MongoDB driven Flask application

SUBMITTED BY
MANASWINI MURALIDARAN (D17B-47)
MEGHNA MOHAN (D17B-51)

ABSTRACT:
The project involves creating Flask application using MongoDB using pyMongo API. The application will be wrapped up in docker containers. A tool called docker compose is used to build the complete application including running the flask app in the container, the Mongo database in a container. The two parts are linked together so that they can communicate all in one command which is docker-compose up.

INTRODUCTION:

MongoDB
MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. MongoDB is developed by MongoDB Inc. and is free and open-source, published under a combination of the GNU Affero General Public License and the Apache License.
PROS:

1. MongoDB is a document database in which one collection holds different documents.
2. No complex joins.
3. MongoDB is easy to scale.

CONS:

1. Since there are no joins, it has less flexibility with quering.
2. Data size in MongoDB is typically higher.

Flask

Flask is a micro web framework written in Python. Flask is called a micro framework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. Flask supports extensions that can add application features as if they were implemented in Flask itself.

PROS:

1. Convenient to use
2. Easy integration with html-css fles for backend processing.
3. Contains a built-in web development server.

DOCKER:

Docker is a great tool that allows you wrap up your app and everything it needs to run: code, runtime, and even system libraries and guarantee that it will always run the same, regardless of the environment (local machine, server, or even the cloud).The Docker container allows the applications to interact with the kernel or host OS. Your applications and the libraries are packaged together in a common shared format, built by the developer and deployed to whatever cloud you are running on. Whether you're deploying a web app, performing data analysis, or creating local environments for your dev team or CI builds, Docker can help.

PROS:

1. Improves the performance.
2. The architecture is lean, lightweight, portable, efficient and isolated

IMPLEMENTATION DETAILS

Dockerfile

```
FROM python:2.7
ADD . /todo
WORKDIR /todo
RUN pip install -r requirements.txt
```

app.py

```
import os

from flask import Flask, redirect, url_for, request, render_template
from pymongo import MongoClient

app = Flask(__name__)

client = MongoClient(
   os.environ['TODO_DB_1_PORT_27017_TCP_ADDR'],
   27017)
db = client.tododb
```

```
@app.route('/')
def todo():

    _items = db.tododb.find()
    items = [item for item in _items]

    return render_template('todo.html', items=items)


@app.route('/new', methods=['POST'])
def new():

    item_doc = {
        'name': request.form['name'],
        'description': request.form['description']
    }
    db.tododb.insert_one(item_doc)

    return redirect(url_for('todo'))

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True)
```

docker-compose.yml

```
web:
  build: .
  command: python -u app.py
  ports:
    - "5000:5000"
  volumes:
    - .:/todo
  links:
    - db
db:
  image: mongo:3.0.2
```

todo.html

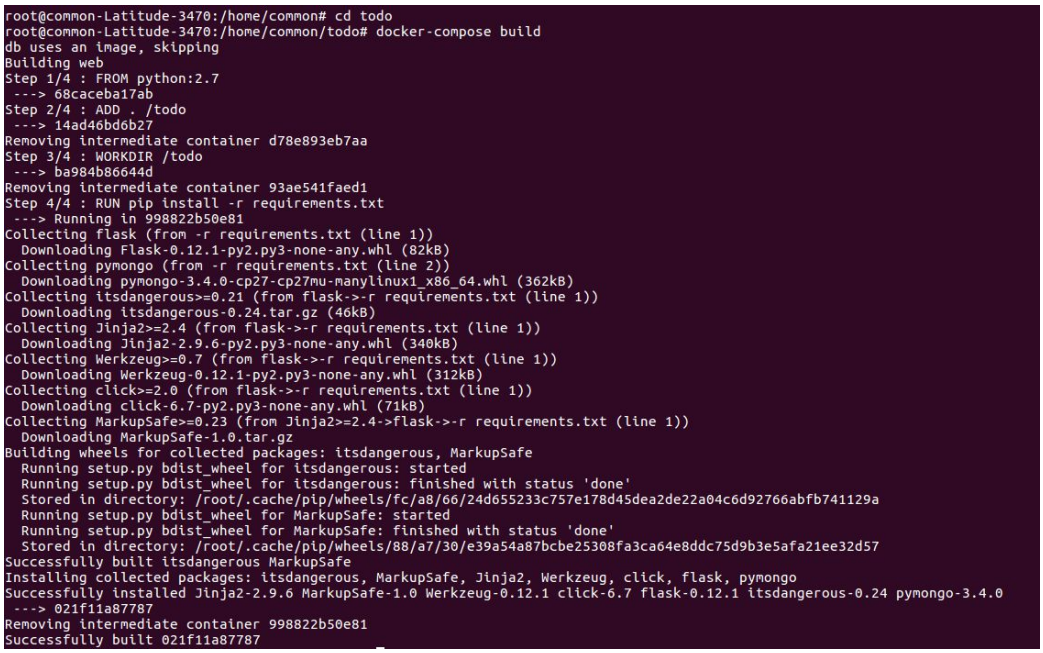
```
<!doctype html>

<form action="/new" method="POST">
  <input type="text" name="name"></input>
  <input type="text" name="description"></input>
  <input type="submit"></input>
</form>
```

```
{% for item in items %}
  <h1> {{ item.name }} </h1>
  <p> {{ item.description }} <p>
{% endfor %}
```

