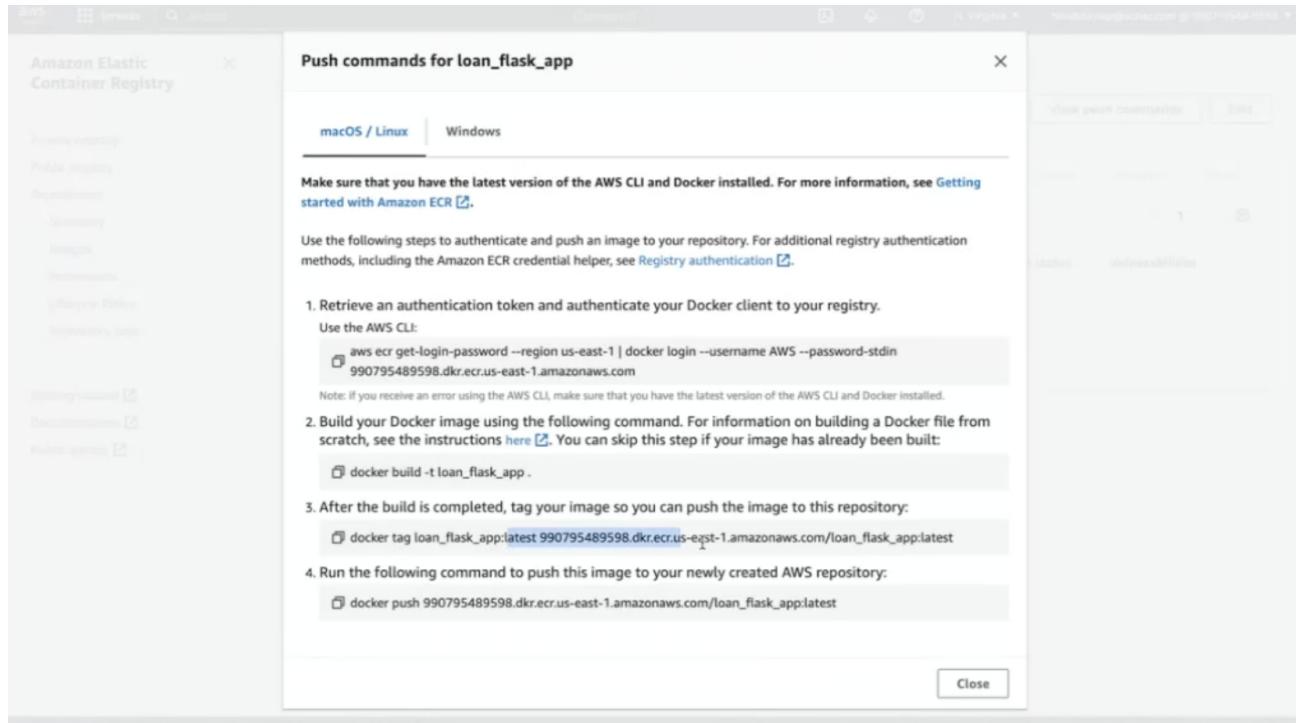
The screenshot shows the 'Summary' page for the user 'harshit.tyagi@scaler.com'. The left sidebar is identical to the previous screenshot. The main page has a banner about CloudTrail events. It shows the user ARN 'arn:aws:iam::990795489598:user/harshit.tyagi@scaler.com', Path '/', and Creation time '2023-01-12 21:32 UTC+0530'. Below this, there are tabs for 'Permissions', 'Groups (2)', 'Tags', 'Security credentials' (which is selected and highlighted in orange), and 'Access Advisor'. Under 'Sign-in credentials', it shows 'Console sign-in link: https://990795489598.signin.aws.amazon.com/console'. Under 'Multi-factor authentication (MFA)', it says 'No MFA assigned.' There are 'Manage' and 'Assign MFA device' buttons.

- 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.

