RESTful microservices with python

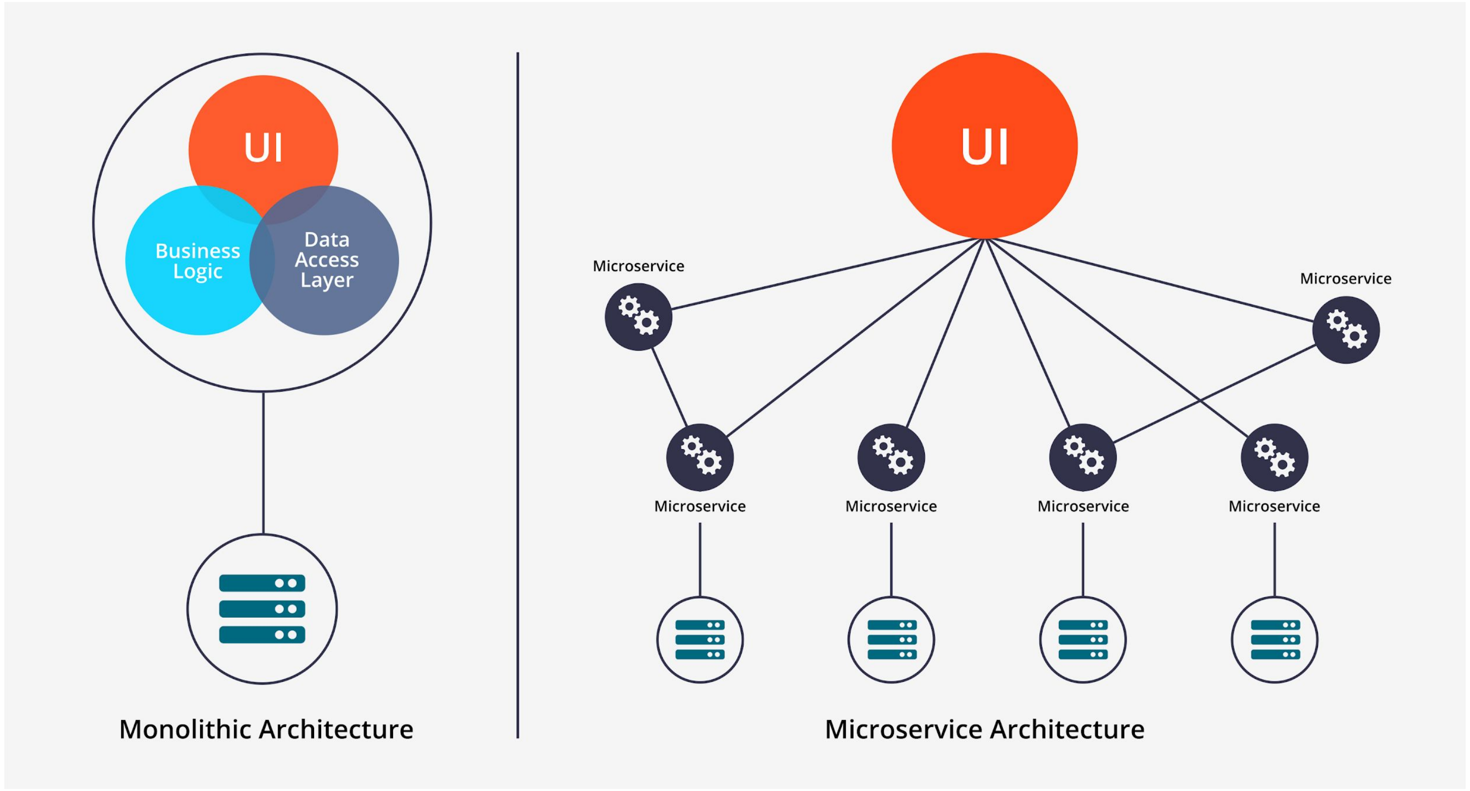
https://github.com/lastlegion/Python_Microservices_Meetup

