PiterPy

TECHNICAL CONFERENCE FOR HARDCORE PYTHON DEVELOPERS

SAINT PETERSBURG
2019 NOVEMBER 1

Python network workflow

REST, JSON, GraphQL or gRPC?

Grigory Petrov

What's next?

Speaker	Grigory Petrov
Specialization	Generalist
Role	DevRel at Evrone
Experience	20 years
Talk time	30 minutes
Questions	At the end of the talk, 15 minutes
Slides	

Let's use social networks to communicate

	grigoryvp	@evrone.com
t.me/	grigoryvp	
fb.com/	grigoryvp	
vk.com/	grigoryvp	
github.com/	grigoryvp	
twitter.com/	grigoryvp	
instagram.com/	grigoryvp	

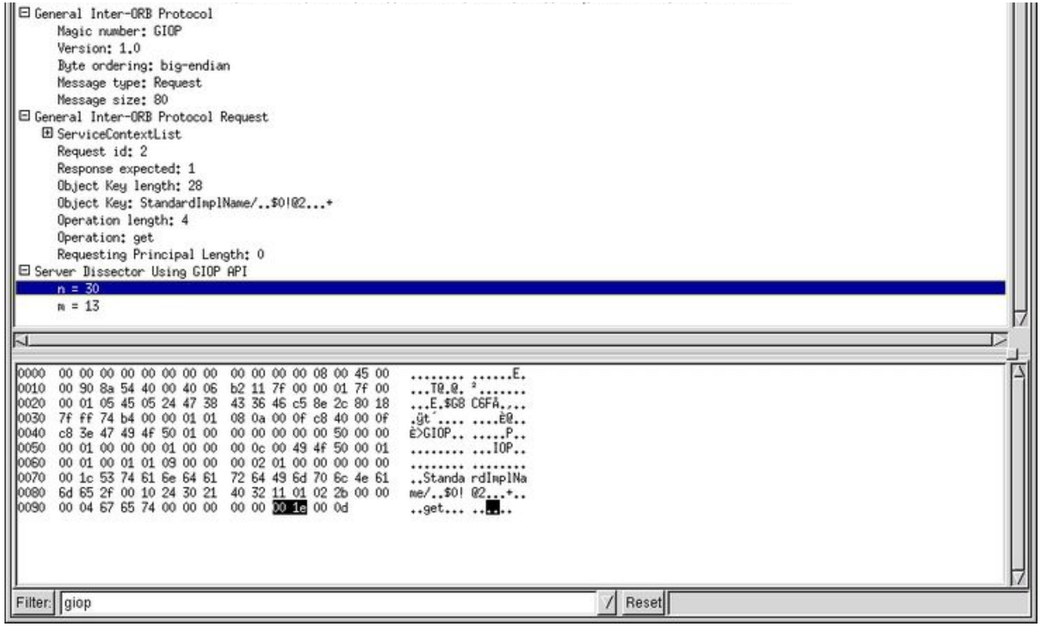
↓

@grigoryvp

Some history of network communications

Some history of network communications

- 1990s: CORBA RPC.