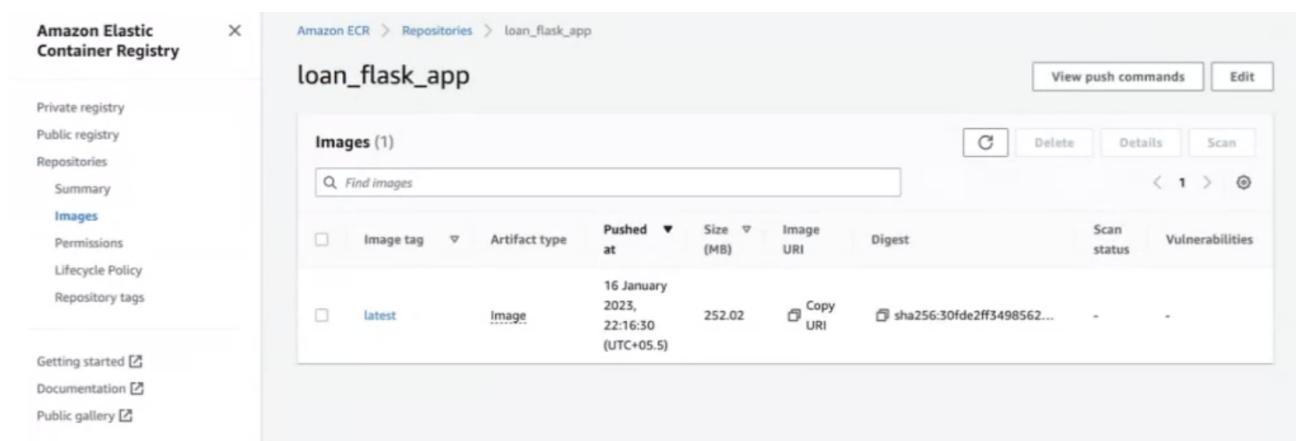
AWS account ID: 990785489598

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



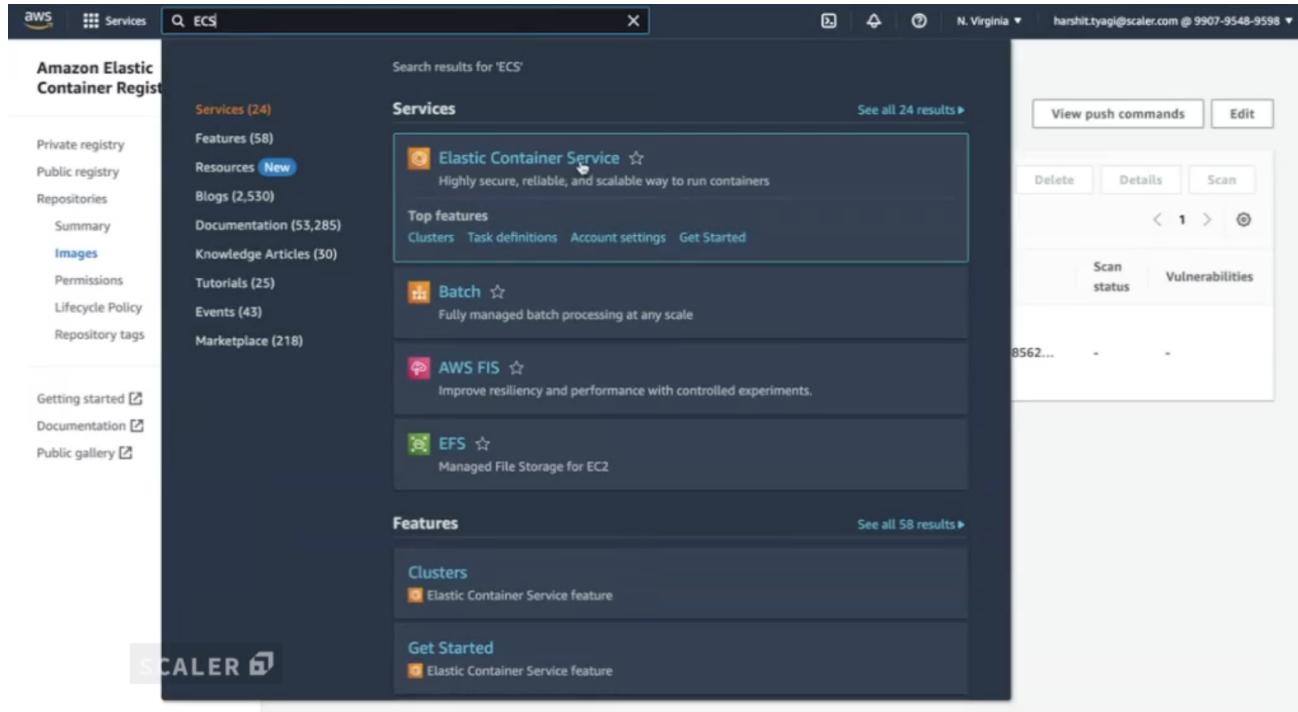
- Based on your machine, you'll need to simply copy paste these commands on to your AWS CLI.
- The first command will log into your AWS account. After running it, you should see a message saying `Login Succeeded`.
- Second command is asking for building docker image. Since, we've already done that, we'll 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.



