Mayank Satnalika

ML Engineer

@mayanksatnalika

CloudSEK

Moving from Notebooks to Production

How do we use your magical model?

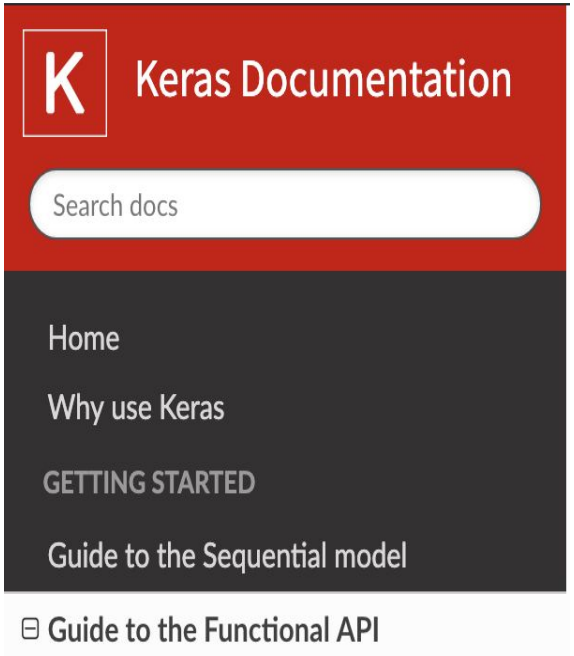
Some Introduction

Things to consider:

- Usability → Provide a standard API
- Dependencies Management & Easy Deployment → Docker
- Scalability, decoupling etc. → Queuing

API(s): Application Programming Interface

Connector between your prediction logic and the business logic.
Commonly used in context of HTTP web API
Can be other forms.
Keras provides a wrapper which acts as a API between you and the underlying tensorflow code.



APIs are how data is exchanged, content is published, media is consumed, and algorithms are applied across the web today. APIs are how you access your social data, your photos, your accounting information, and much, much more.

APIs are not a specific service or tool from a company, they are just like the web, but instead of getting HTML back with each request, you get JSON, XML, and CSV – providing structured, machine-readable information that can be used by other systems and within other applications with very little assistance from a human.

Ref: https://blog.getpostman.com/2019/09/24/intro-to-apis-what-is-an-api/

Let’s build an API

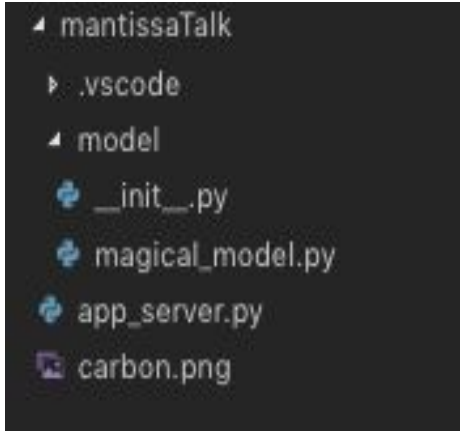
More specifically a *http* API
Anyone who wants to use our model can send a request to the *endpoint*. Input data to be predicted upon is sent in form of *post* request. You process the input request, and return the answer.