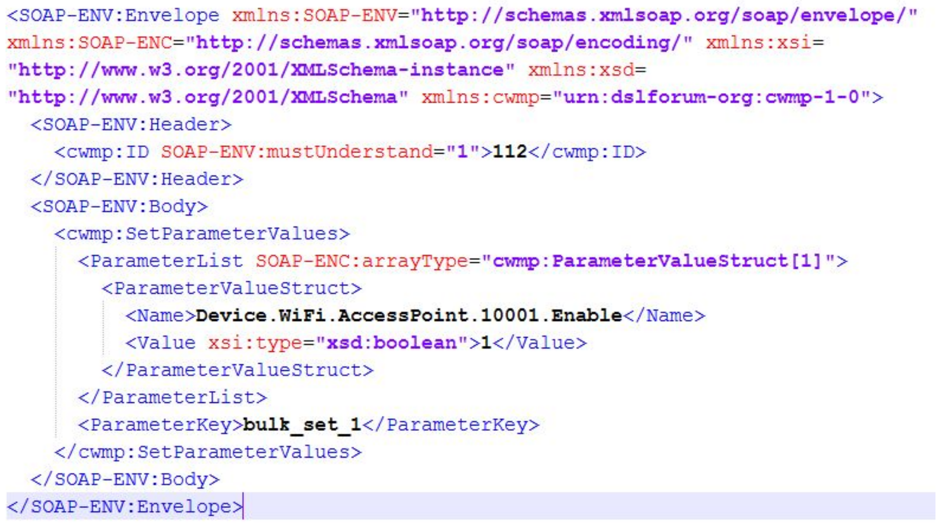
```
⊟ General Inter-ORB Protocol
    Magic number: GIOP
    Version: 1.0
    Byte ordering: big-endian
    Message type: Request
    Message size: 80
⊟ General Inter-ORB Protocol Request
  ⊞ ServiceContextList
    Request id: 2
    Response expected: 1
    Object Key length: 28
    Object Key: StandardImplName/..$0!@2...+
    Operation length: 4
    Operation: get
    Requesting Principal Length: 0
⊟ Server Dissector Using GIOP API
    n = 30
    m = 13
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 90 8a 54 40 00 40 06  b2 11 7f 00 00 01 7f 00   ...T@.@. ².......
0020  00 01 05 45 05 24 47 38  43 36 46 c5 8e 2c 80 18   ...E.$G8 C6FÅ.,..
0030  7f ff 74 b4 00 00 01 01  08 0a 00 0f c8 40 00 0f   .ÿt´.... ....È@..
0040  c8 3e 47 49 4f 50 01 00  00 00 00 00 00 50 00 00   È>GIOP.. .....P..
0050  00 01 00 00 00 01 00 00  00 0c 00 49 4f 50 00 01   ........ ...IOP..
0060  00 01 00 01 01 09 00 00  00 02 01 00 00 00 00 00   ........ ........
0070  00 1c 53 74 61 6e 64 61  72 64 49 6d 70 6c 4e 61   ..Standa rdImplNa
0080  6d 65 2f 00 10 24 30 21  40 32 11 01 02 2b 00 00   me/..$0! @2...+..
0090  00 04 67 65 74 00 00 00  00 00 00 1e 00 0d         ..get... ......
```

Some history of network communications

1990s: CORBA RPC.

- 2000s: SOAP RPC.



