

# The Process of Containerising and Deploying a Flask Application

The general overview of deploying a flask app using ECS was as follows:

1. Write a flask app. The one I wrote can be found in the our Github repository.
2. Containerise the flask app with Docker. This involves writing a Docker file (also on Github) and building the Docker image of the application.
3. Deploying the Docker image on ECS. This involves pushing the the image onto an AWS repository (ECR), creating an ECS cluster, creating a *class definition*, and running the class definition on the ECS cluster.

## Stage 1 - Writing a Flask App

Main things to consider are making sure Python and a pip installer are installed on your machine. From there you can create a new project directory and a virtual environment with

```
mkdir projectFolder && cd projectFolder
python3 -m venv ./venv
```

We can then activate the virtual environment with

```
source ./venv/bin/activate
```

(To later deactivate the virtual environment we can do `deactivate` )

So now all python installations will be specific to this virtual environment.

Run `pip3 install Flask`

For **Windows** commands, they can be found here [🔗 Installation — Flask Documentation \(3.1.x\)](#)

From there, we can write the Flask application. A simple app will have only one route. Create a file **app.py** and the following code.

```
app.py > ...
1 from flask import Flask, render_template
2 app = Flask(__name__)
3
4 @app.route('/')
5 def index():
6     # only need the file name because render_template knows to look in the directory called "templates"
7     return render_template('index.html')
8
9 if __name__ == "__main__":
10     app.run(host='0.0.0.0', port=5001, debug = True)
11
```

Code for app.py

An important thing to note is that the **host='0.0.0.0'**.

By following a Flask tutorial, I also saw how to render html files with a Flask app. This is outside the scope of this document, but in essence, you create a directory called **templates** and an html file (**index.html**) in templates. In the **app.py** file, we import `render_template` from flask and use this function to render our HTML file. The exact structure of these files can be seen in the Github repository. If not interested in rendering an html file, replace

```
render_template(index.html) with "Hello World"
```

## Stage 2 - Creating a Docker Image of the Flask App

I would recommend downloading the **Docker** application from [🔗 Docker Desktop: The #1 Containerization Tool for Developers | Docker](#) to make it easy to visualise what is going on when you containerise the Flask app. I believe you are able to do everything from the terminal and the application is quite heavy (~1.8GB) so this step is optional.

Create a file **requirements.txt** in your working directory. This is where your python packages go. We only have flask, so that is all we have to write.

```
requirements.txt
1 flask
```

Create a **Dockerfile**. Be sure to name it exactly "Dockerfile". This file is the instructions to run when building your docker image. The particular Dockerfile I have is as follows

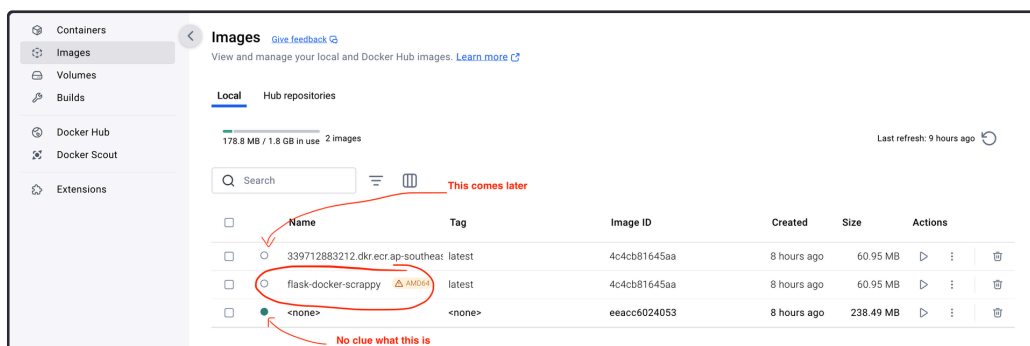
```
1 FROM python:3.8-slim-buster
2
3 WORKDIR /python-docker
4
5 COPY requirements.txt requirements.txt
6 RUN pip3 install -r requirements.txt
7 EXPOSE 5001
8 COPY . /python-docker
9 ENTRYPOINT [ "python3" ]
10 CMD [ "app.py" ]
```

We now run the **build** command:

```
docker build -t flask-docker-scrappy --platform linux/amd64 .
```

- The **-t** option is to give the docker image a *tag*
- The **--platform linux/amd64** part is to specify what kind of operating system we want the image to be working on. This part is *not* necessary for building the image, but was required when getting the image to run on AWS ECS. To save time, add it here.
- **.** is to specify the working directory as the