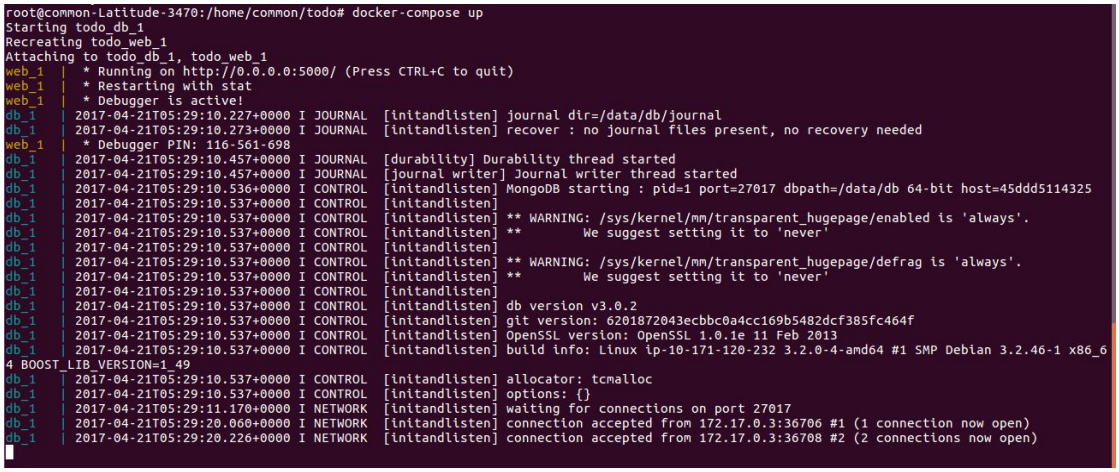
RESULTS AND SNAPSHOTS

Docker-compose build

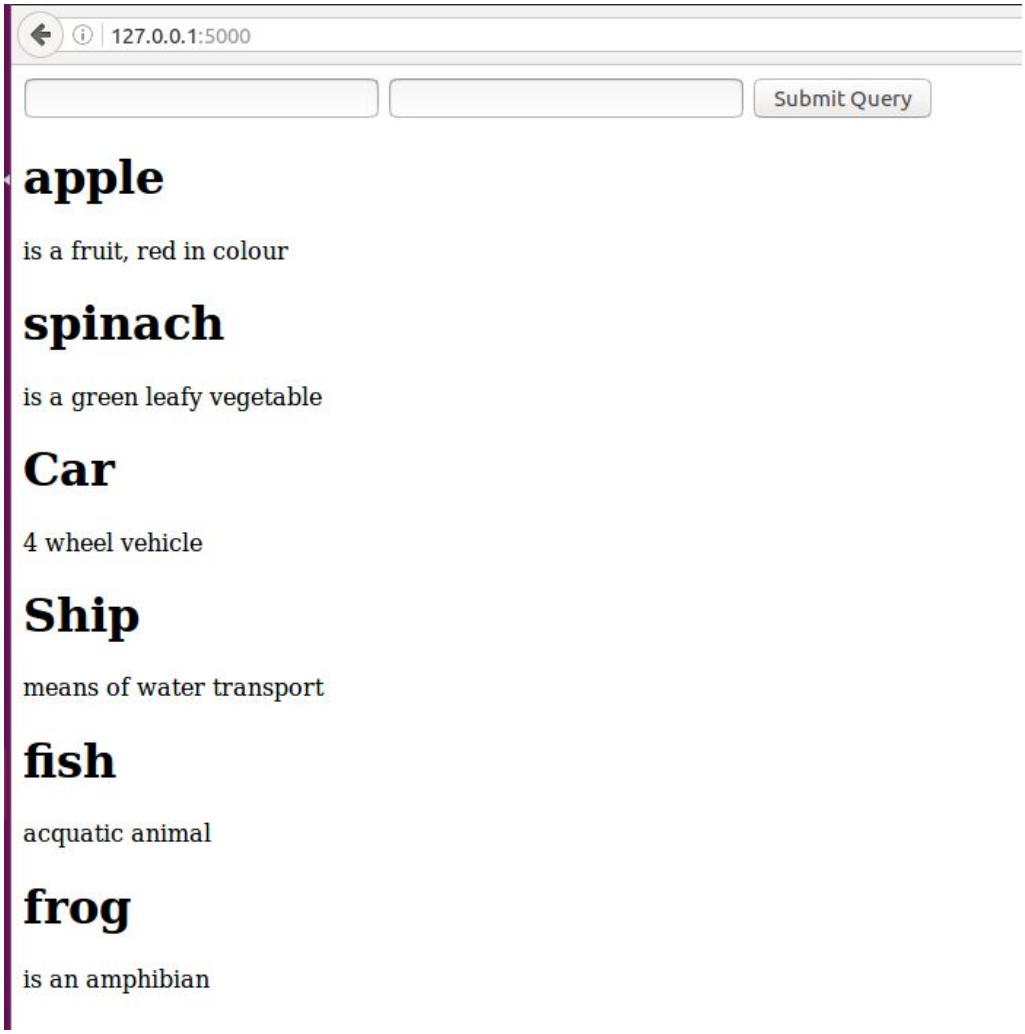


```
root@common-Latitude-3470:/home/common# cd todo
root@common-Latitude-3470:/home/common/todo# docker-compose build
db uses an image, skipping
Building web
Step 1/4 : FROM python:2.7
 ---> 68caceba17ab
Step 2/4 : ADD . /todo
 ---> 14ad46bd6b27
Removing intermediate container d78e893eb7aa
Step 3/4 : WORKDIR /todo
 ---> ba984b86644d
Removing intermediate container 93ae541faed1
Step 4/4 : RUN pip install -r requirements.txt
 ---> Running in 998822b50e81
Collecting flask (from -r requirements.txt (line 1))
  Downloading Flask-0.12.1-py2.py3-none-any.whl (82kB)
Collecting pymongo (from -r requirements.txt (line 2))
  Downloading pymongo-3.4.0-cp27-cp27mu-manylinux1_x86_64.whl (362kB)
Collecting itsdangerous>=0.21 (from flask->-r requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.4 (from flask->-r requirements.txt (line 1))
  Downloading Jinja2-2.9.6-py2.py3-none-any.whl (340kB)
Collecting Werkzeug>=0.7 (from flask->-r requirements.txt (line 1))
  Downloading Werkzeug-0.12.1-py2.py3-none-any.whl (312kB)
Collecting click>=2.0 (from flask->-r requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->flask->-r requirements.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous: started
  Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/fc/a8/66/24d655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe: started
  Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/88/a7/30/e39a54a87bcbe25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, flask, pymongo
Successfully installed Jinja2-2.9.6 MarkupSafe-1.0 Werkzeug-0.12.1 click-6.7 flask-0.12.1 itsdangerous-0.24 pymongo-3.4.0
 ---> 021f11a87787
Removing intermediate container 998822b50e81
Successfully built 021f11a87787
```

