MINDSONAI

PACKAGING

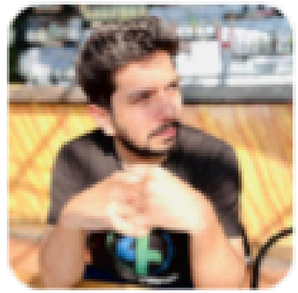
ML MODELS

linkedin.com/rishabhio

Go to the following repository:

rishabhio/**packaging-ml-model**

Learn how to package a machine learning model into a container



1 Contributor

0 Issues

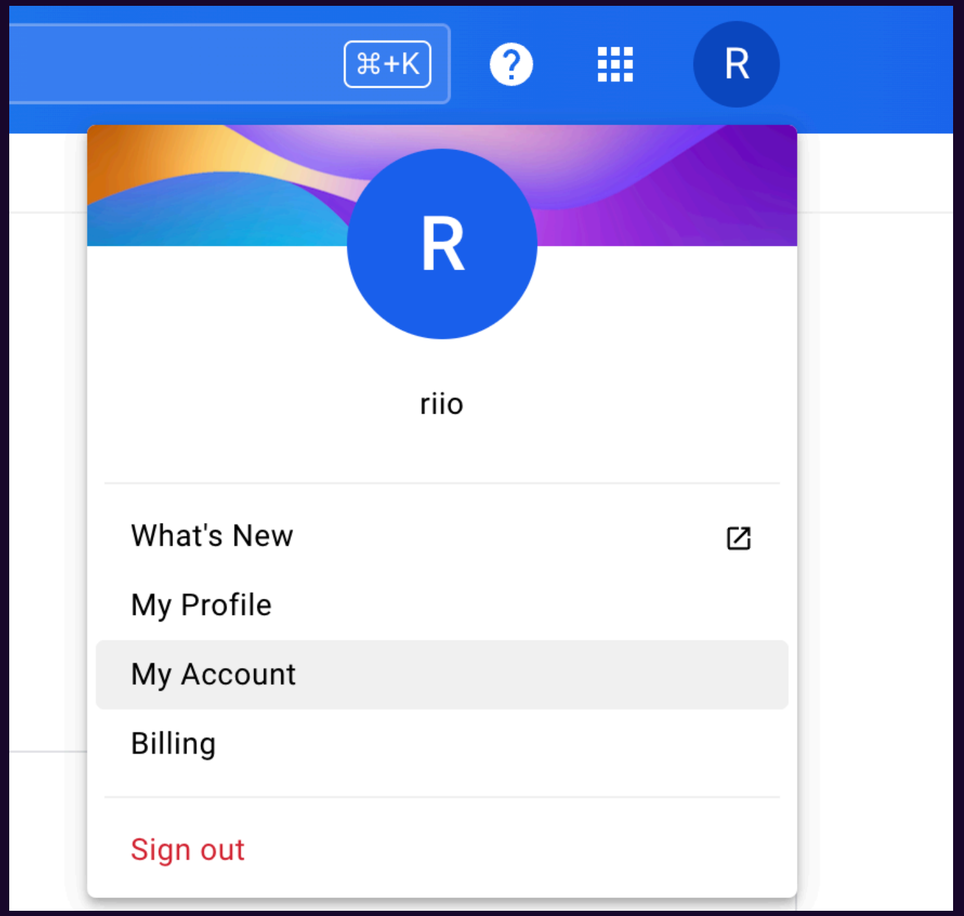
0 Stars

0 Forks

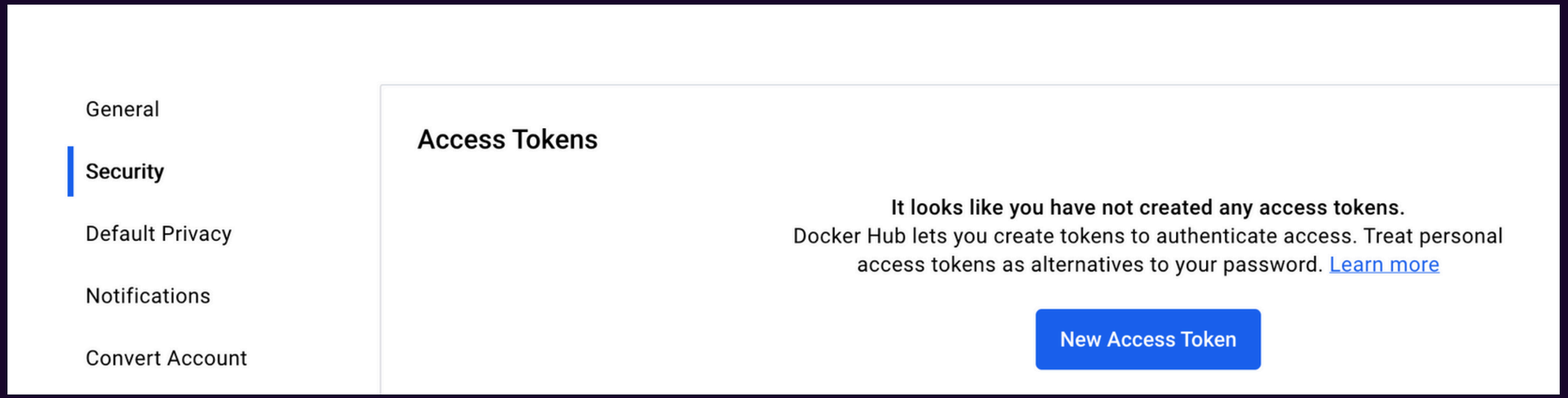
rishabhio/packaging-ml-model: Learn how to package a machine learning model into a container

Learn how to package a machine learning model into a container - rishabhio/packaging-ml-model

