Flask API Chapter 3:
Integrate all components into one service app

Roger

Recap

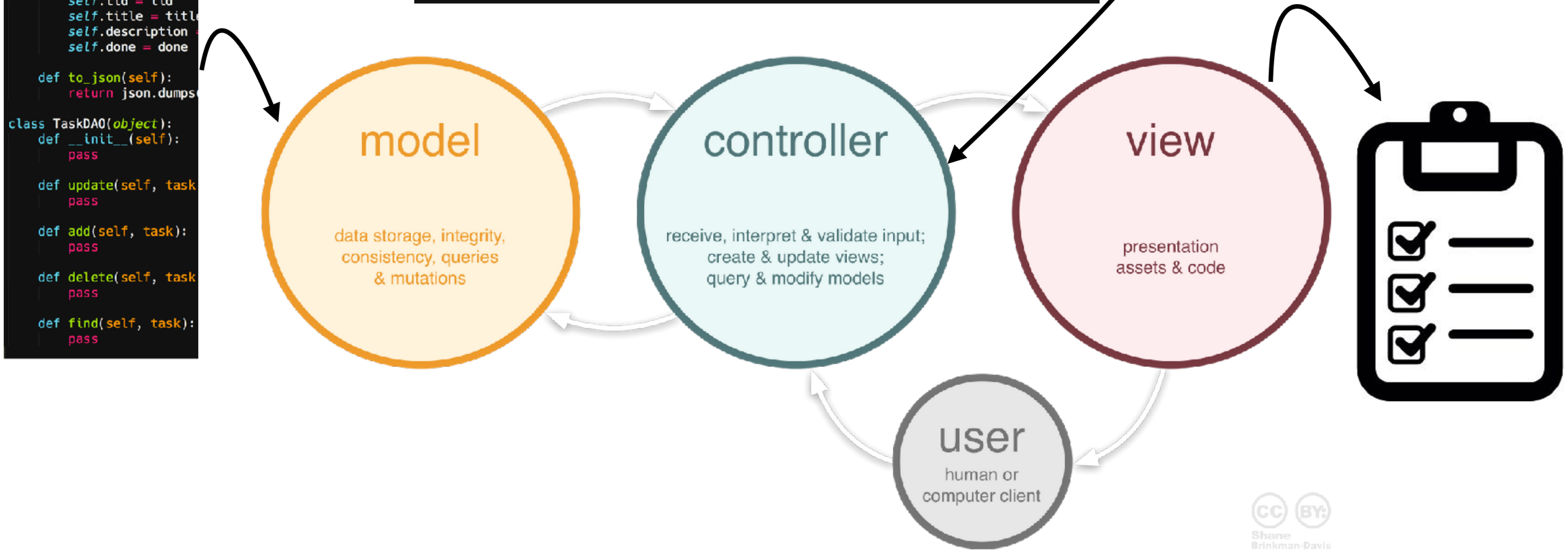
Chapter 1: 2017.07.07-FlaskAPI


```python
#!/usr/local/bin/python
from flask import Flask, request, jsonify
from pymongo import MongoClient
import json

app = Flask(__name__)

#Mongo connection
MONGO_HOST = 'localhost' #88.8.141.118
MONGO_DB = 'mydb'
client = MongoClient(MONGO_HOST)
db = client[MONGO_DB]

@app.route('/userscore', methods = ['POST'])
def score():
    '''
    Type username and interest then return the
    '''
    json_data = request.get_json(force=True)
    username = json_data['username']
    interest = json_data['interest']
    collect = db.interest.find({
        '$and':[
            {'name':username},
            {'interest':interest}
        ]})
    return jsonify(collect[0]['score'])

if __name__ == "__main__":
    app.run(host='0.0.0.0', port='5000')
```

Chapter 2: 2017.07.13-MVC

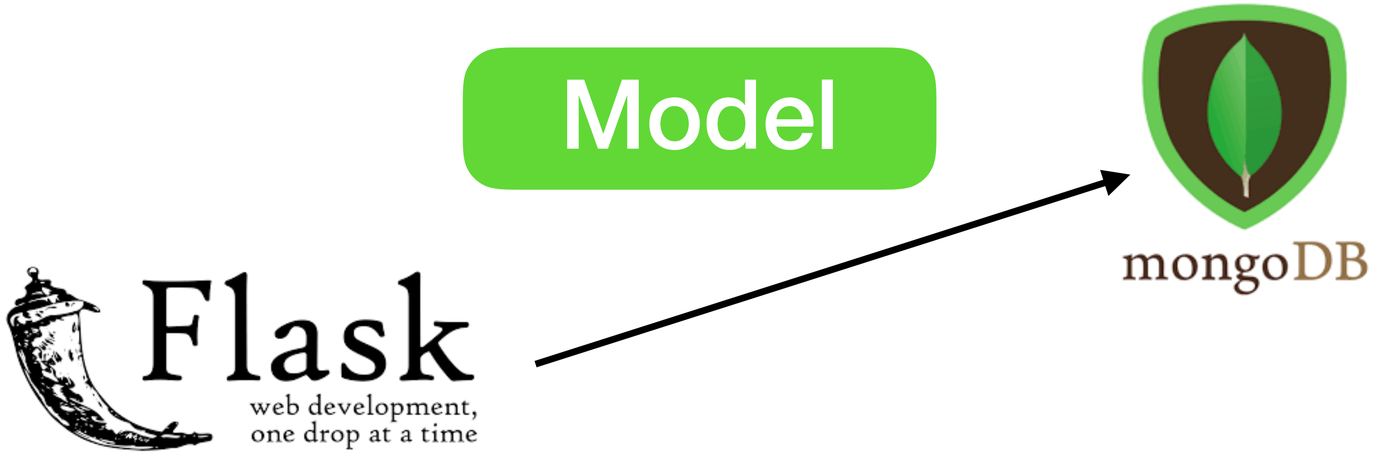
```python
#curl -i http://localhost:5000/todo/api/v1.0/tasks/2
@app.route('/todo/api/v1.0/tasks/<int:task_id>', methods=['GET'])
def get_task(task_id):
    task = [task for task in tasks if task['id'] == task_id]
    if len(task) == 0:
        abort(404)

    return jsonify({'task': task[0]})
```

