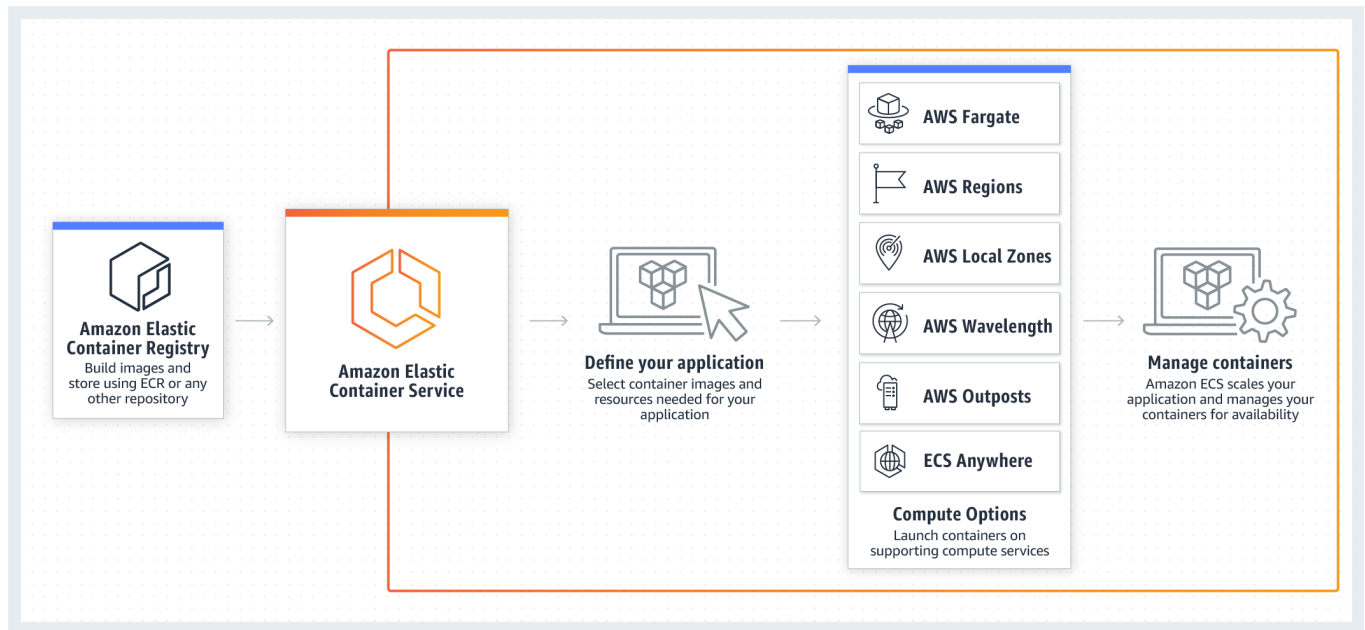


Deployment to AWS ECS

ECS - Elastic Container Service

Amazon Elastic Container Service (ECS) is a cloud computing service in Amazon Web Services (AWS) that manages containers and lets developers run applications in the cloud without having to configure an environment for the code to run in. It enables developers with AWS accounts to deploy and manage scalable applications that run on groups of servers called clusters through API calls and task definitions.



ECR - Elastic container repository

Amazon Elastic Container Registry (Amazon ECR) is an AWS managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM.

this is where we will store our docker image, and all other services will retrieve the resources and docker images from here itself.

To create an image repository

A repository is where you store your Docker or Open Container Initiative (OCI) images in Amazon ECR. Each time you push or pull an image from Amazon ECR, you specify the repository and the registry location which informs where to push the image to or where to pull it from.

Open the Amazon ECR console at <https://console.aws.amazon.com/ecr/>.

- Choose Get Started.
- For Visibility settings, choose Private.
- For Repository name, specify a name for the repository.
- For Tag immutability, choose the tag mutability setting for the repository. Repositories configured with immutable tags will prevent image tags from being overwritten. do not use this for now
- leave the rest as it is and click on create repository

Amazon ECR > Repositories > Create repository

Create repository

General settings

Visibility settings [Info](#)

Choose the visibility setting for the repository.

☒ **Private**

Access is managed by IAM and repository policy permissions.

☐ **Public**

Publicly visible and accessible for image pulls.

Repository name

Provide a concise name. A developer should be able to identify the repository contents by the name.


506732760924.dkr.ecr.ap-south-1.amazonaws.com/ flask_app

9 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

Tag immutability [Info](#)

Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

☒ **Disabled**

 Once a repository has been created, the visibility setting of the repository can't be changed.

after you create the repository you will be back on the repository listing page

It will show the list of the repo. Click the flask_app repo, or the name you choose. it will open an images page that lists all the images in that repository, but it will be empty now, because we do not have any image till now.

we will now push a docker image for the flask app that we have.

