

Name: Jagadeesh  
Id: 2000032181

## Cloud devops end sem project:

Deploying ML models using flask , Docker and Kubernetes

### INTRODUCTION:

To Build a ML Web application, Dockerized and Deploy on Kubernetes

### FLOW OF THE PROJECT DEPLOYMENT:



## STEPS:

- 1 Build your Flask app
- 2 Dockerize the Flask app by creating a Docker file.
- 3 Build a Docker image of your Flask app.
- 4 Push the Docker image to a Docker registry (e.g., Docker Hub)
- 5 Create a Kubernetes cluster on AWS
- 6 Deploy a Kubernetes deployment object to run your Docker image as a container in the cluster
- 7 Create a Kubernetes service object to expose your Flask app to the internet

## Tools used :

Github

Docker

Flask

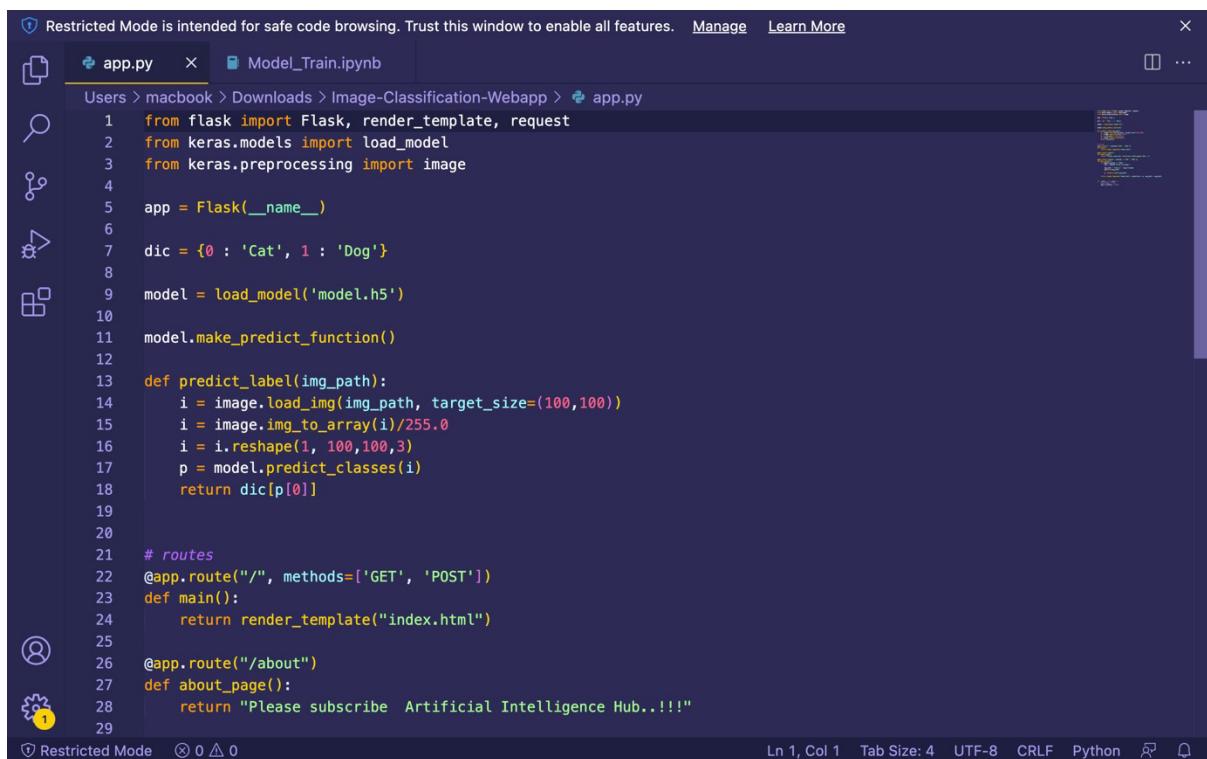
Ecr

Iam

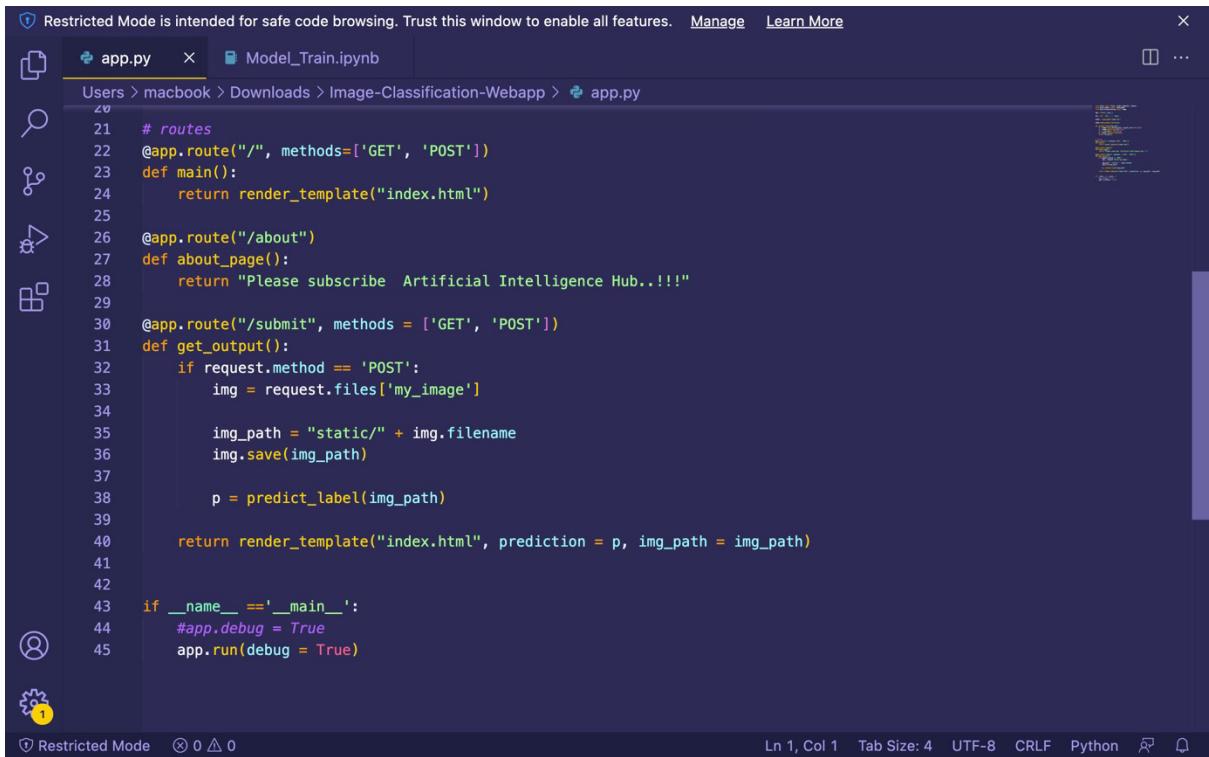
kubernetes

# implementation:

-> implementing the machine learning code into the flask

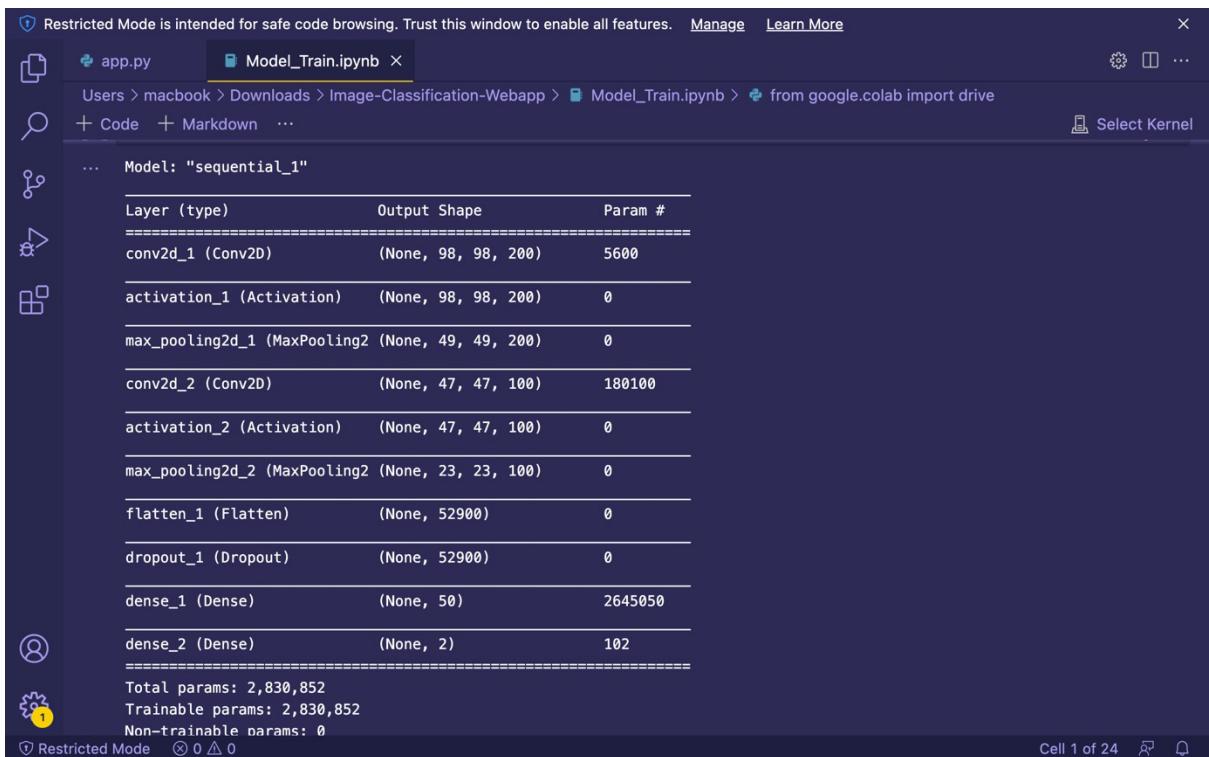


```
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
app.py × Model_Train.ipynb
Users > macbook > Downloads > Image-Classification-Webapp > app.py
1  from flask import Flask, render_template, request
2  from keras.models import load_model
3  from keras.preprocessing import image
4
5  app = Flask(__name__)
6
7  dic = {0 : 'Cat', 1 : 'Dog'}
8
9  model = load_model('model.h5')
10
11 model.make_predict_function()
12
13 def predict_label(img_path):
14     i = image.load_img(img_path, target_size=(100,100))
15     i = image.img_to_array(i)/255.0
16     i = i.reshape(1, 100,100,3)
17     p = model.predict_classes(i)
18     return dic[p[0]]
19
20
21 # routes
22 @app.route("/", methods=['GET', 'POST'])
23 def main():
24     return render_template("index.html")
25
26 @app.route("/about")
27 def about_page():
28     return "Please subscribe Artificial Intelligence Hub...!!!"
Ln 1, Col 1 Tab Size: 4 UTF-8 CRLF Python ⌂ ⓘ
```