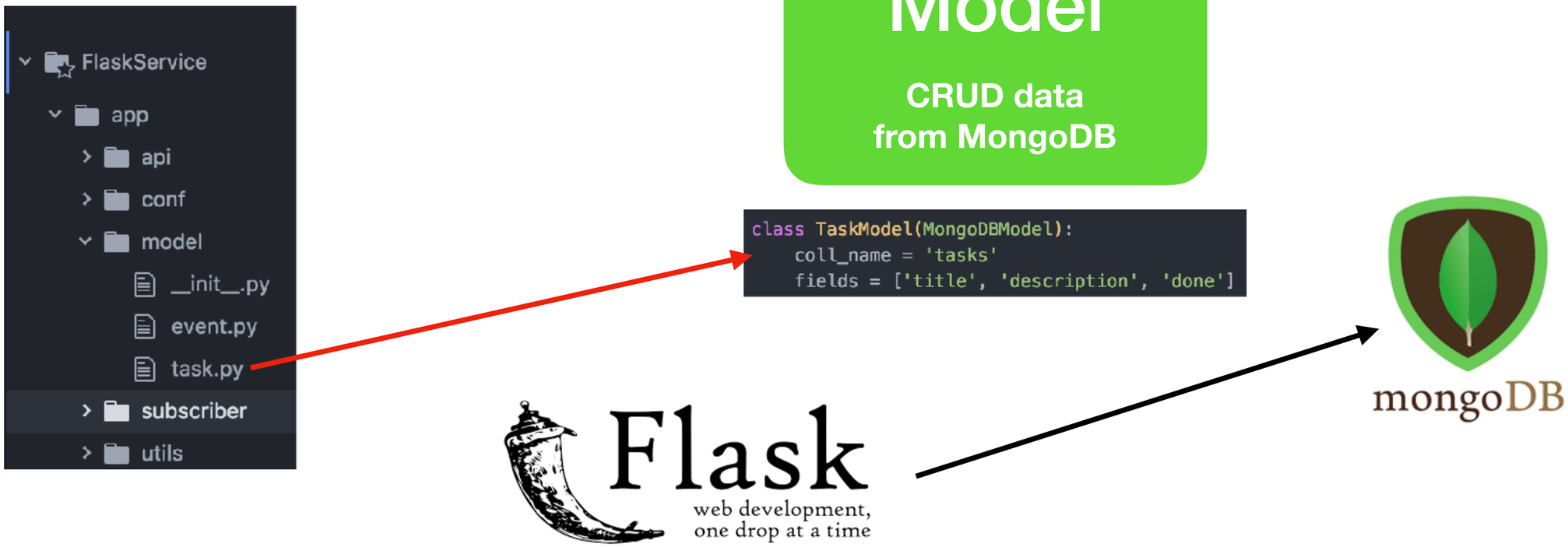
```python
import json

class Task(object):
    def __init__(self, ti
        self.tid = tid
        self.title = titl
        self.description
        self.done = done

    def to_json(self):
        return json.dumps

class TaskDAO(object):
    def __init__(self):
        pass

    def update(self, task
        pass

    def add(self, task):
        pass

    def delete(self, task
        pass

    def find(self, task):
        pass
```



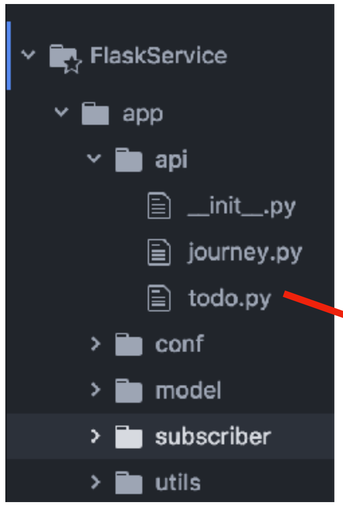
Let’s summarize
the final service structure


Flask
web development,
one drop at a time


Model
Flask
web development,
one drop at a time
mongoDB


FlaskService
app
api
conf
model
__init__.py
event.py
task.py
subscriber
utils
Model
CRUD data
from MongoDB
class TaskModel(MongoDBModel):
coll_name = 'tasks'
fields = ['title', 'description', 'done']
Flask
web development,
one drop at a time
mongoDB


Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB



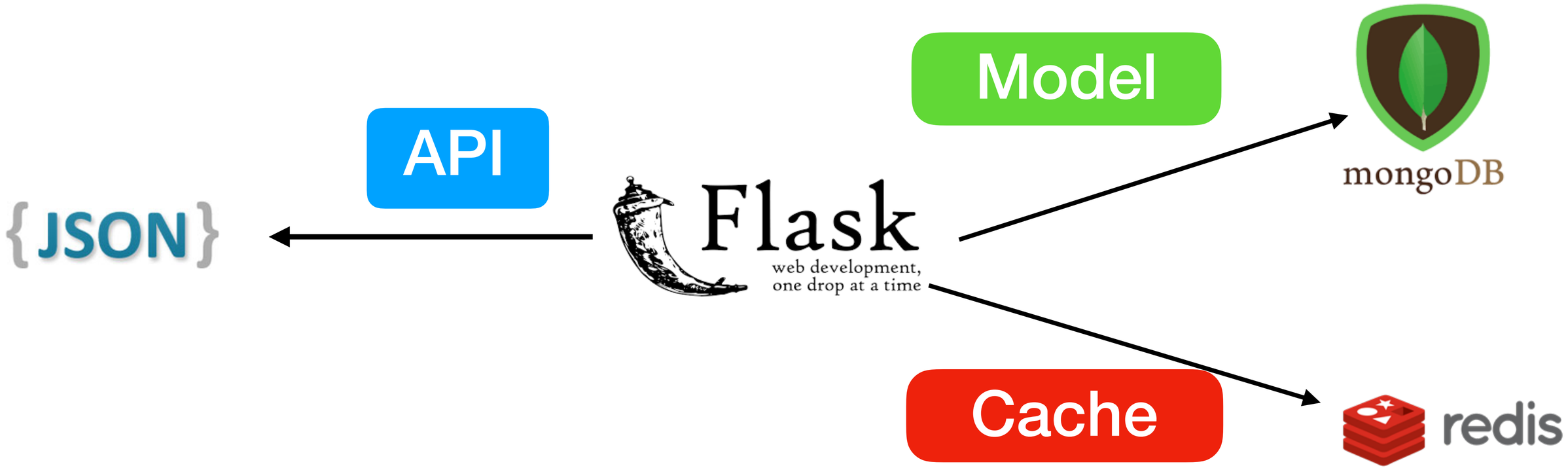
API

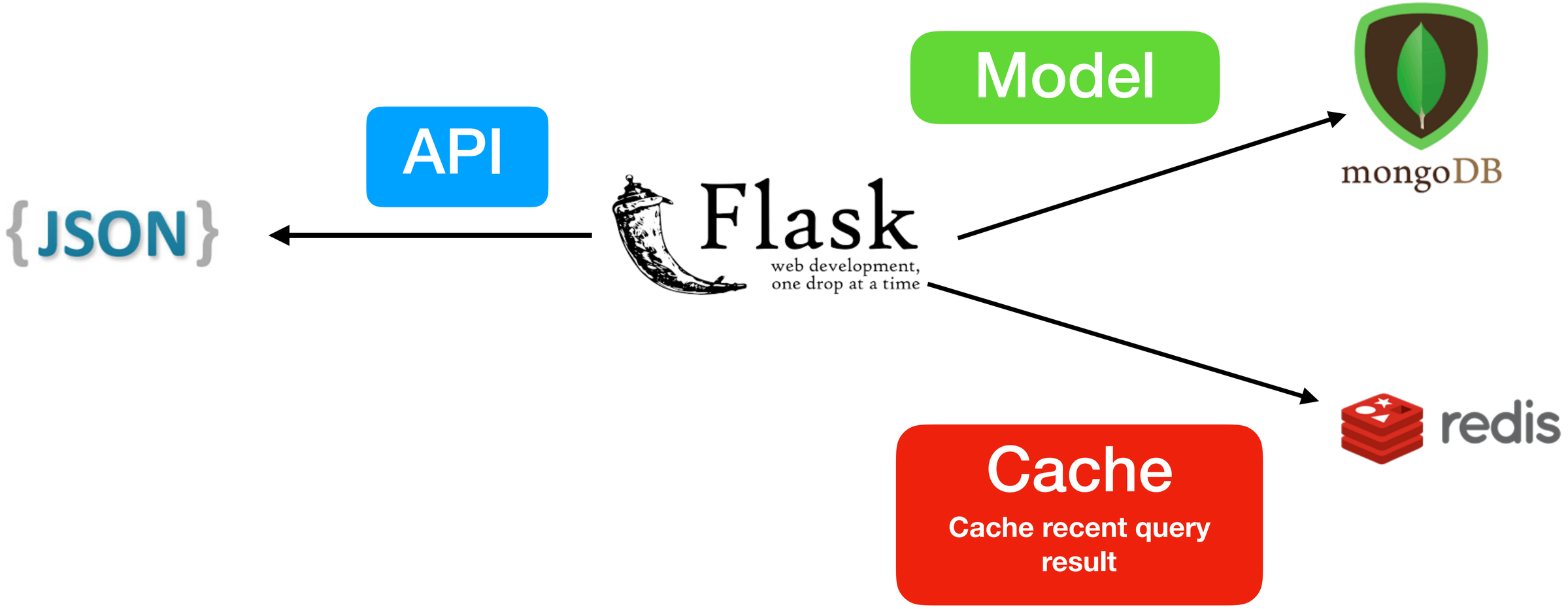
* Route URI to controller
* Business logic processing