Some considerations

- In project structure, keep your model training code and your model serving code seperate.
- Need to validate all incoming requests, handle invalid requests properly.
- Handle multiple requests concurrently, PS: ML Logic is generally resource intensive.
- Handle long running requests.

```python
from config import *
import time
from flask import Flask, jsonify, make_response, request
from model import magic_predictoion_function

app = Flask(__name__)

def is_valid_body(request):
    try:
        body = request.json
        if "key1" not in body:
            return (False, "key1 not found in body.")
        if "key2" not in body:
            return (False, "key2 not found in body.")
          ...
          ...
    except Exception as e:
        return (False, e)
    return (True, "Ok")


@app.route('/')
def index():
    return "<h2>Hi! from my app</h2>"


@app.route('/classify', methods=['PUT'])
def classify():
    if not request.json:
        return make_response(jsonify(
            {"error": "Failed not json "}
        ), 400)
    body_is_valid = is_valid_body(request)
    if body_is_valid[0] is False:
        return make_response(jsonify(
            {"error": "Invalid body: {err}.".format(err=body_is_valid[1])}
        ), 422)
    ans = magic_prediction_function(request.json)
    return make_response(jsonify(
        {"ans": ans}
    ), 200)


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Flask A micro-web-framework



Issues:

- Flask is single threaded.
- Two incoming requests will be executed in order
- Second one needs to wait for first request to complete until it gets processed.
- Is only meant to be used by one person at a time, and is built this way

Ref: https://vsupalov.com/flask-web-server-in-production/

We need a more capable web-application server.

- Flask inbuilt server: A development server just made for personal testing.
  Use uwsgi + Nginx:
- WSGI stands for (Web Server Gateway Interface) → A standard Python specification for applications and servers to implement.
- uwsgi → uWSGI is a WSGI implementation package. Acts as a interface between flask and the actual web server nginx.
- uwsgi works by creating an instance of the python interpreter and importing the python files related to your application.
- This is roughly equivalent to just starting that number of python interpreter instances by hand, except that uWSGI will handle incoming HTTP requests and forward them to your application.
- This also means that each process has memory isolation—no state is shared, so each process gets its own *GIL.*
- **`The web client <-> the web server <-> the socket <-> uwsgi <-> Flask`**

Ref: https://www.reddit.com/r/Python/comments/4s40ge/understanding_uwsgi_threads_processes_and_gil/
Ref: https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uswgi-and-nginx-on-ubuntu-18-04

Queuing and Batching

- ML Systems processing time might be greater than HTTP request timeout time.
- Generally
  time (10*model.predict([something]))
  > time (model.predict([10*something]))

Data Flow

Endpoint: my-awesome-app.com/process: [POST request with payload]

Client → Server → Validate Payload → Generate a ID → Save ID to database → Push to Queue → Return client the ID

In Background: Process payload, update corresponding answers in the database .

Endpoint: my-awesome-app.com/process?id=1234:

Query database → Is processed ? → return

Magical-results

else return

Check again later

RabbitMQ

- Library for queuing. Written in erlang
- Api available in all popular languages
- Python: Pika

https://pika.readthedocs.io/en/stable/

```python
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)

message = ' '.join(sys.argv[1:]) or "Hello World!"
channel.basic_publish(
    exchange='',
    routing_key='task_queue',
    body=message,
    properties=pika.BasicProperties(
        delivery_mode=2,  # make message persistent
    ))
print(" [x] Sent %r" % message)
connection.close()
```