Ganesh Iyer
twitter: @lastlegion
github: @lastlegion
http://iyer.ai

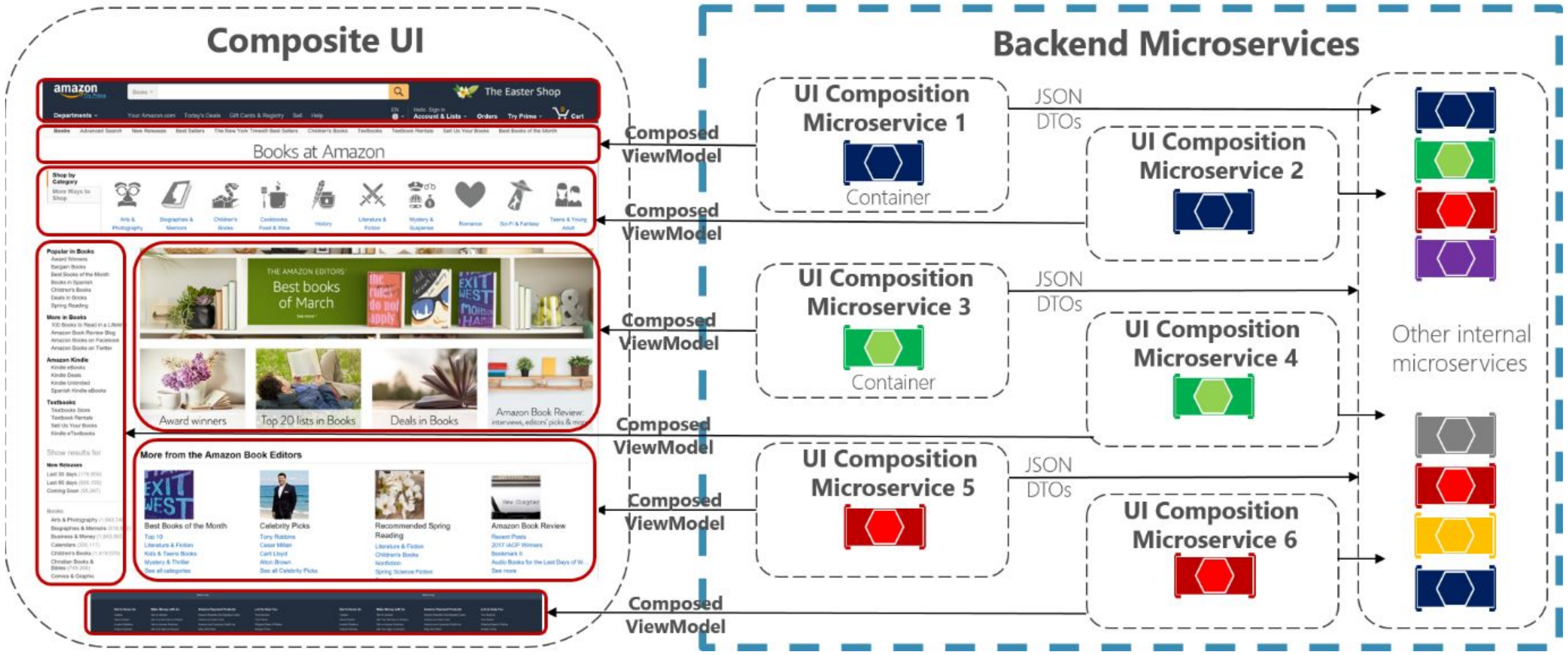
RESTful MicroServices with Python

- Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services.
- Introduced in 2000 by Roy Fielding in his dissertation
- For the purposes of this tutorial we'll be using **HTTP** as the transport layer for REST.

RESTful MicroServices with Python



RESTful MicroServices with Python



Communication Patterns in Microservices

Message formats:

- JSON
- XML
- Protobuf

Transports:

- HTTP
- gRPC
- Kafka

MOTIVATION: MonoLiths vs Microservices

- Understanding
- Tight coupling
- Making changes, deployments
- Scalabilities: components can't be scaled independently of other components
- Embracing new technologies
- Risk

Agenda

- Quick introduction about REST + Microservices
- Installation: Git, virtualenv, flask etc.
- Create a hello world python REST app
- RESTful clients and testing your rest endpoints
- TODO list app with REST template
- Todo list app (Clean Code): service layer + repository layer
- Deployment, packaging with Docker etc.

Setup

- Install git, python, virtualenv

  ```
  git clone https://github.com/lastlegion/Python_Microservices_Meetup.git
  ```