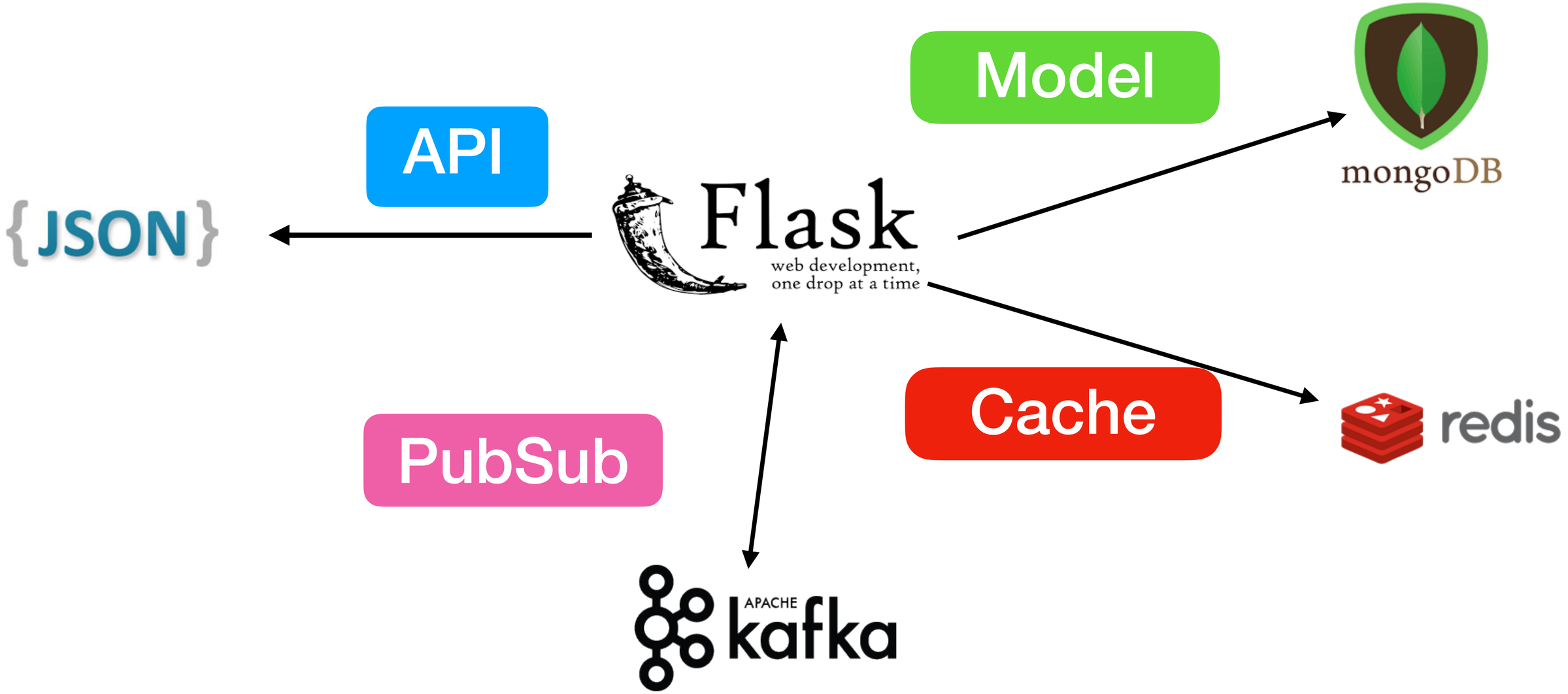
```python
@ctrler.route('/tasks', methods=['GET'])
def list_tasks():
    tasks = task_model.find()
    return jsonify({'tasks': tasks})
```