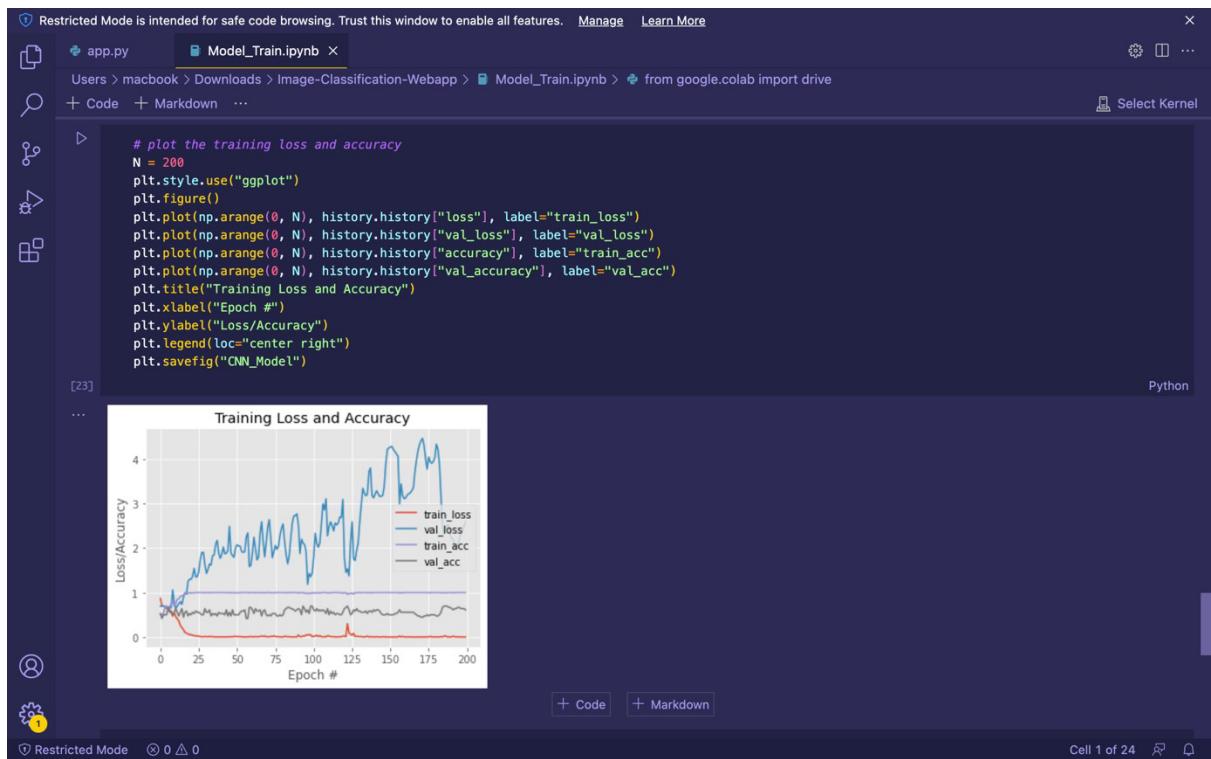
```
21 # routes
22 @app.route("/", methods=['GET', 'POST'])
23 def main():
24     return render_template("index.html")
25
26 @app.route("/about")
27 def about_page():
28     return "Please subscribe Artificial Intelligence Hub..!!!"
29
30 @app.route("/submit", methods = ['GET', 'POST'])
31 def get_output():
32     if request.method == 'POST':
33         img = request.files['my_image']
34
35         img_path = "static/" + img.filename
36         img.save(img_path)
37
38         p = predict_label(img_path)
39
40     return render_template("index.html", prediction = p, img_path = img_path)
41
42
43 if __name__ == '__main__':
44     #app.debug = True
45     app.run(debug = True)
```