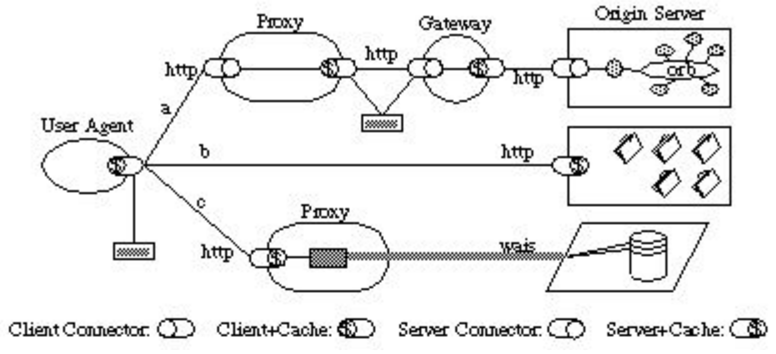
```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
  <SOAP-ENV:Header>
    <cwmp:ID SOAP-ENV:mustUnderstand="1">112</cwmp:ID>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <cwmp:SetParameterValues>
      <ParameterList SOAP-ENC:arrayType="cwmp:ParameterValueStruct[1]">
        <ParameterValueStruct>
          <Name>Device.WiFi.AccessPoint.10001.Enable</Name>
          <Value xsi:type="xsd:boolean">1</Value>
        </ParameterValueStruct>
      </ParameterList>
      <ParameterKey>bulk_set_1</ParameterKey>
    </cwmp:SetParameterValues>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Some history of network communications

1990s: CORBA RPC.

2000s: SOAP RPC.

- Year 2000: Roy Fielding REST doctoral dissertation.



Some history of network communications

1990s: CORBA RPC.

2000s: SOAP RPC.

Year 2000: Roy Fielding REST doctoral dissertation.

- Year 2002: SalesForce, eBay introduce RESTful external APIs.

Some history of network communications

1990s: CORBA RPC.

2000s: SOAP RPC.

Year 2000: Roy Fielding REST doctoral dissertation.

Year 2002: SalesForce, eBay introduce RESTful external APIs.

- Year 2003: Rails with its opinion about REST, JSON and URLs.

Why so popular?

Why so popular?

Complexity offload.

Why so popular?

Complexity offload into:

- URLs.

Why so popular?

Complexity offload into:

URLs.

- HTTP Headers.