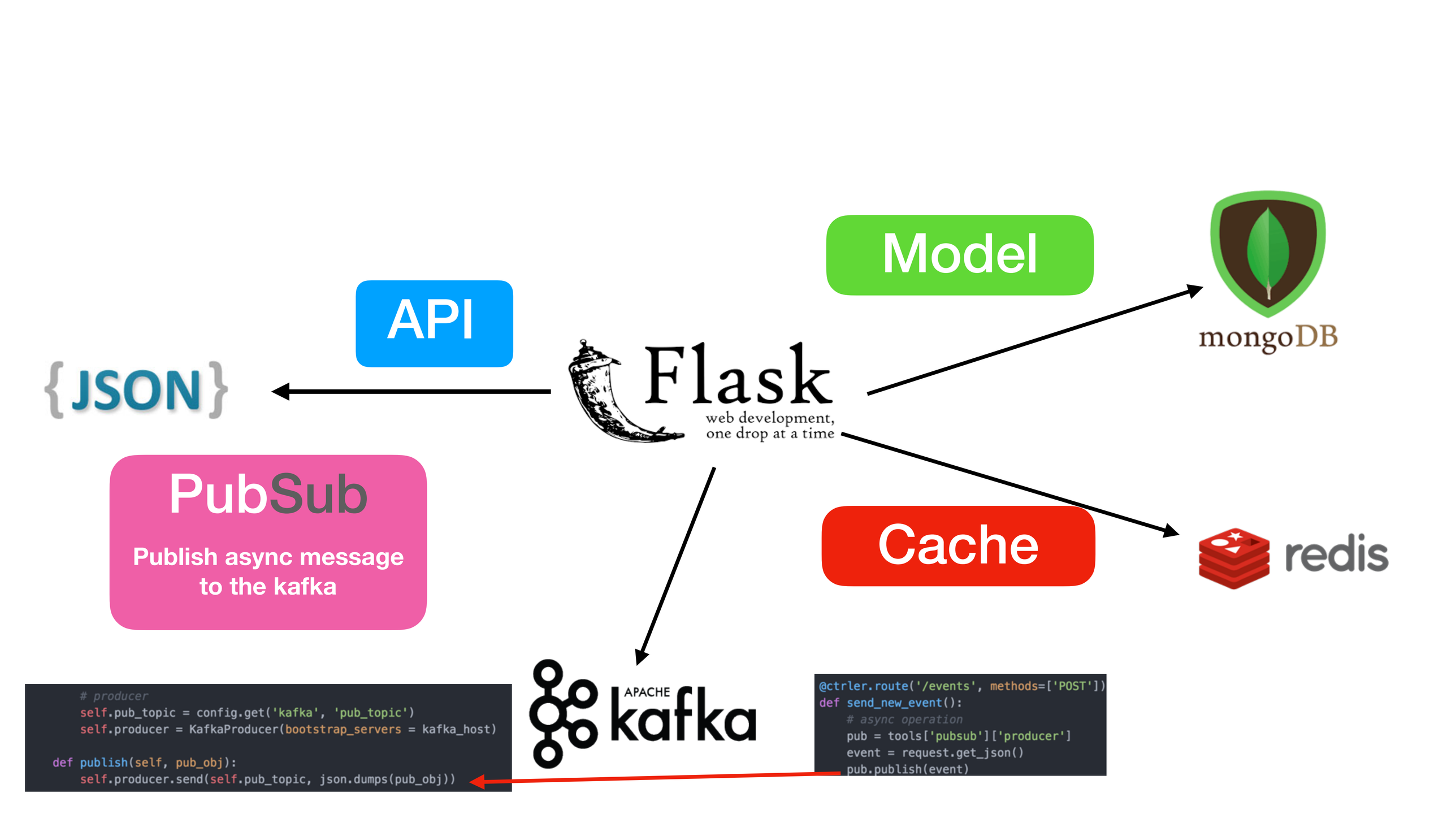
Model

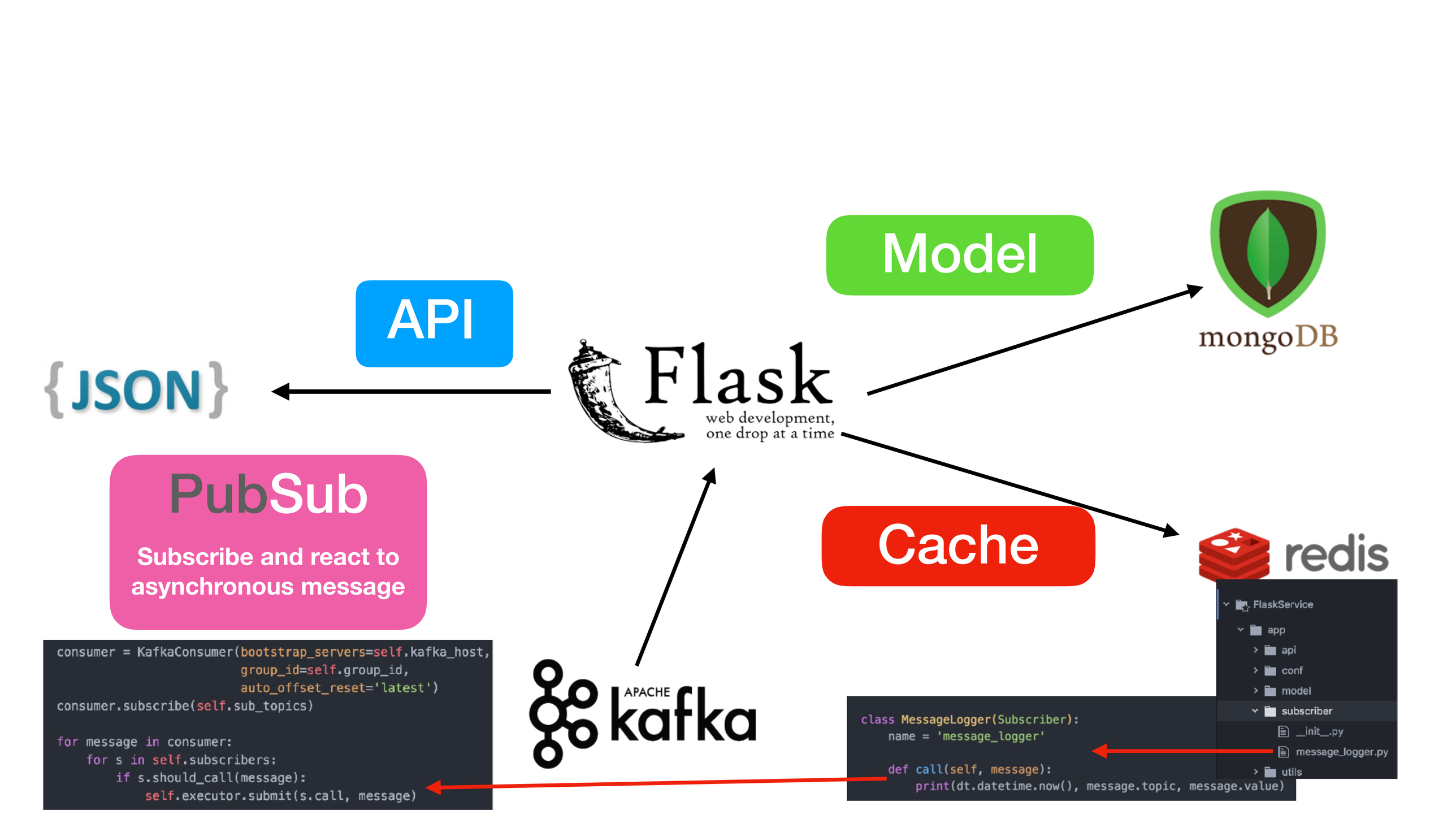

```
{
    "tasks": [
        {
            "_id": "59702b54c412980aaa5aad91",
            "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
            "done": false,
            "title": "Buy groceries"
        },
        {
            "_id": "59702b54c412980aaa5aad92",
            "description": "Need to find a good Python tutorial on the web",
            "done": false,
            "title": "Learn Python"
        }
    ]
}
```