## ->summary

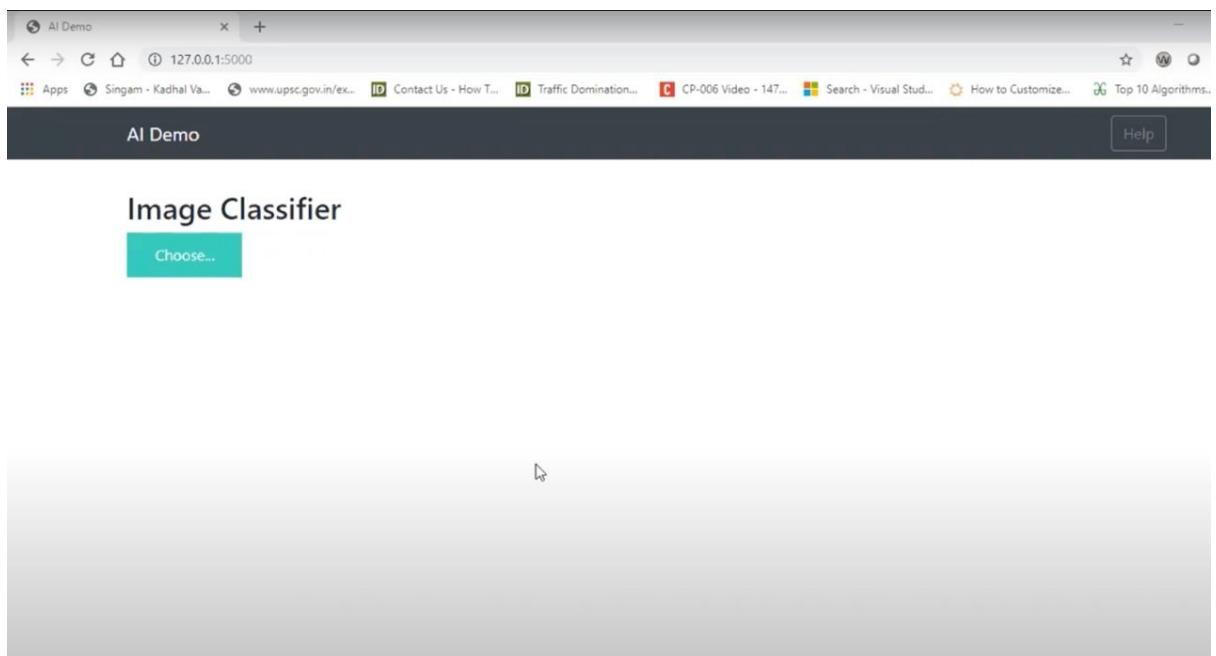


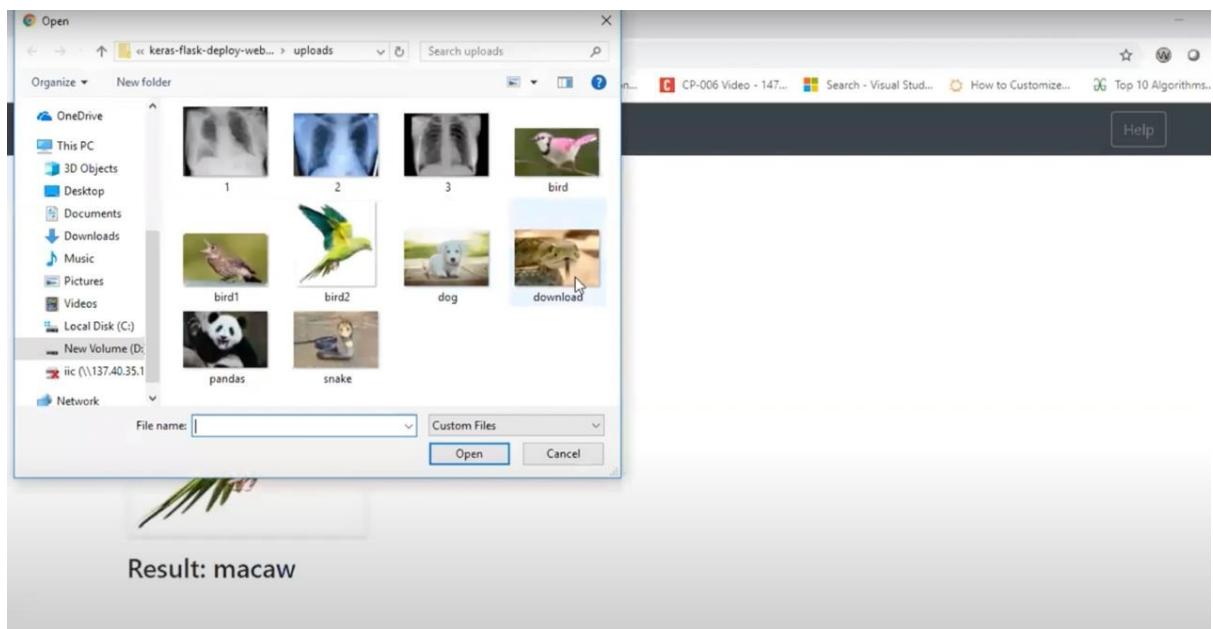
```
... Model: "sequential_1"
Layer (type)          Output Shape       Param #
=====
conv2d_1 (Conv2D)     (None, 98, 98, 200)   5600
activation_1 (Activation) (None, 98, 98, 200)   0
max_pooling2d_1 (MaxPooling2D) (None, 49, 49, 200)   0
conv2d_2 (Conv2D)     (None, 47, 47, 100)    180100
activation_2 (Activation) (None, 47, 47, 100)   0
max_pooling2d_2 (MaxPooling2D) (None, 23, 23, 100)   0
flatten_1 (Flatten)   (None, 52900)        0
dropout_1 (Dropout)   (None, 52900)        0
dense_1 (Dense)      (None, 50)           2645050
dense_2 (Dense)      (None, 2)            102
=====
Total params: 2,830,852
Trainable params: 2,830,852
Non-trainable params: 0
```