▼ Build and check docker image locally. (if docker image not already built)

docker file:

```
# syntax=docker/dockerfile:1

FROM python:3.8-slim-buster

WORKDIR /flask-docker

RUN python3 -m pip install --upgrade pip
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD ["python", "-m", "flask", "run", "--host=0.0.0.0"]
```

inside the folder with the required files and the docker file use the following command. the platform rg is important if you are using and arm powered device like the m1 mac else you can skip that argument in the command

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

```
[(flask_ecs) abduhad_scaler@Abduls-Air Flask_ecs % docker build --platform=linux/amd64 --tag=flask_app .
[+] Building 290.2s (15/15) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 167B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> resolve image config for docker.io/docker/dockerfile:1 3.9s
=> CACHED docker-image://docker.io/docker/dockerfile:1@sha256:9ba7531bd8 0.0s
=> [internal] load .dockerignore 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> [internal] load metadata for docker.io/library/python:3.8-slim-buster 3.2s
=> [1/6] FROM docker.io/library/python:3.8-slim-buster@sha256:db90d6607 38.3s
=> => resolve docker.io/library/python:3.8-slim-buster@sha256:db90d6607a 0.0s
=> => sha256:77c3488ce5476695aa0b018e6dcfe6c4a2bf2e5d1e4 1.37kB / 1.37kB 0.0s
=> => sha256:5cc94b67760dc542cb56567c7d7f5d9d19d9ec4bf6d 7.53kB / 7.53kB 0.0s
=> => sha256:32820e52a00eb22dc76d70d992be7082cd24b636 27.14MB / 27.14MB 36.6s
=> => sha256:6563993b0507bc602c27e376c26abd7766a8c8fa09 2.78MB / 2.78MB 11.6s
=> => sha256:eae6fe00ee3c3d4a62faa08362ff949fcf2fcb41 11.29MB / 11.29MB 23.0s
=> => sha256:db90d6607a1c2ccb9bbc8a6fdf286d183c44bf8e9c92193 988B / 988B 0.0s
=> => sha256:bbcf4b417983bad30035c8d28bae1dbca13f8e7636def1 234B / 234B 12.2s
=> => sha256:b2f1e8a99da34fd5fdbc38fcf88bd78ab7d248f3d2 3.18MB / 3.18MB 20.2s
=> => extracting sha256:32820e52a00eb22dc76d70d992be7082cd24b636cfb597ff 0.9s
=> => extracting sha256:6563993b0507bc602c27e376c26abd7766a8c8fa094da83b 0.1s
```

additionally you can run and check the app locally

now we need to push this to ecr repo

AWS Permissions and access

- ECS full access
- ECR full access
- Cloud Watch
- cloud Formation

▼ AWS CLI

Install the AWS CLI

You can use the AWS command line tools to issue commands at your system's command line to perform Amazon ECR and other AWS tasks. This can be faster and more convenient than using the console. The command line tools are also useful for building scripts that perform AWS tasks.

<https://aws.amazon.com/cli/>

according to your system use the above link to install the cli

AWS Command Line Interface

The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

The AWS CLI v2 offers several [new features](#) including improved installers, new configuration options such as AWS IAM Identity Center (successor to AWS SSO), and various interactive features.



[Getting Started »](#)



[AWS CLI Reference](#)
»



[GitHub Project »](#)



[Community Forum »](#)

Windows

Download and run the [64-bit Windows installer](#).

MacOS

Download and run the [MacOS PKG installer](#).

Linux

Download, unzip, and then run the [Linux installer](#)

Amazon Linux

The AWS CLI comes pre-installed on [Amazon Linux AMI](#).

Release Notes

Check out the [Release Notes](#) for more information on the latest version.

▼ Push docker to ECR

Build, tag, and push a Docker image

You use the Docker CLI to tag an existing local image and then push the tagged image to your Amazon ECR registry.

- Select the repository you created and choose View push commands to view the steps to push an image to your new repository.
- Run the login command that authenticates your Docker client to your registry by using the command from the console in a terminal window. This command provides an authorization token that is valid for 12 hours.

use the following command

```
aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]:
```

after this you can use the push commands from the repository to move forward.

directly copy those commands and run them, you dont need to change anything

Push commands for flask_app



macOS / Linux

Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see [Getting started with Amazon ECR](#).

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see [Registry authentication](#).

1. Retrieve an authentication token and authenticate your Docker client to your registry.

Use the AWS CLI:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin  
506732760924.dkr.ecr.us-east-1.amazonaws.com
```

Note: if you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch, see the instructions [here](#). You can skip this step if your image has already been built:

```
docker build -t flask_app .
```

3. After the build is completed, tag your image so you can push the image to this repository:

```
docker tag flask_app:latest 506732760924.dkr.ecr.us-east-1.amazonaws.com/flask_app:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push 506732760924.dkr.ecr.us-east-1.amazonaws.com/flask_app:latest
```