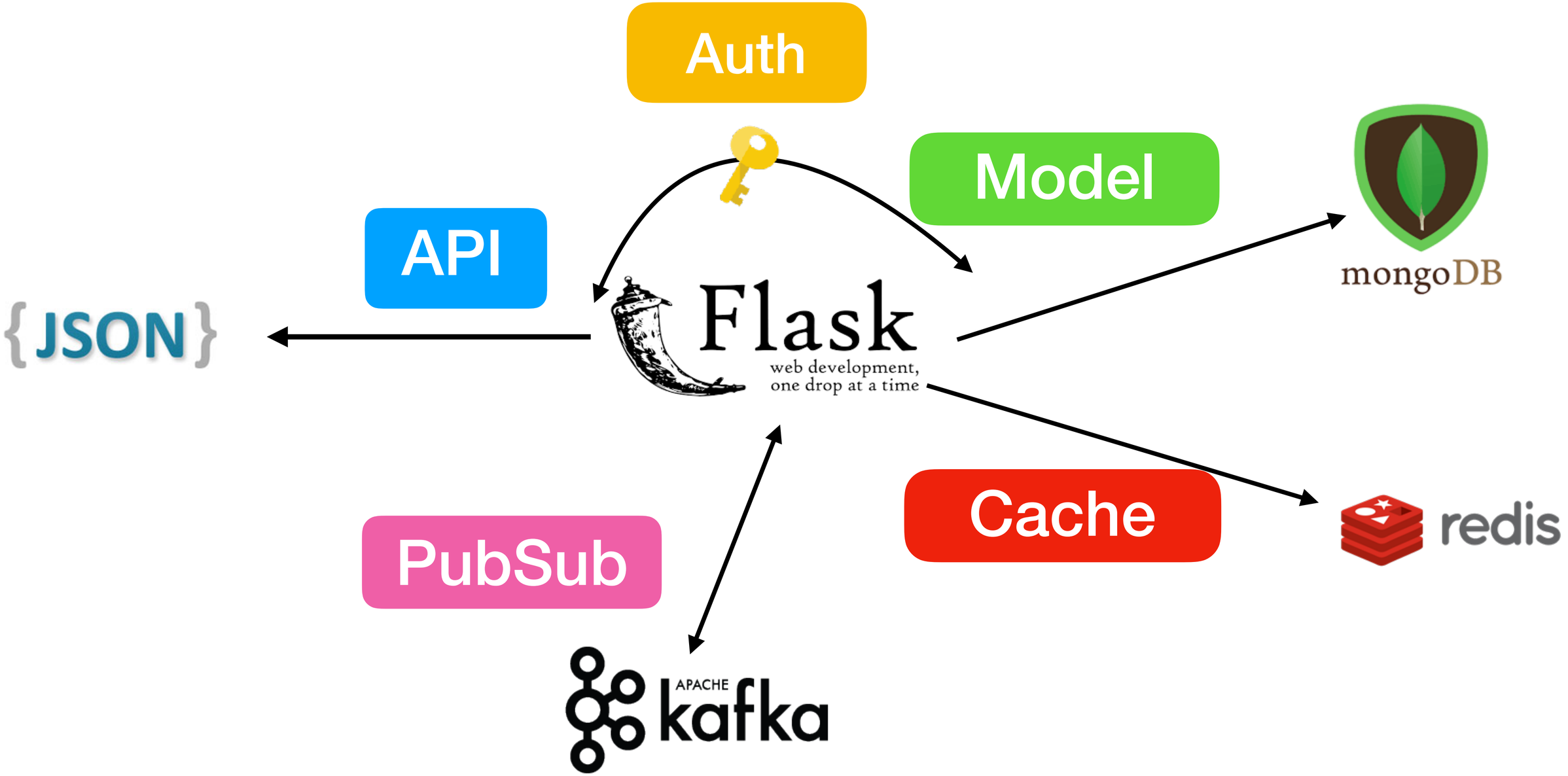

Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
Cache
redis

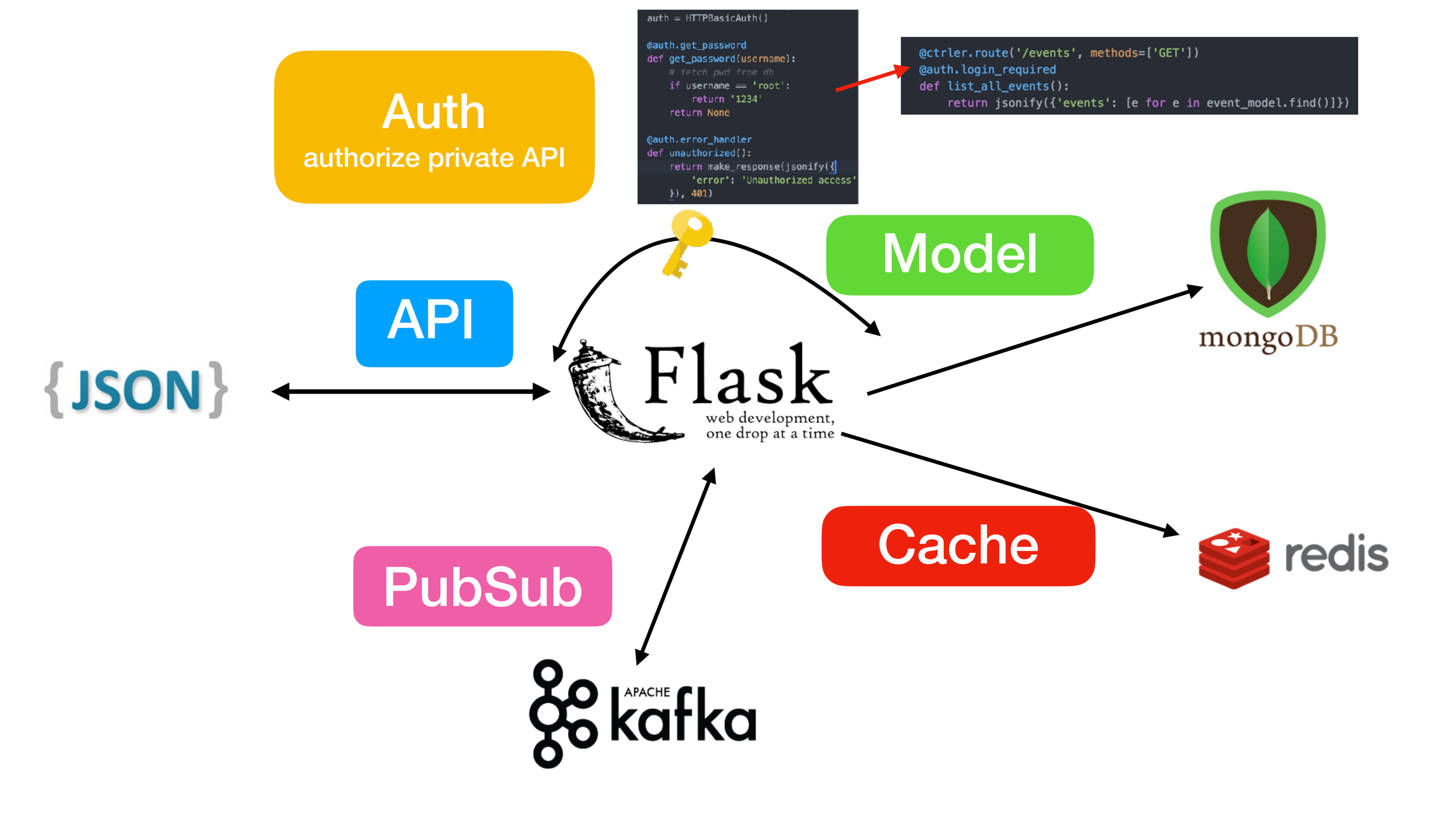

Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
redis
Cache
Cache recent query result
@ctrler.route('/actors/<actor_id>/events', methods=['GET'])
@cache.cached(key_prefix=make_cache_key, timeout=30)
def get_actor_events(actor_id):


Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
Cache
redis
PubSub
APACHE
kafka


Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
PubSub
Publish async message
to the kafka
Cache
redis
APACHE
kafka
# producer
self.pub_topic = config.get('kafka', 'pub_topic')
self.producer = KafkaProducer(bootstrap_servers = kafka_host)
def publish(self, pub_obj):
self.producer.send(self.pub_topic, json.dumps(pub_obj))