If you downloaded the Docker Desktop application, you should now see the following:

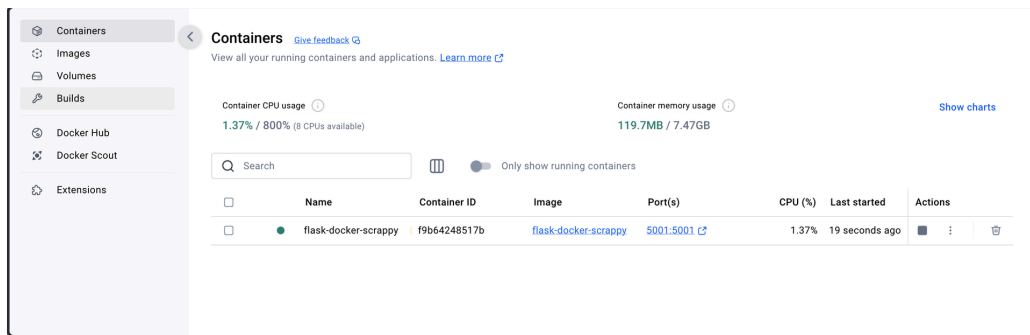


We can run the docker image (not necessary for deployment, but nice to see how it works) using the following **run** command

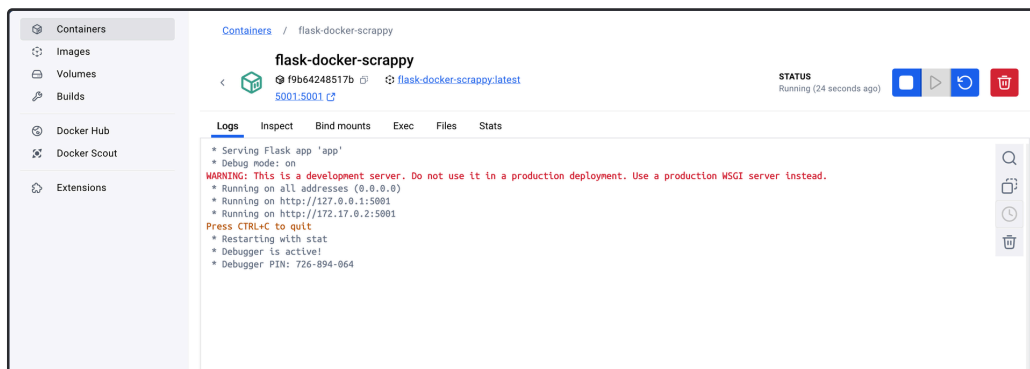
```
docker run --name flask-docker-scrappy -p 5001:5001 flask-docker-scrappy
```

- The **--name flask-docker-scrappy** part tells us what to name the *container*; we could name it whatever we like
- The **-p 5001:5001** part is for port forwarding; it is what allows us to access port 5001 on the Docker machine.
- We specify the image we want to run - in this case **flask-docker-scrappy**

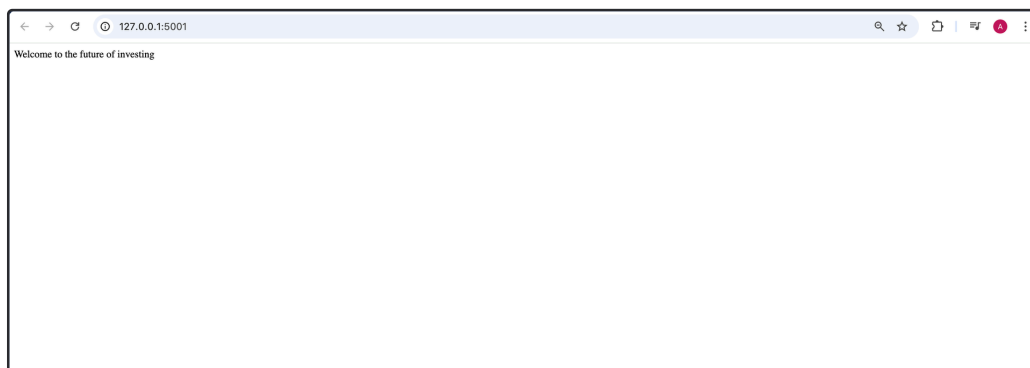
You should now have something that looks like this in the Docker Desktop application



Clicking on the name of the container (flask-docker-scrappy) should take you to a terminal view



and clicking on the 127.0.0.1 address should give you access to the Flask app



So this indicates that we have successfully built a Docker image.