- Activate virtualenv

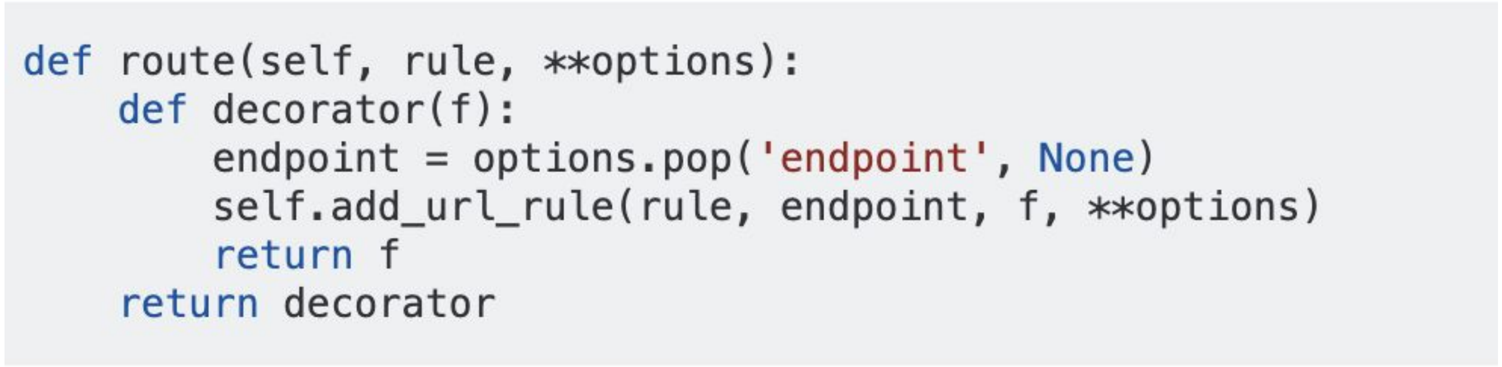
  ```
  pip install -r requirements.txt
  ```

- Install flask (pip install flask)

Create a Few Hello World Endpoints

- goto hello_world/server/
- export FLASK_APP=main.py
- python -m flask run (To run the server)

A note on decorators:(https://realpython.com/primer-on-python-decorators/)



```python
def route(self, rule, **options):
    def decorator(f):
        endpoint = options.pop('endpoint', None)
        self.add_url_rule(rule, endpoint, f, **options)
        return f
    return decorator
```

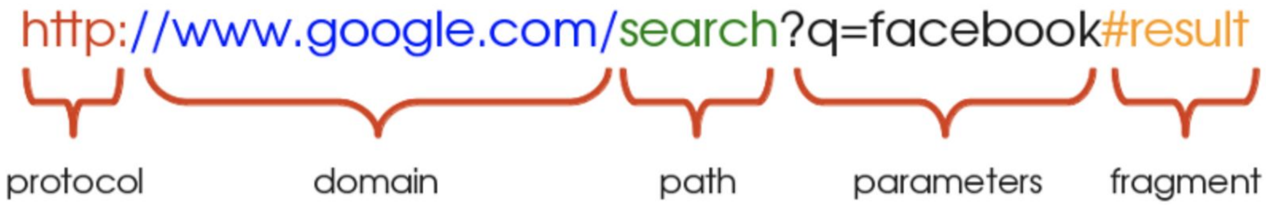
Consuming REST Endpoints

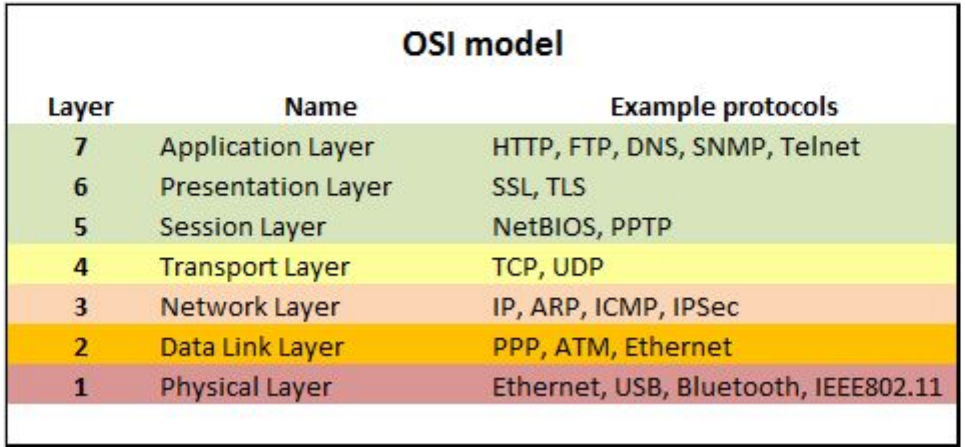
- goto hello_world/client
- python client.py

Other ways to consume REST endpoints:

- Postman
- cURL etc.