@ctrler.route('/events', methods=['POST'])
def send_new_event():
# async operation
pub = tools['pubsub']['producer']
event = request.get_json()
pub.publish(event)

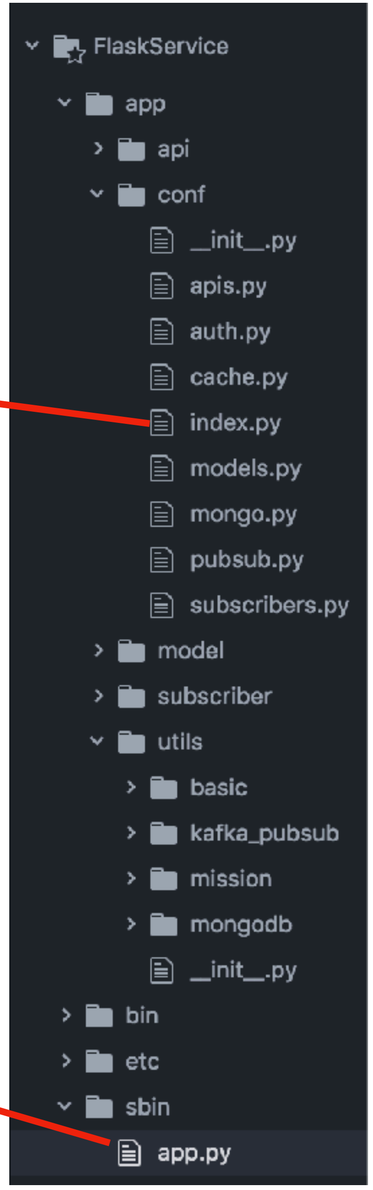

Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
PubSub
Subscribe and react to
asynchronous message
Cache
redis
consumer = KafkaConsumer(bootstrap_servers=self.kafka_host,
group_id=self.group_id,
auto_offset_reset='latest')
consumer.subscribe(self.sub_topics)
for message in consumer:
for s in self.subscribers:
if s.should_call(message):
self.executor.submit(s.call, message)
APACHE
kafka
class MessageLogger(Subscriber):
name = 'message_logger'
def call(self, message):
print(dt.datetime.now(), message.topic, message.value)
FlaskService
app
api
conf
model
subscriber
__init__.py
message_logger.py
utils