## Stage 3 Deploying the Docker Image on ECS [↗](#)

Unfortunately, this part was annoying convoluted. There are a few AWS configuration/set up steps that have to be done. If not done already, create an AWS account. From there, we need to work with several services.

### IAM [↗](#)

We need to create an IAM user to facilitate uploading our image to ECR. Go to the IAM service, click on the number under users and create a new user.

IAM Dashboard
Info

### Security recommendations 0

- Root user has MFA  
Having multi-factor authentication (MFA) for the root user improves security for this account.
- Root user has no active access keys  
Using access keys attached to an IAM user instead of the root user improves security.

### IAM resources

Resources in this AWS Account

|             |       |       |          |                    |
|-------------|-------|-------|----------|--------------------|
| User groups | Users | Roles | Policies | Identity providers |
| 0           | 1     | 5     | 0        | 0                  |

### What's new


Updates for features in IAM

- AWS IAM announces support for encrypted SAML assertions. 4 weeks ago
- AWS CodeBuild announces support for project ARN and build ARN IAM condition keys. 1 month ago
- IAM Roles Anywhere credential helper now supports TPM 2.0. 2 months ago
- Announcing AWS STS support for ECDSA-based signatures of OIDC tokens. 3 months ago

[View all](#)

Click on this number in the IAM service

Name the user whatever you like. The important thing is to provide relevant permissions to the user. The permissions my user used were AdministratorAccess, AmazonEC2ContainerRegistryFullAccess and AmazonECS\_FullAccess. However I believe **AmazonEC2ContainerRegistryFullAccess** is the only one that is required, and the other two came from trying to fix a different problem and are actually not necessary for his task.

|                          |  |                            |          |
|--------------------------|--|----------------------------|----------|
| <input type="checkbox"/> |  <a href="#">AdministratorAccess</a>                  | AWS managed - job function | Directly |
| <input type="checkbox"/> |  <a href="#">AmazonEC2ContainerRegistryFullAcc...</a> | AWS managed                | Directly |
| <input type="checkbox"/> |  <a href="#">AmazonECS_FullAccess</a>                 | AWS managed                | Directly |

The permissions I gave to my IAM user

Now that the IAM user has been created, we must download the AWS CLI from [Installing or updating to the latest version of the AWS CLI - AWS Command Line Interface](#)

Once the AWS CLI has been installed, check it has been successful by running the following in your terminal

```
aws help
```

and you should get output like

```
AWS()
NAME
aws -
DESCRIPTION
The AWS Command Line Interface is a unified tool to manage your AWS
services.
```

From there, we need to connect our CLI to our IAM user. We need to create an **access key** for our new IAM user. Go to the IAM service again, click on the number under users, select your newly created IAM user and click **Create access key**.

| Summary   |                            |   |
|---|----------------------------|---|
| ARN<br>arn:aws:iam::339712883212:user/firstMate | Console access<br>Disabled | Access key 1<br>AKIAU6GDXXHIGJ6AWQDNT - Active<br>Used 8 hours ago, 13 hours old. |
| Created<br>March 01, 2025, 19:24 (UTC+11:00)    | Last console sign-in<br>-  | Access key 2<br><a href="#">Create access key</a>                                 |

“Create access key” should be under Access key 1; since I have created an access key, for me it is under Access key 2

Go through the process. By the end you should have this screen:

### Retrieve access keys [Info](#)

**Access key**  
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

| Access key            | Secret access key          |
|-----------------------|----------------------------|
| AKIAU6GDXXHIGJQGF7Y7V | ***** <a href="#">Show</a> |

Both the access key and the secret access key will be required. Note that if you exit this page, you will **not** be able to retrieve the secret access key and will have to create a new access key.

In your terminal, run

```
aws configure
```

It will ask for an access key. Provide the one you created. It will then ask for a secret access key. Provide the one you created. Your AWS CLI is now being “used” by the IAM role.

## ECR [↗](#)

This is where we will place our Docker image. Go to the ECR service and create a new **private repository**. Keep everything as default, provide a repository name. When you go back to ECR, you should be able to see your new repository under the **private repositories tab**. If you click on the repository, you should see a button called **push commands**.

**Push commands for seng3011/scrappyapp** ×

[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](#).

- Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

```
aws ecr get-login-password --region ap-southeast-2 | docker login --username AWS --password-stdin 339712883212.dkr.ecr.ap-southeast-2.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.
- 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 is already built:

```
docker build -t seng3011/scrappyapp .
```
- After the build completes, tag your image so you can push the image to this repository:

```
docker tag seng3011/scrappyapp:latest 339712883212.dkr.ecr.ap-southeast-2.amazonaws.com/seng3011/scrappyapp:latest
```
- Run the following command to push this image to your newly created AWS repository:

```
docker push 339712883212.dkr.ecr.ap-southeast-2.amazonaws.com/seng3011/scrappyapp:latest
```

Follow these instructions, and you should be able to push your Docker image to the repository.

| Images (6)                                    |           |               |                                  |           |                          |  | <a href="#">Delete</a> <a href="#">Details</a> <a href="#">Scan</a> <a href="#">View push commands</a> |
|---|-----------|---------------|----------------------------------|-----------|--------------------------|--|--|
| <input type="text" value="Search artifacts"/> |           |               |                                  |           |                          |  | <a href="#">&lt;</a> <a href="#">1</a> <a href="#">&gt;</a>  |
| <input type="checkbox"/>                      | Image tag | Artifact type | Pushed at                        | Size (MB) | Image URI                | Digest                                   |  |
| <input type="checkbox"/>                      | latest    | Image Index   | 02 March 2025, 00:17:01 (UTC+11) | 60.95     | <a href="#">Copy URI</a> | sha256-4c4cb81645aa8ca874d9c8d7289e9...  |  |
| <input type="checkbox"/>                      | -         | Image         | 02 March 2025, 00:17:01 (UTC+11) | 0.00      | <a href="#">Copy URI</a> | sha256-7c1924dc78f53ac19d810f629454d7... |  |
| <input type="checkbox"/>                      | -         | Image         | 02 March 2025, 00:17:01 (UTC+11) | 60.95     | <a href="#">Copy URI</a> | sha256-66eb71ac36a6d999a5d5a21c883...    |  |
| <input type="checkbox"/>                      | -         | Image Index   | 02 March 2025, 00:06:20 (UTC+11) | 59.69     | <a href="#">Copy URI</a> | sha256-eeacc602405517800627efe799c395... |  |
| <input type="checkbox"/>                      | -         | Image         | 02 March 2025, 00:06:19 (UTC+11) | 0.00      | <a href="#">Copy URI</a> | sha256-b7bdc7d517018beccbd5af689b3c2...  |  |
| <input type="checkbox"/>                      | -         | Image         | 02 March 2025, 00:06:19 (UTC+11) | 59.69     | <a href="#">Copy URI</a> | sha256-a25600b6444bc3527bd7cf1609abc...  |  |

Under the images tab - since I pushed twice I have 6 items; after pushing once you should get 3 items

Click on **Copy URI** for the **latest image index**. We will need this later.

## ECS [↗](#)

Everything written in the ECS section is done in this tutorial: [📺 How to Deploy Flask App on AWS ECS using AWS Fargate](#)

[e?](#)

Go to ECS, go to **Clusters** and create a new cluster. Provide a name for the cluster, keep Fargate selected for your infrastructure but keep everything else default.

Now go to **task definitions**. Create a new task definition. There are a few important configurations to make here.

### Platform

Ensure **Linux/X86\_64** is selected because this is the platform we specified the Docker image to be when we built it.

### Task Execution Role

Ensure **Create new role** is selected so that a role with the necessary permissions to run this task is created

▼ Infrastructure requirements

Specify the infrastructure requirements for the task definition.

Launch type

Info

Selection of the launch type will change task definition parameters.

☒ AWS Fargate  
Serverless compute for containers.

☐ Amazon EC2 instances  
Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode

Network mode is used for task and is dependent on the compute type selected.

Operating system/Architecture

Info

Linux/X86\_64

Network mode

Info

swogpc

Task size

Info