## ->Performance:



->Flask App Running on the Local Server:





->Creating the Docker File:

```
from alpine:latest
RUN apk add --no-cache python3-dev \
    && pip3 install --upgrade pip
WORKDIR /app
COPY . /app
RUN pip3 --no-cache-dir install -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

## ->Container Building:

```
vagrant@localhost:~/selftuts/python-flask-docker-container(master) » vim Dockerfile
vagrant@localhost:~/selftuts/python-flask-docker-container(master) » python app.py
vagrant@localhost:~/selftuts/python-flask-docker-container(master) » curl localhost:5000
vagrant@localhost:~/selftuts/python-flask-docker-container(master) »
```

WARNING:werkzeug: \* Debugger is active!  
INFO:werkzeug: \* Debugger PIN: 156-489-742  
DEBUG:root:Inside the post method of Task  
INFO:werkzeug:10.0.2.2 - - [24/Nov/2018 11:39:43] "GET / HTTP/1.1" 200 -  
^C  
(venv) -----  
~/selftuts/python-flask-docker-container(master) » vim  
(venv) -----  
~/selftuts/python-flask-docker-container(master) » python app.py  
vagrant@localhost:~/selftuts/python-flask-docker-container(master) » curl localhost:5000  
vagrant@localhost:~/selftuts/python-flask-docker-container(master) »  
DEBBUG:root:Starting Flask Server  
\* Serving Flask app "app" (lazy loading)  
\* Environment: production  
WARNING: Do not use the development server in a production environment.  
Use a production WSGI server instead.  
\* Debug mode: on  
INFO:werkzeug: \* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)  
INFO:werkzeug: \* Restarting with stat  
DEBUG:root:Starting Flask Server  
WARNING:werkzeug: \* Debugger is active!  
INFO:werkzeug: \* Debugger PIN: 156-489-742  
DEBUG:root:Inside the post method of Task  
INFO:werkzeug:10.0.2.2 - - [24/Nov/2018 11:40:18] "GET / HTTP/1.1" 200 -  
DEBUG:root:Inside the post method of Task  
INFO:werkzeug:10.0.2.2 - - [24/Nov/2018 11:40:19] "GET / HTTP/1.1" 200 -  
DEBUG:root:Inside the post method of Task  
INFO:werkzeug:10.0.2.2 - - [24/Nov/2018 11:40:20] "GET / HTTP/1.1" 200 -  
^C  
(venv) -----  
~/selftuts/python-flask-docker-container(master) » ls  
api app.py \_\_pycache\_\_ Readme.md requirements.txt venv  
(venv) -----  
~/selftuts/python-flask-docker-container(master) » vim Dockerfile  
(venv) -----  
~/selftuts/python-flask-docker-container(master) » docker build -t flask\_docker\_container .

## ->Container Running:

```
vagrant@localhost ~ % ~/selftuts/python-flask-docker-container(master*) » docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
fa4bc775a1f3        flaskapp           "/bin/sh"          14 seconds ago   Up 13 seconds
snyder
vagrant@localhost ~ % ~/selftuts/python-flask-docker-container(master*) »
```

-> Creating an Elastic container registry repository

The screenshot shows the AWS ECR (Amazon Container Registry) console. The URL in the address bar is `ap-south-1.console.aws.amazon.com/ecr/repositories?region=ap-south-1`. The main content area displays a green success message: "Successfully created repository identicalcloud". Below this, under the heading "Private repositories (1)", there is a table listing the repository "identicalcloud". The table columns include Repository name, URI, Created at, Tag immutability, Scan on push, and Encryption type. The "Created at" column shows the date and time as Feb 25, 2021 11:09:15 PM. The "Tag immutability" column shows "Disabled". The "Scan on push" and "Encryption type" columns also show "Disabled" and "AES-256" respectively.

Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type
identicalcloud	[REDACTED]	Feb 25, 2021 11:09:15 PM	Disabled	Disabled	AES-256

Creating iam role for pushing to repository:

The screenshot shows the AWS IAM Management Console with the URL [console.aws.amazon.com/iam/home?region=ap-south-1#users\\$new?step=permissions&accessKey&userNames=ecr-identicalcloud&permissionType=...](https://console.aws.amazon.com/iam/home?region=ap-south-1#users$new?step=permissions&accessKey&userNames=ecr-identicalcloud&permissionType=...). The page is titled 'Add user' and is at step 2: Set permissions. There are three options: 'Add user to group', 'Copy permissions from existing user', and 'Attach existing policies directly', with the last one being selected. A search bar filters the results by 'regist'. The table below shows 17 results:

	Policy name	Type	Used as
<input type="checkbox"/>	AmazonEC2ContainerRegistryFullAccess	AWS managed	None
<input type="checkbox"/>	AmazonEC2ContainerRegistryPowerUser	AWS managed	None
<input type="checkbox"/>	AmazonEC2ContainerRegistryReadOnly	AWS managed	None
<input type="checkbox"/>	AmazonElasticContainerRegistryPublicFullAccess	AWS managed	None
<input type="checkbox"/>	AmazonElasticContainerRegistryPublicPowerUser	AWS managed	None
<input type="checkbox"/>	AmazonElasticContainerRegistryPublicReadOnly	AWS managed	None
<input type="checkbox"/>	AmazonRoute53AutoNamingRegistrantAccess	AWS managed	None
<input type="checkbox"/>	AWSCloudMapRegisterInstanceAccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None
<input type="checkbox"/>	awscloudmapregisterinstanceaccess	AWS managed	None

Buttons at the bottom include 'Cancel', 'Previous', 'Next: Tags', and a 'Create policy' button.

## Push commands for identicalcloud

The screenshot shows the AWS ECR console with the URL [ap-south-1.console.aws.amazon.com/ecr/repositories?region=ap-south-1](https://ap-south-1.console.aws.amazon.com/ecr/repositories?region=ap-south-1). The left sidebar shows 'Amazon Container Services' and 'Amazon ECR'. A modal window titled 'Push commands for identicalcloud' is open. It has tabs for 'macOS / Linux' (selected) and 'Windows'. The content area contains instructions:

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 ap-south-1 | docker login --username AWS --password-stdin
```

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 is already built:  

```
docker build -t identicalcloud .
```

Buttons at the bottom include 'Close'.

## Pushing into the ECR repository:

```
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@ip-172-31-9-60:~# docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
identicalcloud/nodejs  2        6f7124bld12c  7 days ago   86MB
ic-nodejs           2        6f7124bld12c  7 days ago   86MB
identicalcloud/nodejs  1        50a63ebdlc63  7 days ago   86MB
node                10-alpine  cd8095d6b851  12 days ago  0B 7MB
root@ip-172-31-9-60:~#
```

```
2344f49dfbe3: Pushed
d179931639eb: Pushed
37cc999115d4: Pushed
9d8759356f0a: Pushed
4992425f60b2: Pushed
729a2badad91: Pushing [=====>] 8.371MB
ec7c69516687: Pushing [=====>] 18.35MB/68.78MB
0fcbbbeeeb0d7: Pushing [=====>] 5.396MB/5.615MB
```

## After pushing into the ECR repository:

The screenshot shows the AWS ECR service page. On the left, there's a sidebar with navigation links for Amazon Container Services (Amazon ECS, Clusters, Task definitions), Amazon EKS (Clusters), and Amazon ECR (Repositories, Images, Permissions, Lifecycle Policy, Tags, Registries, Public gallery). The 'Images' link under ECR is highlighted. The main content area shows the 'identicalcloud' repository. It has a heading 'identicalcloud' with 'View push commands' and 'Edit' buttons. Below that is a table titled 'Images (1)'. The table has columns: Image tag, Pushed at, Size (MB), Image URI, Digest, Scan status, and Vulnerabilities. One row is listed: an image tag with a small icon, pushed on Feb 25, 2021, at 11:18:18 PM, size 28.72 MB, image URI, digest sha256:6268f914498bed..., scan status -, and vulnerability status -.