Why so popular?

Complexity offload into:

URLs.

HTTP Headers.

- JSON payloads.

Why so popular?

Complexity offload into:

URLs.

HTTP Headers.

JSON payloads.

- Existing browsers and servers.

Why so popular?

Roy Fielding, REST author

Why so popular?

Roy Fielding, REST author

- Co-authored URI and HTTP.

Why so popular?

Roy Fielding, REST author

Co-authored URI and HTTP.

- Battle-tested "Web" since 1994.

Why so popular?

Roy Fielding, REST author

Co-authored URI and HTTP.

Battle-tested "Web" since 1994.

- "REST" **is** "Web".

Why so popular?

Roy Fielding, REST author

Co-authored URI and HTTP.

Battle-tested "Web" since 1994.

"REST" **is** "Web".

- Well suited for CRUD.

Some history of network communications

1990s: CORBA RPC

2000s: SOAP RPC

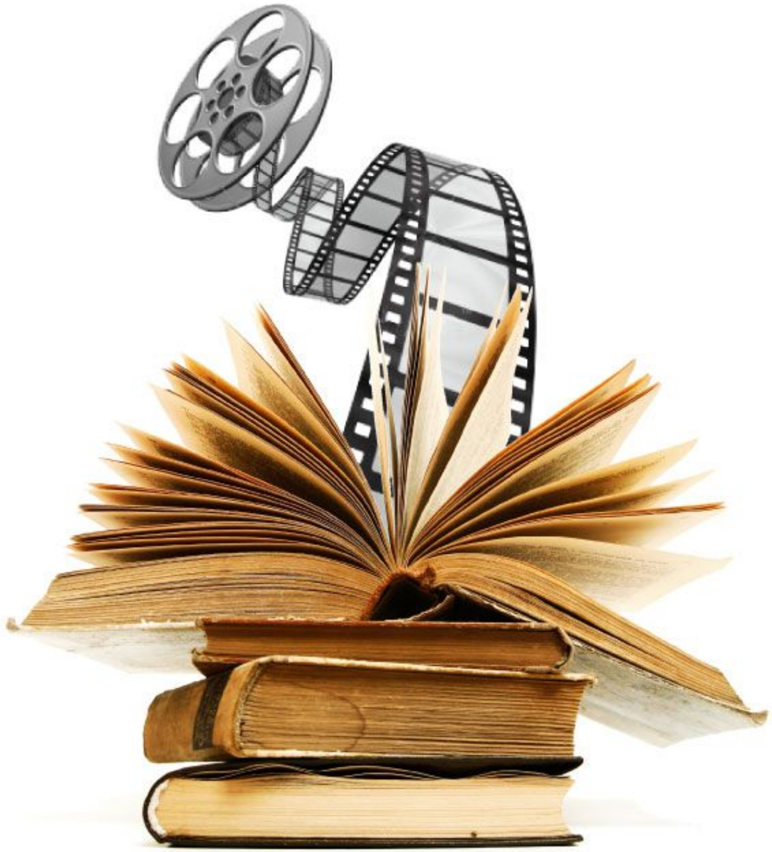
Year 2000: Roy Fielding REST doctoral dissertation.

Year 2002: SalesForce, eBay introduce RESTful external APIs.

Year 2003: Rails with it's opinion about REST, JSON and URLs.

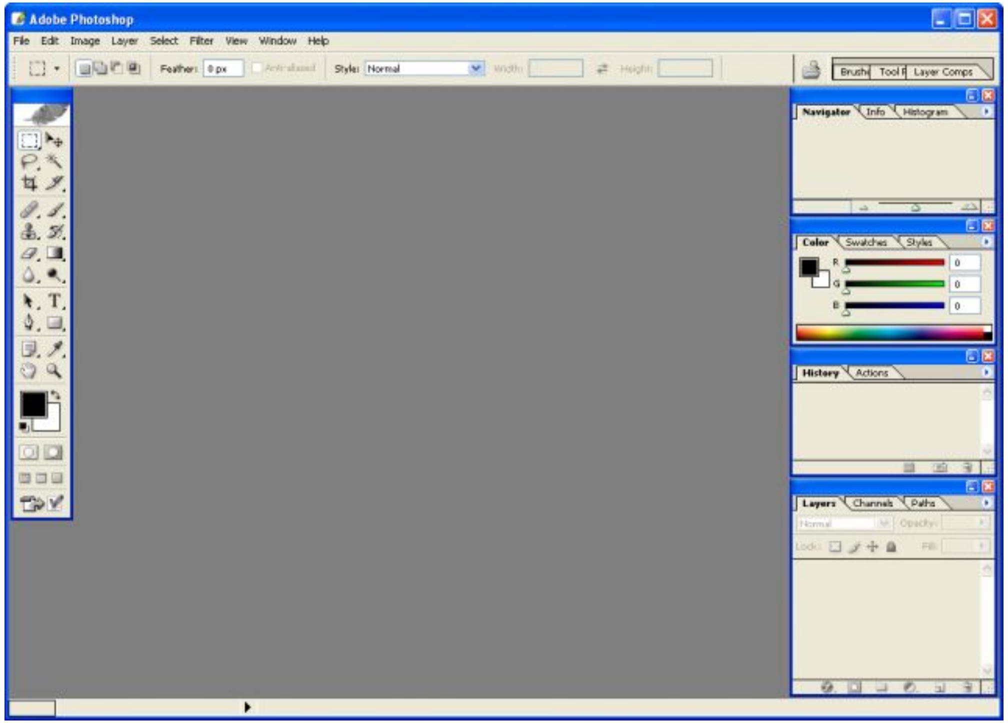
- Year 2004: Gmail

Evolution from SSR to SPA



/evrone.

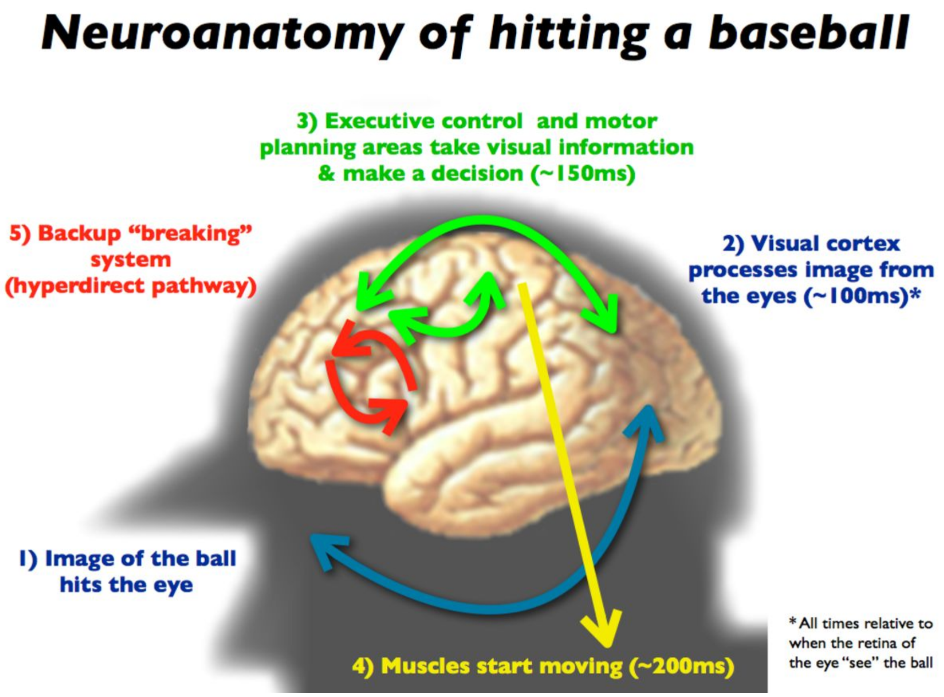
We expect "applications" to be fast



PiterPy

/evrone.

We expect "application" reaction under 150ms



Neuroanatomy of hitting a baseball
3) Executive control and motor planning areas take visual information & make a decision (~150ms)
5) Backup "breaking" system (hyperdirect pathway)