```python
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()
```

Add the processing function calls

Useful to know:

- RabbitMQ Management Plugin: Shows you all messages in queues
  Alternative:
- If possible: Use AWS SQS

GitHub Releases an ImageNe | mantisaa DS Talk - Google Si | wordpress - Docker Hub | localhost:5000 | wordpress - Docker Hub | Issues · docker-library/wordp | Amazon Simple Queue Servi

aws.amazon.com/sqs/

aws

Contact Sales | Support | English | My Account | Sign In to the Console

Products | Solutions | Pricing | Documentation | Learn | Partner Network | AWS Marketplace | Customer Enablement | Explore More

Amazon SQS | Overview | Features | Pricing | Getting Started | Resources | FAQs

Amazon Simple Queue Service

Fully managed message queues for microservices, distributed systems, and serverless applications

Get started for free

FEATURED

Decouples Counseling

Join us on October 1 at 12 PM PDT on Twitch for an intro to building microservices.

Save the date »

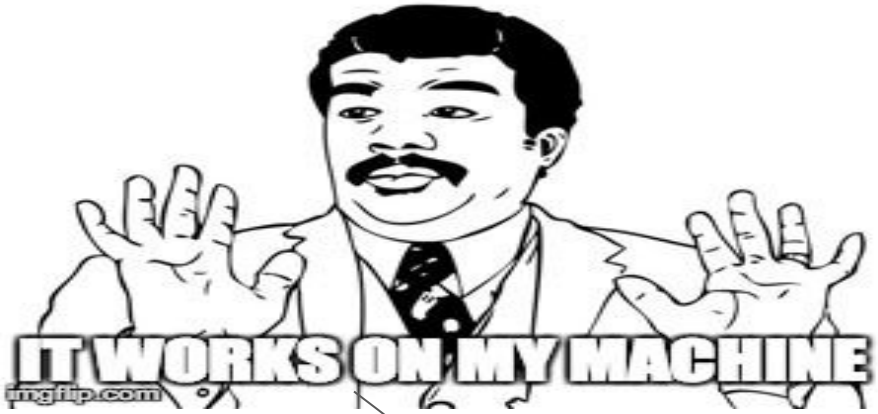
Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with managing and operating message oriented middleware, and empowers developers to focus on differentiating work. Using SQS, you can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available. Get started with SQS in minutes using the AWS console, Command Line Interface or SDK of your choice, and three simple commands.

SQS offers two types of message queues. Standard queues offer maximum throughput, best-effort ordering, and at-least-once delivery. SQS FIFO queues are designed to guarantee that messages are processed exactly once, in the exact order that they are sent.

https://www.twitch.tv/events/M49g5hvQRhy9N3SrXnTIew?sc_icampaign=pac_event-decouples-counseling-part2&sc_ichannel=ha&sc_icontent=awssm-3054&sc_iplace=ribbon&trk=ha_awssm-3054

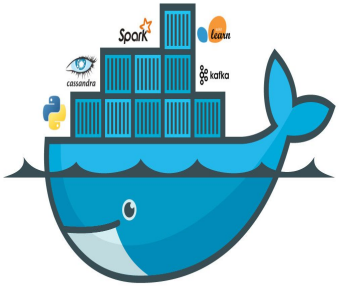
Dependencies/ Runtime management

```
(py36)  ✘  ~/Desktop/mantissaTalk/scripts  python run_model.py
Traceback (most recent call last):
  File "run_model.py", line 1, in <module>
    import keras
ModuleNotFoundError: No module named 'keras'
```



ONLY

Containers all the way



- Containerisation tool
- Everything is inside a container.
- 2 containers built from same image are exactly equal (almost :p)
- Instead of distributing just the code, you distribute the image
- Virtualenv (but much powerful)
- Code ⊂ Image.
- Image = Code + Dependencies + Running instructions
- A image which is running = a container
- Dockerfile: a yaml-based file with instructions to build your image.
- Dockerhub: GitHub for your docker images.

tensorflow / tensorflow

Used by 49,897 | Watch 8,582 | Star 134,517 | Fork 77,474

Code | Issues 2,461 | Pull requests 218 | Projects 1 | Security | Insights

Filters | forward compatibility | Labels 71 | Milestones 7 | New issue

Clear current search query, filters, and sorts

13 Open | 113 Closed | Author | Labels | Projects | Milestones | Assignee | Sort

[r1.15-CherryPick]:forward_compatible env variable caching perf optimization. ✕ 3
cla: yes size:S
#32632 opened 5 days ago by kkimdev • Approved

ValueError from invalid weights while loading older .h5 model TF 2.0 9
stat:awaiting response
#32398 opened 13 days ago by ghannum

[tf.data] Cleanup of tf.data op definitions. ✕ cla: yes size:XL 1
#31336 by mihaimaruseac was merged on 5 Aug • Approved

Can not use large dimension in Embedding layer on GPU(s). TF 2.0 comp:dist-strat 27
comp:keras type:bug
#31162 opened on 30 Jul by rishabhsahrawat

speedup reduce op grads when keep_dims=True ✕ cla: yes comp:ops ready to pull 9
size:S
#31106 by piiswrong was merged on 15 Aug • Approved

[TF2.0]: Skipping optimization due to error while loading function TF 2.0.0-beta0 40
comp:apis type:bug
#30263 by Slacker-WY was closed on 7 Jul

Set 1.14 forward compatibility date to 9/10 ✕ cla: yes size:XS 1
#29764 by bananabowl was merged on 14 Jun • Approved

Set the forward-compat date to the future. ✕ cla: yes size:XS
#28906 by josh11b was merged on 22 May

TF 1.13.1 SequenceExample & Dataset/Estimator Saved Model: No attr named 'Ncontext_sparse' in NodeDef TF 1.13 comp:data stat:awaiting tensorflower type:bug 1

https://github.com/tensorflow/tensorflow/issues?utf8=%E2%9C%93&q=forward+compatibility+

Either use pre-built images for known tasks or build your own image (and publish)

- Pull =~ Clone
- Publish on dockerhub =~ Publishing code on github (optional)
- Can use prebuilt images for almost all popular services.
- Instead of setting up on local and following a set of instructions, just docker-pull + docker run

https://hub.docker.com/search?q=&type=image&category=analytics%2Capplication_framework%2Capplication_infrastructure%2Capplication_services%2Cbase%2Cdatabase%2Cmessaging%2Cstorage

The docker flow

docker images → Display available images
docker ps → Display running containers

- Pull / build image
  docker pull image-name
- Run image → starts a container using docker run image-name + options

- Check: https://hub.docker.com/_/wordpress/
- docker run --name some-wordpress -p 8080:80 -d wordpress

mantissaTalk ▸ complex_app ▸ Dockerfile ▸ ...