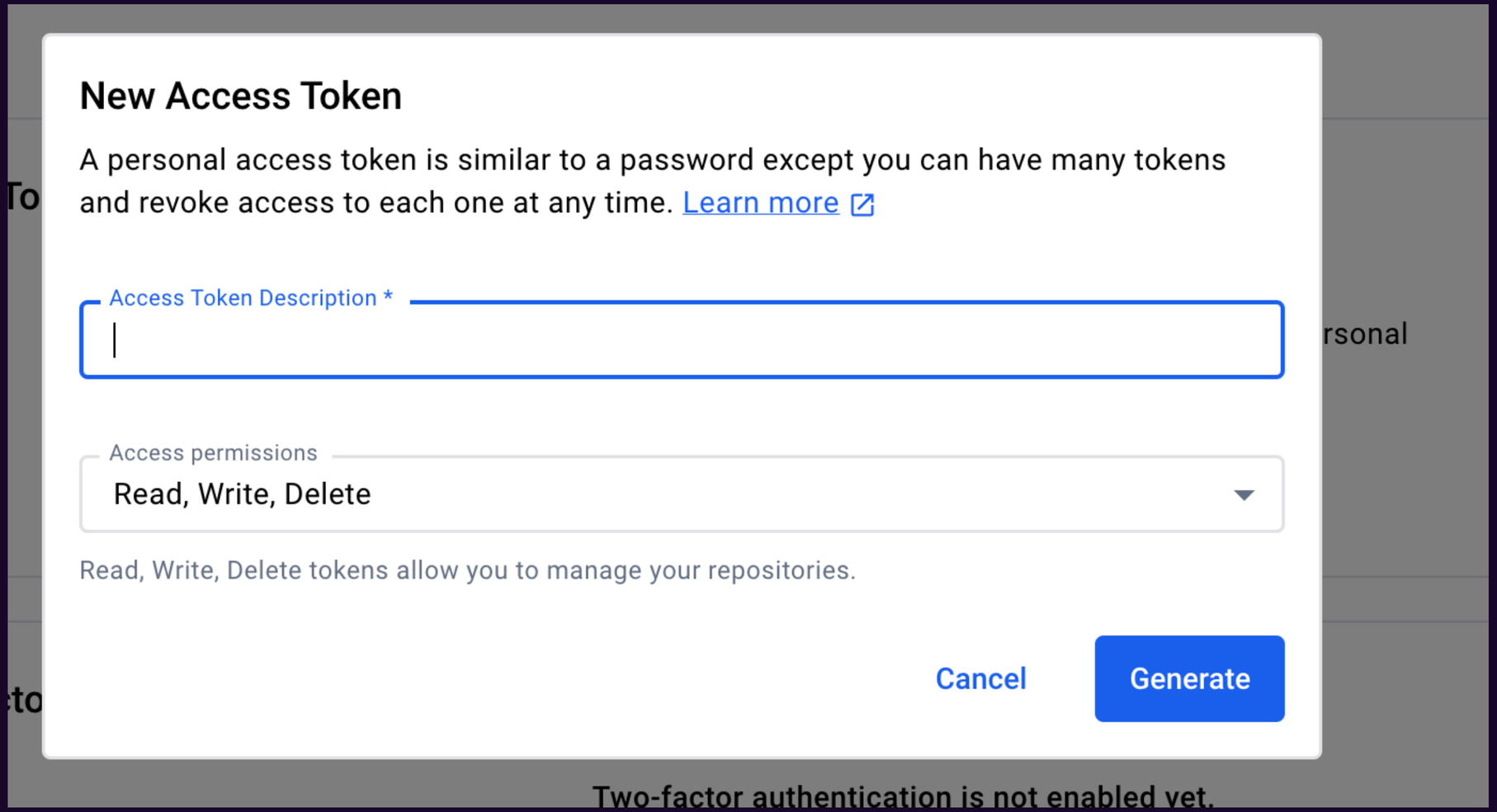
DockerHub Account PAT



Create Token

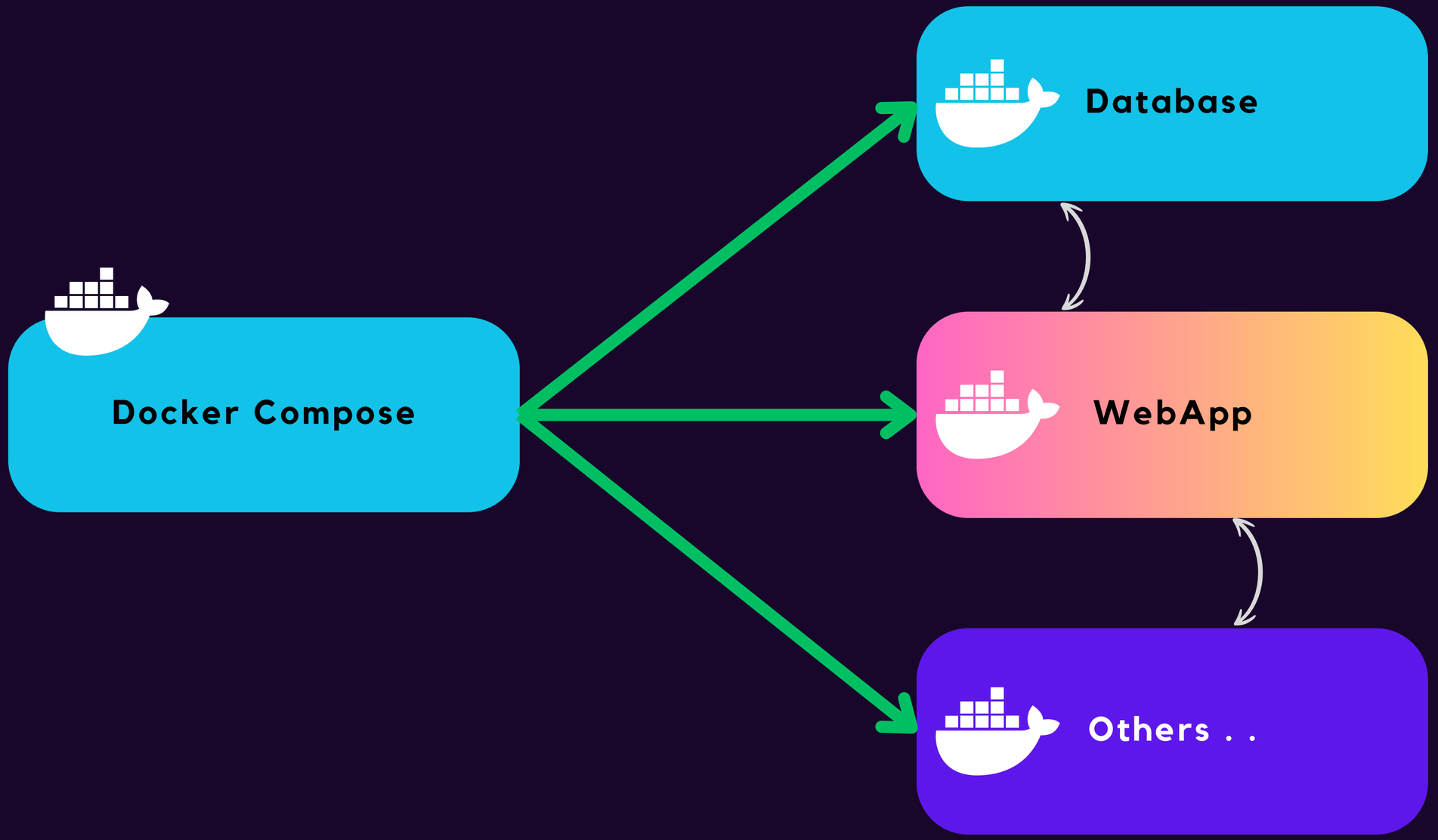


Provide a Name



Managing Multi Container Apps

Intro to Docker Compose


Database
Docker Compose
WebApp
Others . .

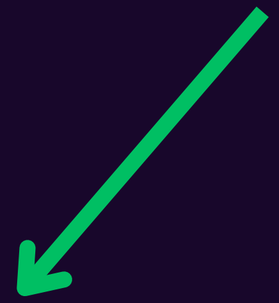
Docker Compose is a tool for defining and running multi-container applications

```
docker-compose --help
```

```
docker-compose --version
```

What all does compose help with?

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service



Service is nothing but a fancier name for a container based on the goal.

Simplified control

Docker Compose allows you to define and manage multi-container applications in a single YAML file

Efficient collaboration

Docker Compose configuration files are easy to share, facilitating collaboration among developers, operations teams, and other stakeholders

Rapid application development

Compose caches the configuration used to create a container. When you restart a service that has not changed, Compose re-uses the existing containers

Portability across Environments