2) Visual cortex processes image from the eyes (~100ms)*
1) Image of the ball hits the eye
4) Muscles start moving (~200ms)
*All times relative to when the retina of the eye "see" the ball

PiterPy

Network efficiency challenge

Network efficiency challenge

- TCP 3-way handshake, graceful shutdown and RTT latency.

/evrone.

Network efficiency challenge

TCP 3-way handshake, graceful shutdown and RTT latency.

- HTTPS handshake.

PiterPy

Network efficiency challenge

TCP 3-way handshake, graceful shutdown and RTT latency.
HTTPS handshake.

- 6-connection limit.

Network efficiency challenge

TCP 3-way handshake, graceful shutdown and RTT latency.

HTTPS handshake.

6-connection limit.

- Underfetching.

Network efficiency challenge

TCP 3-way handshake, graceful shutdown and RTT latency.

HTTPS handshake.

6-connection limit.

Underfetching.

- Over-fetching and internet speed.

Network efficiency challenge

TCP 3-way handshake, graceful shutdown and RTT latency.
HTTPS handshake.
6-connection limit.
Underfetching.
Over-fetching and internet speed.

- Payload size.

/evrone.

REST evolution to answer the efficiency challenge

{REST}

PiterPy

/evrone.

REST evolution to answer the efficiency challenge

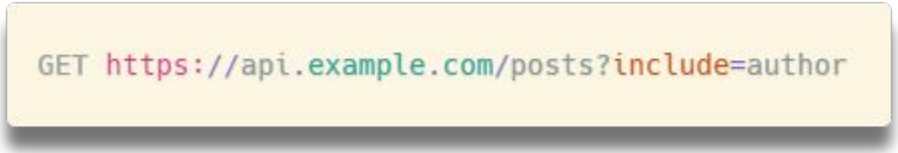
- May 2013, "JSON:API" extracted from Ember.js

PiterPy

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

- Compound documents.