```
FROM ubuntu
RUN apt-get -y  update  --fix-missing
RUN apt-get  install  -y python2.7 python-pip python-dev  build-essential
# Install Nginx
RUN apt-get install -y  nginx
# Copy requirements
COPY ./requirements.txt /root/app/
# Install requirements
WORKDIR /root/app/
RUN pip install -r requirements.txt
# Copy all other stuff
COPY ./ /root/app/
WORKDIR /root/app/

# Enable the new webservice to server using nginx
RUN cp nginx_app_block /etc/nginx/sites-available/
RUN rm /etc/nginx/sites-available/default
RUN rm /etc/nginx/sites-enabled/default
RUN ln -s /etc/nginx/sites-available/nginx_app_block /etc/nginx/sites-enabled/

RUN chmod +x ./start.sh
CMD ["./start.sh"]
```

Ref: https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uswgi-and-nginx-on-ubuntu-18-04

Some must-know stuff: Docker volumes

- Lifetime of a container : When a container is stopped, whatever you saved in it is lost.
- Containers are stateless
- Need to persist data between a container.

Some must-know stuff: Docker networking

3 types of networks:

- Bridge → A default network common for all containers
- Host → Same network as your host machine
- Custom → Define a network and run container in that network
- Expose ports to host using -p option
  Containers in same network can talk to each other using the container-names
  Each network is an isolated island with it’s own set of IPs and set of ports (per container)

docker run command

docker run -it -d --network=some-network -v some-volume --p 9200:9200 image command

docker run -it ubuntu bash

docker run -it --network repo_scanner_v2_default --restart unless-stopped get_screenshots

docker run --name some-wordpress -p 8080:80 -d wordpress (ref: https://hub.docker.com/_/wordpress/)

docker run -it --name some-wordpress3 -p 8080:80 wordpress bash

https://github.com/docker-library/wordpress/blob/master/Dockerfile-alpine.template

Bringing it all together

4 separate containers:

1. api
2. queuing
3. the magic model
4. the database storage(mongo)

Data Flow

Endpoint: my-awesome-app.com/process: [POST request with payload]

Client → Server (API Endpoint) → Validate Payload → Generate a ID → Save ID to database → Push to Queue → Return client the ID

In Background: Process payload, update corresponding answers in the database .

Endpoint: my-awesome-app.com/process?id=1234:

Query database → Is processed ? → return

Magical-results

else return

Check again later

END!!
Questions/Feedback ?