Quick Guide to HTTp: [HyperText Transport Protocol]



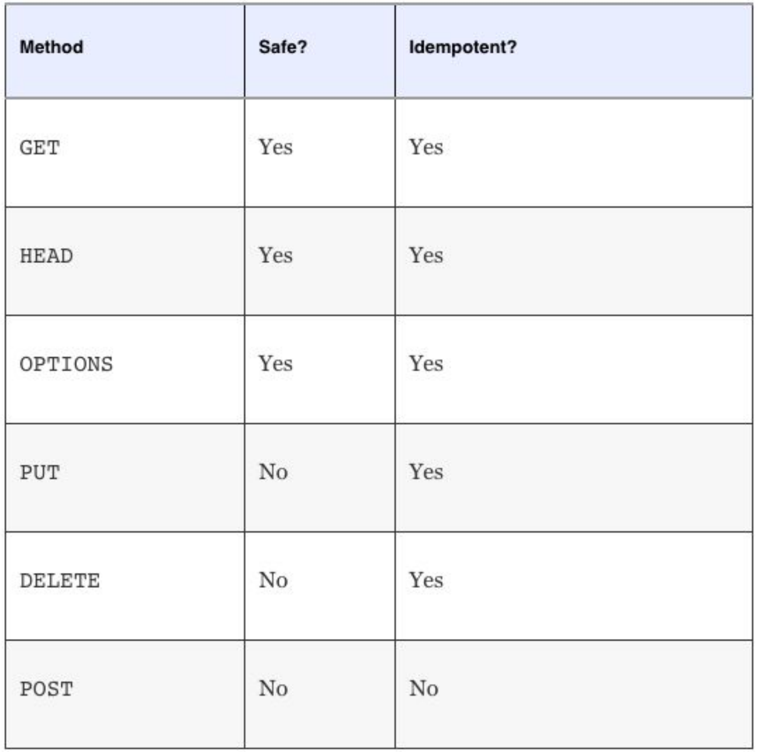

OSI model

Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

Quick Guide to HTTp

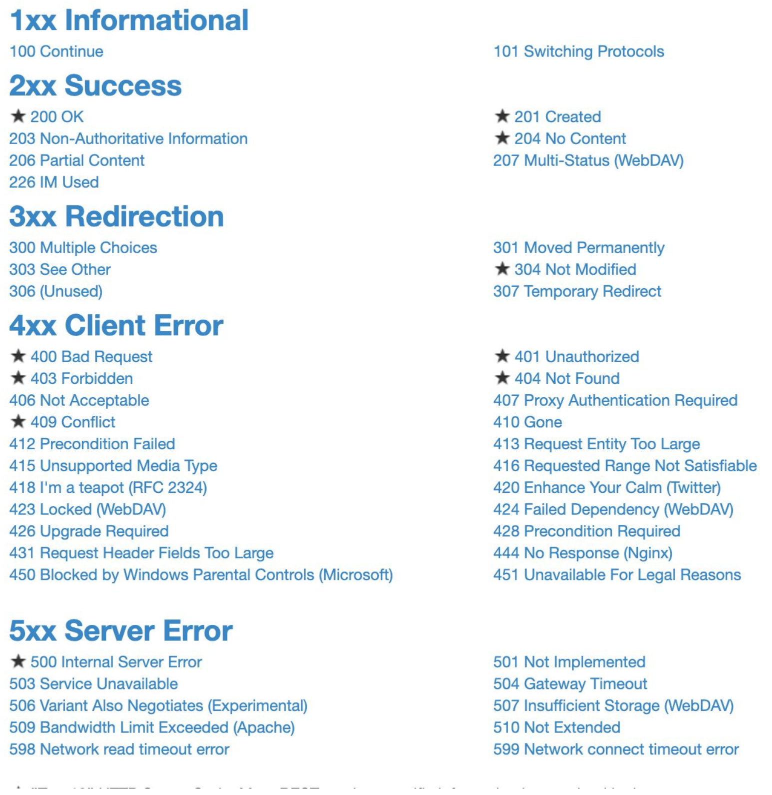
Idempotent method that can be called many times without different outcomes.

Safe methods are HTTP methods that do not modify resources. *Can be cached.*



Method	Safe?	Idempotent?
GET	Yes	Yes
HEAD	Yes	Yes
OPTIONS	Yes	Yes
PUT	No	Yes
DELETE	No	Yes
POST	No	No

Quick Guide to HTTP: Status Codes



1xx Informational

100 Continue
101 Switching Protocols

2xx Success

★ 200 OK
★ 201 Created
203 Non-Authoritative Information
★ 204 No Content
206 Partial Content
207 Multi-Status (WebDAV)
226 IM Used