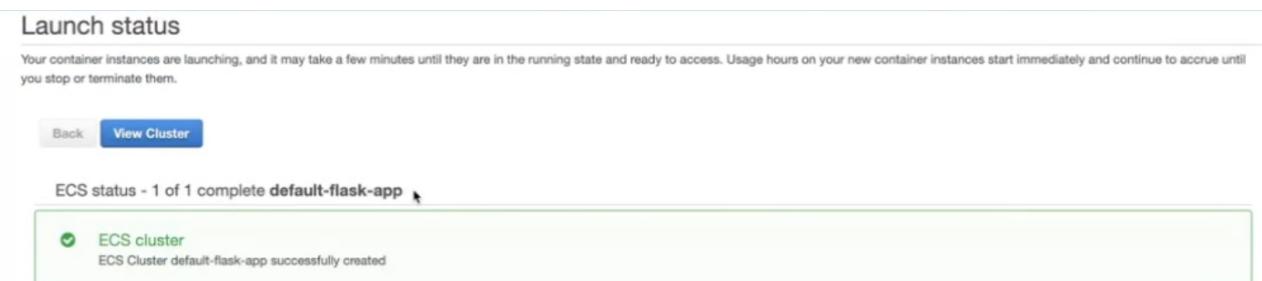
- Next thing to setup would be AWS Fargate clusters. Note that, as a data scientist, you would not require to set this up. All these things would be taken care by DevOps Engineers.

▼ 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.

Create new Task Definition

Step 1: Select launch type compatibility

Step 2: Configure task and container definitions

Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

| Launch Type | Description | Price | Infrastructure Management |
|-------------|---|--|----------------------------------|
| FARGATE | AWS-managed infrastructure, no Amazon EC2 instances to manage | Based on task size | Network mode: awsvpc |
| EC2 | Self-managed infrastructure using Amazon EC2 instances | Based on resource usage | Multiple network modes available |
| EXTERNAL | Self-managed on-premise infrastructure with ECS Anywhere | Based on instance-hours and additional charges for other AWS services used | N/A |

- 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:

The screenshot shows the AWS Lambda Task Definition Builder interface. The task definition name is set to 'loan-flask-app'. Under 'Network mode', 'awsvpc' is selected. Other settings include 'Operating system family: Linux', 'Compatibilities: EC2, FARGATE', and 'Requires compatibilities: FARGATE'. A note about the 'Task execution IAM role' is present.

- 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 Custom TCP , and port range as 5000 and hit save .
 - We're 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:

The screenshot shows a confirmation message from the AWS ECS Task Definition interface. It says 'Created tasks successfully' and provides the Task IDs: '[default/37e3f155f1eb40aaa10224b1f5e721d7]'