Specify the amount of CPU and memory to reserve for your task.

CPU

.25 vCPU

Memory

.5 GB

▼ Task roles - conditional

Task role

Info

A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the IAM console.

-

Task execution role

Info

A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role created, we can create one for you.

Create new role

▼ Task placement - optional

Task placement constraints are not supported for AWS Fargate launch type.

► Fault injection - optional

### Image URI

Paste the image URI you previously copied (can be found in ECR) here. Give whatever name you like.

▼ Container - 1

Info

Essential container

Remove

Container details

Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name

wordpress

Image URI

repository-uri/image-tag

Essential container

Yes

Private registry

Info

Stores credentials in Secrets Manager, and then use the credentials to reference images in private registries.

☐ Private registry authentication

Best practices

Info

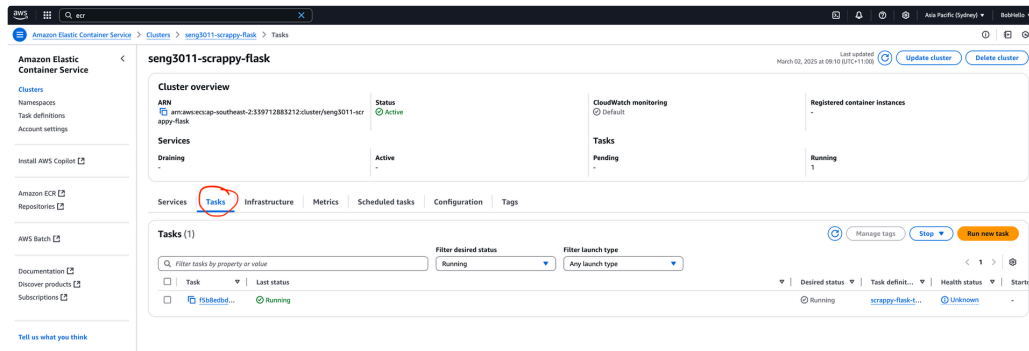
### Port

Give the same port that you were running on the Flask app and the Docker image.

The screenshot shows the 'Add port mapping' dialog in the Amazon ECS console. It has four input fields: 'Container port' with the value '5001', 'Protocol' with a dropdown set to 'TCP', 'Port name' with the value 'container-port-protocol', and 'App protocol' with a dropdown set to 'HTTP'. There is a 'Remove' button to the right of the 'App protocol' dropdown and an 'Add port mapping' button at the bottom left.

Everything else can be left as default. Create the task definition.

This task definition tells us that we *should* run the Docker image located in our ECR repository. However, the image is not actually running yet. We go back to our cluster, click on the **Tasks** tab and click **Run new task**.



The tutorial I followed selected **Launch type** instead of **Capacity provider strategy** but I am not sure that makes a difference.

The two important things to do are providing the task definition under the **Family** heading in **Deployment configuration**

The screenshot shows the 'Deployment configuration' form in the Amazon ECS console. The 'Application type' is set to 'Task'. The 'Task definition' section has a red box around the 'Family' dropdown menu, which is set to 'scrappy-flask-task3'. The 'Revision' dropdown is set to '2 (LATEST)'. The 'Desired tasks' field is set to '1'. The 'Task group' field is empty.

and under the **Networking** tab, making the task and the relevant port available from anywhere. Also make sure **Public IP** is turned on.

**Networking**

**VPC** Info  
Select a VPC to use for your Amazon ECS resources.  
vpc-0d9222f5da44f373b  
default Create a new VPC

**Subnets**  
Choose the subnets within the VPC that the task scheduler should consider for placement.  
Choose subnets Clear current selection  
subnet-0d23e55c2d9e99da8 x subnet-07e4ba326400e8736 x subnet-0577260445af0c0d8 x  
ap-southeast-2c 172.31.16.0/20 ap-southeast-2b 172.31.32.0/20 ap-southeast-2a 172.31.0.0/20

**Security group** Info  
Choose an existing security group or create a new security group.  
☐ Use an existing security group  
☒ Create a new security group **Select this**

**Security group details**  
Specify the configuration to use when creating the new security group.  
**Security group name**  
ecs-ls666ts2  
**Security group description**  
Created in ECS Console  
Security group description must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, underscores (\_), hyphens (-), colons (:), forward slashes (/), parentheses (()), hash-tags (#), commas (,), at signs (@), brackets ([]), plus signs (+), equal signs (=), ampersands (&), semicolons (;), brackets ({}), exclamation points (!), dollar signs (\$), asterisks (\*).

**Inbound rules for security groups**  
Add one or more rules for your security group.  
**Type** Custom TCP **Protocol** TCP **Port range** 80-6000 **Source** Anywhere **Values** 0.0.0.0/0, ::/0 Delete  
Enter a valid port or port range between 0 and 65535. For example: 80 or 0-1023.

**Public IP** Info  
Choose whether to auto-assign a public IP to the task's elastic network interface (ENI).  
☒ Turned on

This is all default

Add this security rule