Docker-compose up



```
root@common-Latitude-3470:/home/common/todo# docker-compose up
Starting todo_db_1
Recreating todo_web_1
Attaching to todo_db_1, todo_web_1
web_1  |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1  |  * Restarting with stat
web_1  |  * Debugger is active!
db_1   | 2017-04-21T05:29:10.227+0000 I JOURNAL  [initandlisten] journal dir=/data/db/journal
db_1   | 2017-04-21T05:29:10.273+0000 I JOURNAL  [initandlisten] recover : no journal files present, no recovery needed
web_1  |  * Debugger PIN: 116-561-698
db_1   | 2017-04-21T05:29:10.457+0000 I JOURNAL  [durability] Durability thread started
db_1   | 2017-04-21T05:29:10.457+0000 I JOURNAL  [journal writer] Journal writer thread started
db_1   | 2017-04-21T05:29:10.536+0000 I CONTROL  [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=45ddd5114325
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten]
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten]
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten]
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] db version v3.0.2
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] git version: 6201872043ecbbc0a4cc169b5482dcf385fc464f
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1e 11 Feb 2013
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] build info: Linux ip-10-171-120-232 3.2.0-4-amd64 #1 SMP Debian 3.2.46-1 x86_6
4 BOOST_LIB_VERSION=1_49
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] allocator: tcmalloc
db_1   | 2017-04-21T05:29:10.537+0000 I CONTROL  [initandlisten] options: {}
db_1   | 2017-04-21T05:29:11.170+0000 I NETWORK  [initandlisten] waiting for connections on port 27017
db_1   | 2017-04-21T05:29:20.060+0000 I NETWORK  [initandlisten] connection accepted from 172.17.0.3:36706 #1 (1 connection now open)
db_1   | 2017-04-21T05:29:20.226+0000 I NETWORK  [initandlisten] connection accepted from 172.17.0.3:36708 #2 (2 connections now open)
```



output

CONCLUSION:

A Flask Application was developed using MongoDB. The entire application was wrapped in docker containers. Docker allows you to build and ship applications easily. It also eliminates dependencies. The way we deploy applications these days tends to focus a lot on virtual machines.So all you have to do in order to run your application on a new OS instead of having to provision an entirely new host OS or VMs when you ship the application all you have to do when you have to spin up more processes is you have to spin up more docker containers either on the same docker Host or another Docker Host. The advantage is that the cloud provider can have a number of docker hosts pre provisioned and no other configuration is required.

REFERENCES

https://github.com/kpurdon/docker-flask-todo
https://www.youtube.com/watch?v=6opltZu4ABw