3xx Redirection

300 Multiple Choices
301 Moved Permanently
303 See Other
★ 304 Not Modified
306 (Unused)
307 Temporary Redirect

4xx Client Error

★ 400 Bad Request
★ 401 Unauthorized
★ 403 Forbidden
★ 404 Not Found
406 Not Acceptable
407 Proxy Authentication Required
★ 409 Conflict
410 Gone
412 Precondition Failed
413 Request Entity Too Large
415 Unsupported Media Type
416 Requested Range Not Satisfiable
418 I'm a teapot (RFC 2324)
420 Enhance Your Calm (Twitter)
423 Locked (WebDAV)
424 Failed Dependency (WebDAV)
426 Upgrade Required
428 Precondition Required
431 Request Header Fields Too Large
444 No Response (Nginx)
450 Blocked by Windows Parental Controls (Microsoft)
451 Unavailable For Legal Reasons

5xx Server Error

★ 500 Internal Server Error
501 Not Implemented
503 Service Unavailable
504 Gateway Timeout
506 Variant Also Negotiates (Experimental)
507 Insufficient Storage (WebDAV)
509 Bandwidth Limit Exceeded (Apache)
510 Not Extended
598 Network read timeout error
599 Network connect timeout error

Todo List: APP

Product Management 101: Features

- Adding items to a list
- Get all items from the list
- Update an existing item in the list
- Delete an item from the list

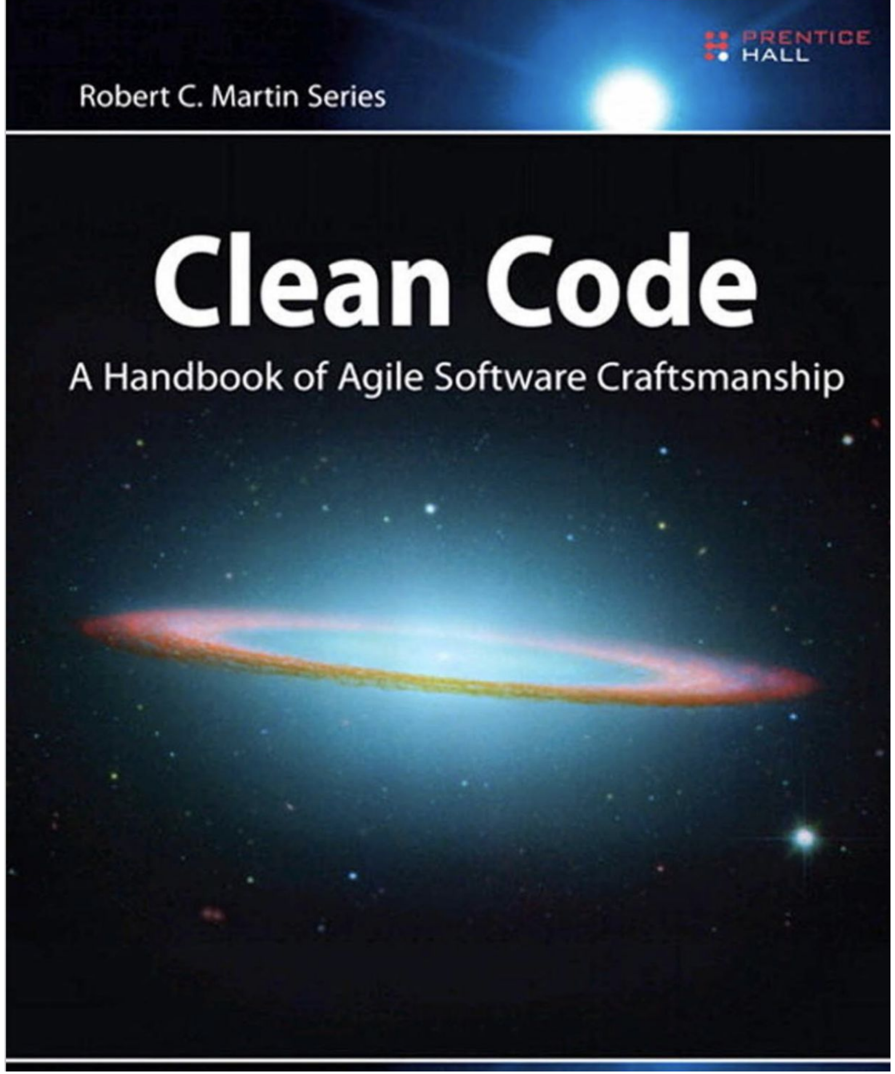
From Requirements to REST

- Adding items to a list	`POST /todo`
- Get all items from the list	`GET /todo`
- Update an existing item in the list	`UPDATE /todo/:id`
- Delete an item from the list	`DELETE /todo/:id`

Clean Code

Separation of concerns

- Transport Layer: REST, gRPC, Kafka etc.
- Service Layer: Core business Logic
- Repository Layer: DB interactions



Deployment: DOcker