Compose supports variables in the Compose file. You can use these variables to customize your composition for different environments

Setting up Your first multi-container app

Step-1

Clone the github repo to your chosen working directory.

Step-2

Create a file called app.py

Step-3

app.py

```
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)
```

Step-3

app.py

```
def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)
```

Step-3

app.py

```
@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

Step-2

Create another file called requirements.txt

Step-2

Create another file called requirements.txt

Step-3

requirements.txt

```
flask
redis
```

Step-2

Now create a file called
Dockerfile

Step-3

app.py

```
# syntax=docker/dockerfile:1
FROM python:3.10-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run", "--debug"]
```

Compose.yaml

Create a file called
compose.yaml

Define Services

compose.yaml

```
services:
  web:
    build: .
    ports:
      - "8000:5000"
  redis:
    image: "redis:alpine"
```

Compose defines services

compose.yaml

This Compose file defines two services:
web and redis.

Build and Run

compose.yaml

Build and Run your App

docker compose up

Build and Run

compose.yaml

Other important commands

docker compose down

docker compose logs

docker compose exec

Task-01 (Carries weightage towards final evaluation)

Use Docker to Build the following app

Use of Google / ChatGPT is allowed

Use docker compose to build an app which supports the following:

1. Fast api (a simple api to trigger tasks)
2. Apache airflow (to orchestrate tasks)
3. Define a dag (a way to define a pipeline of tasks)
4. Execute the dag

Task-02 (Carries weightage towards final evaluation)

Use Docker to Build the following app

Use of Google / ChatGPT is allowed

Use docker to build an app which supports the following:

1. Fast api (a simple api to draw inference)
2. Hugging face model (choose any text generation model)
3. Define a docker file (which packages both api and model)
4. Push the docker image to Dockerhub or GHRC.io or Azure Registry
5. Deploy the Docker to Azure Containers WebApp

Task-03 (Exploration Task counts towards finals)

1. Work in Teams of 2-4
2. Choose any feature from zenML
3. Work on that feature (research + implementation)
4. Present your work to class on Monday