seng3011-scrappy-flask Last updated March 02, 2025 at 09:17 (UTC+11:00) Update cluster Delete cluster

**Cluster overview**

ARN: [arn:aws:ecs:ap-southeast-2:339712883212:cluster/seng3011-scr](#) **Status** Active

**CloudWatch monitoring** Default

**Registered container instances**

**Services**

**Draining** Active

**Tasks** Pending

**Running** 1

Services **Tasks** Infrastructure Metrics Scheduled tasks Configuration Tags

**Tasks (1)**

Filter tasks by property or value Filter desired status Filter launch type  
Running Any launch type

☐ Task ☐ Last status

☒ f5b8edbd... Running

Manage tags Stop Run new task

Desired status Task definition... Health status Start...

Running scrappy-flask-t... Unknown

If you click on the cluster, and click on the new task (f5b8edbd... for me) you should be able to see a public IP address.

f5b8edbd4f74f4fa131a218fdc7bcc2 Last updated March 02, 2025 at 09:17 (UTC+11:00) Stop

**Configuration** Logs Networking Volumes (0) Tags

**Task overview**

ARN: [arn:aws:ecs:ap-southeast-2:339712883212:task/seng3011-scrappy-flask/f5b8edbd4f74f4fa131a218fdc7bcc2](#) **Last status** Running **Desired status** Running **Started/Created at** March 02, 2025 at 00:35 (UTC+11:00) March 02, 2025 at 00:35 (UTC+11:00)

**Fargate ephemeral storage**

**Encryption** Info Default: AWS Fargate encryption **Size (GiB)** 20

**Configuration**

**Operating system/Architecture** Linux/X86\_64

**CPU | Memory** 1 vCPU | 3 GB

**Platform version** T.4.0

**Fault injection** Turned off

**ECS Exec** Info Turned off

**Capacity provider** FARGATE

**Launch type** FARGATE

**Container instance ID** -

**Task definition: revision** scrappy-flask-task3.2

**Task group** family/scrappy-flask-task3

**ENI ID** [eni-0c92700208425329e](#)

**Network mode** awsvpc

**Subnet ID** [subnet-0d23e55c2d9e99da8](#)

**Public IP** 13.236.7.118 open address (a)

**Private IP** 172.31.23.246

**MAC address** 08:09:ae:f9:52:e7

**Container details for scrappy-flask-container**

**Details** Log configuration Restart policy Network bindings Docker labels and hosts Environment variables and files Volume configuration

**Details**

**Image URI** [339712883212.dkr.ecr.ap-southeast-2.amazonaws.com/seng3011/scrappyapp/latest](#) **Essential** Yes **Command** -

Public IP address of the task running on the cluster

If you take this IP address, add a : and the port number (for me 5001), you should have the address to access Flask app publicly. So for me, this address would look like <http://13.236.7.118:5001/>.



# Cost Considerations [↗](#)

Depending on what kind of CPU and Memory you selected for creating your cluster, your costs will scale accordingly.

## Pricing Details

Pricing is based on requested vCPU, memory, Operating Systems, CPU Architecture<sup>1</sup>, and storage resources for the Task or Pod. The five dimensions are independently configurable.

<sup>1</sup> Windows Operating System and ARM CPU Architecture are currently only available for Amazon ECS.

Linux/X86Linux /ARMWindows/X86

Region:

Asia Pacific (Sydney)

|                   | Price     |
|-------------------|-----------|
| per vCPU per hour | \$0.04856 |
| per GB per hour   | \$0.00532 |

Fargate costs in Sydney

NOTE that using Linux/Arm is cheaper than Linux/X86.

ECR seems essentially free for a year because we would get 500 MB / month for the first year

## AWS Free Tier \*

As a new Amazon ECR customer, you get 500 MB per month of storage for your **private** repositories for one year as part of the [AWS Free Tier](#).

Both new and existing customers get 50 GB per month of always-free storage for their public repositories. You can anonymously (without using an AWS account) transfer 500 GB of data to the Internet from a public repository each month for free. If you sign up for an AWS account, or authenticate to Amazon ECR with an existing AWS account, you can transfer 5 TB of data to the Internet from a public repository each month for free. You also get unlimited bandwidth at no cost when transferring data from a public repository to AWS compute resources in any AWS Region.

Your free usage is calculated each month across all regions and automatically applied to your bill; free usage does not accumulate.