```
(flask_ecs) abduhad_scaler@Abduls-Air Flask_ecs % aws configure  
AWS Access Key ID [None]: AKIAXL65AJ50CLCWJ560  
AWS Secret Access Key [None]: oAz3yk/S7VuacthrfSCs/KLd+NRFP6U5YgC9LvLO  
Default region name [None]: us-east-1  
Default output format [None]:  
(flask_ecs) abduhad_scaler@Abduls-Air Flask_ecs % aws ecr get-login-password --  
-region us-east-1 | docker login --username AWS --password-stdin 506732760924.dkr.ecr.us-east-1.amazonaws.com  
Login Succeeded  
(flask_ecs) abduhad_scaler@Abduls-Air Flask_ecs %
```



```
[(flask_ecs) abduhad_scaler@Abduls-Air Flask_ecs % docker push 506732760924.dkr]
.ecr.us-east-1.amazonaws.com/flask_app:latest
The push refers to repository [506732760924.dkr.ecr.us-east-1.amazonaws.com/flas
k_app]
463880ed5ad1: Pushed
fd054f25fb1e: Pushed
32dc62f1e25e: Pushed
9edf2546b63c: Pushed
d457755e6d31: Pushed
7f2102a09744: Pushed
6ba5538cb764: Pushed
ffc2134468e0: Pushed
a332d6832c82: Pushed
3ebd45c2fd2c: Pushed
latest: digest: sha256:76e19987b1215a603dfe9564649055f0d572efef5a9ab57400ae184e1
d76739a size: 2421
(flask_ecs) abduhad_scaler@Abduls-Air Flask_ecs %
```

Create a Fargate Cluster.

AWS Fargate

AWS Fargate is a serverless, pay-as-you-go compute engine that lets you focus on building applications without managing servers, it is a serverless compute for containers, which eliminates the need to configure and manage control plane, nodes, and instances.

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
- 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.
- we'll name the cluster default, and the rest we can leave as is.

▼ 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 let's go back to the ECS page and do the following:

- Select Task Definitions from the left menu.
- Then select Create new Task Definition
- Select Fargate
- Select Next Step
- Enter a name for the task. I am going to use flask_app.
- Leave Task Role and Network Mode set to their default values.
- Leave Task Execution Role set to its default.
- For Task memory and Task CPU select 0.5 v cpu and 1 gb memory. that will be sufficient for our usecase

Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on your behalf. If you do not have the `ecsTaskExecutionRole` already, we can create one for you.

Task execution role ecsTaskExecutionRole ⓘ

Task size ⓘ

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (GB) 1GB ▼

The valid memory range for 0.5 vCPU is: 1GB - 4GB, in 1GB increments.

Task CPU (vCPU) 0.5 vCPU ▼

The valid CPU range for 1GB memory is: 0.25 vCPU - 0.5 vCPU.

Task memory maximum allocation for container memory reservation



Task CPU maximum allocation for containers



Container definitions ⓘ

- Under Container definition select Add Container.
- Enter a Container name. we will use `flask_app`. In the Image box enter the ARN of our image. You will want to copy and paste this from the ECR dashboard .
- 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.

▼ Standard

Container name* flask_app ⓘ

Image* 506732760924.dkr.ecr.us-east-1.amazonaws.com/flask_app:latest ⓘ

- 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.

Run the ECS Task!

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

Task Definition: flask_app:1

View detailed information for your task definition. To modify the task definition, you need to create a new revision and then make the re task definition

Create new revision

Actions ▾

Builder

JSON

Tags

Run Task

Create Service

Update Service

Deregister

flask_app

Task role

None

Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#)

Run Task

Select the cluster to run your task definition on and the number of copies of that task to run. instances, click Advanced Options.

Launch type ☒ FARGATE ☐ EC2 ☐ EXTERNAL 

AWS Fargate is migrating service quotas from the current Amazon ECS task count-based quotas to vCPU-based quotas. [To learn more, refer to the AWS Fargate FAQs.](#)

[Switch to capacity provider strategy](#) 

Operating system family

Task Definition

Family

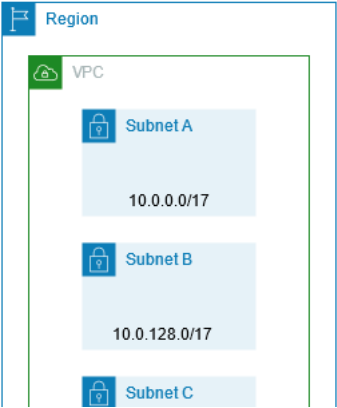
flask_app

- For Launch type: select Fargate
- Make sure Cluseter: is set to the default we created earlier.
- Cluster VPC select a vpc from the list.
- Add at least one subnet.
- Auto-assign public IP should be set to ENBABLED

▼ VPC and subnets

A virtual private cloud (VPC) is a virtual network dedicated to your AWS account. It is logically isolated from other virtual networks in the AWS Cloud. You can specify an IP address range for the VPC, add subnets, add gateways, and associate security groups.

A subnet is a range of IP addresses in your VPC. You launch AWS resources, such as Amazon EC2 instances, into your subnets. You can connect a subnet to the internet, other VPCs, and your own data centers, and route traffic to and from your subnets using route tables.



VPC and security groups