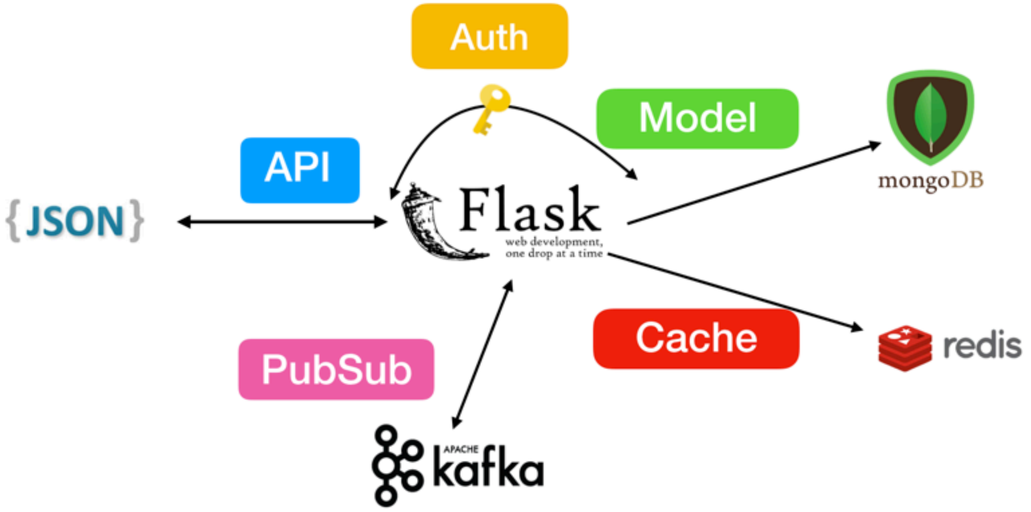

Auth
Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
Cache
redis
PubSub
APACHE
kafka


Auth
authorize private API
auth = HTTPBasicAuth()
@auth.get_password
def get_password(username):
# fetch pwd from db
if username == 'root':
return '1234'
return None
@auth.error_handler
def unauthorized():
return make_response(jsonify({
'error': 'Unauthorized access'
}), 401)
@ctrler.route('/events', methods=['GET'])
@auth.login_required
def list_all_events():
return jsonify({'events': [e for e in event_model.find()]})
Model
API
{JSON}
Flask
web development,
one drop at a time
mongoDB
PubSub
Cache
redis
APACHE
kafka

```python
def configure_app(app, config):
    CORS(app) # cross domain

    tools = {
        'auth': configure_auth(app, config),
        'cache': configure_cache(app, config)
    }

    if config.getboolean('hippo', 'mongo'):
        print('connect MongoDB...')
        tools['mongo'] = configure_mongo(config)

    if config.getboolean('hippo', 'pubsub'):
        print('connect Kafka...')
        tools['pubsub'] = configure_kafka(config)

    models = load_models(config, tools)
    apis = load_apis(config, tools, models)

    if 'pubsub' in tools:
        print('register Subscribers...')
        register_subscribers(config, tools, models)

    if config.getboolean('hippo', 'refresh_data'):
        init_tasks(models)
        init_events(models)

    return tools, apis
```

```python
# flask app
app = Flask(__name__)
_, apis = configure_app(app, config)

@app.errorhandler(404)
def not_found(error):
    return make_response(jsonify({'error': 'Not found'}), 404)


if __name__ == '__main__':
    HOST = config.get('client', 'host')
    PORT = config.getint('client', 'port')

    # journey
    for api in apis:
        print('register API: {}'.format(api['prefix']))
        app.register_blueprint(api['ctrler'], url_prefix=api['prefix'])

    app.run(debug=False, host=HOST, port=PORT, threaded=True)
```





Let’s try it!

github.com/b96705008/FlaskService

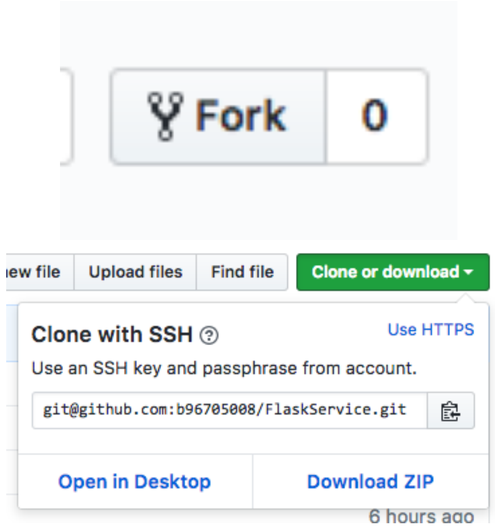
API app using Flask and MongoDB

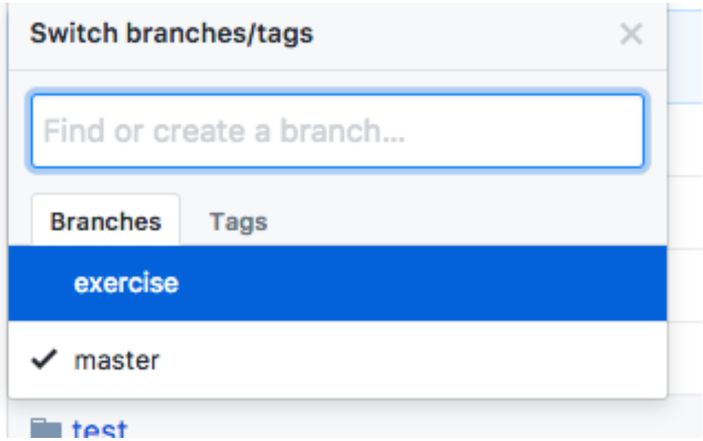
Structure

- app: api server code
- bin: start server script
- etc: config file
- sbin: entry app python file

app folder

- app/conf: flask app initial setting (cache, mongo, auth...)
- app/model: model which connect to MongoDB (or others)
- app/api: flask blueprint route and controller
- app/subscriber: subscribers which process async message from Kafka
- app/utils: mongodb, kafka, paginator related utilities





References

lagom

Other good API framework

StrongLoop

An IBM Company

play