```
GET https://api.example.com/posts?include=author
```

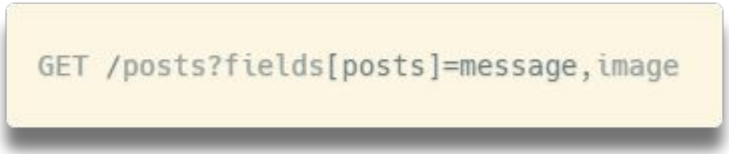
/evrone.

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

- Sparse fieldsets.



```
GET /posts?fields[posts]=message,image
```

PiterPy

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

- And really bad naming.

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming:

- "json api" is same as "web api" or "RESTful api".

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming:

- "json api" is same as "web api" or "RESTful api".
- "jsonapi" (website).

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming:

- "json api" is same as "web api" or "RESTful api".
- "jsonapi" (website).
- "json-api" (github).

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming:

- "json api" is same as "web api" or "RESTful api".
- "jsonapi" (website).
- "json-api" (github).

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming.

- Also, not to confuse with:

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming.

Also, not to confuse with:

- OpenAPI and RAML API definition languages.

REST evolution to answer the efficiency challenge

May 2013, "JSON:API" extracted from Ember.js

Compound documents.

Sparse fieldsets.

And really bad naming.

Also, not to confuse with:

- OpenAPI and RAML API definition languages.
- "JSON Schema" data definition language.

REST evolution to answer the efficiency challenge

REST evolution to answer the efficiency challenge

- Django Rest Framework with **drf_yasg**.

REST evolution to answer the efficiency challenge

Django Rest Framework with **drf_yasg**.

- **Connexion**: Flask, can consume "swagger.yaml"

REST evolution to answer the efficiency challenge

Django Rest Framework with **drf_yasg**.

Connexion: Flask, can consume "swagger.yaml"

- **Falcon**: threads, native REST support.

/evrone.

REST evolution to answer the efficiency challenge

Django Rest Framework with **drf_yasg**.

Connexion: Flask, can consume "swagger.yaml"

Falcon: threads, native REST support.

- **Eve**: specifically for REST.

PiterPy

REST evolution to answer the efficiency challenge

Django Rest Framework with **drf_yasg**.
Connexion: Flask, can consume "swagger.yaml"
Falcon: threads, native REST support.
Eve: specifically for REST.

- **aiohttp-apispec**

REST evolution to answer the efficiency challenge

Django Rest Framework with **drf_yasg**.

Connexion: Flask, can consume "swagger.yaml"

Falcon: threads, native REST support.

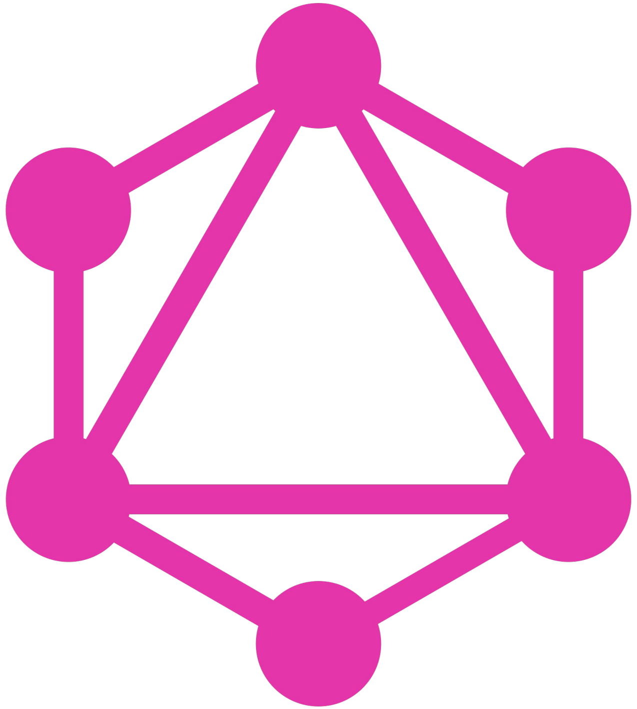
Eve: specifically for REST.

aiohttp-apispec

- ... and much more.

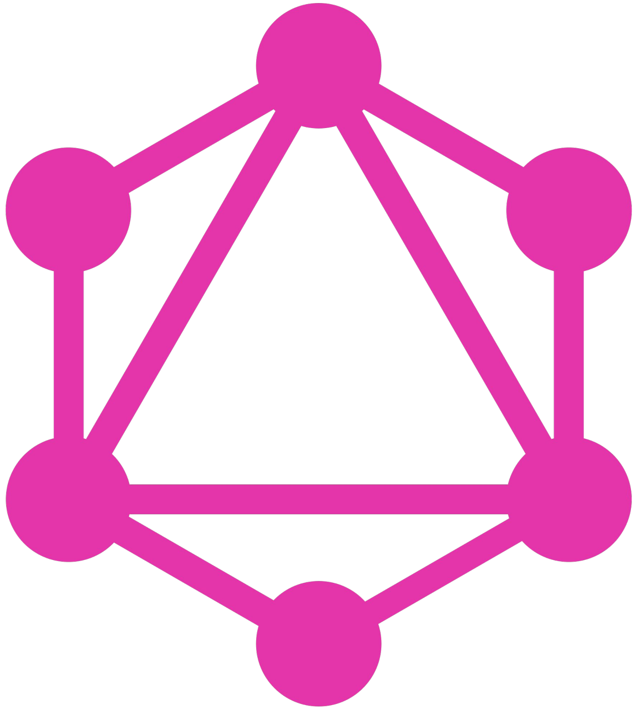
/evrone.

Other challengers: GraphQL



PiterPy

Other challengers: GraphQL



Other challengers: GraphQL

- Publicly released by Facebook in 2015.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

- Based on RESTish Graph API and FQL experience.

Other challengers: GraphQL

Publicly released by Facebook in 2015.
Based on RESTish Graph API and FQL experience.
- Trades REST "uniform interface" for transfer efficiency.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

- At a cost.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.
- Cache.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.
- ○ Cache, auth.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.
- Cache, auth, pagination.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.
- Cache, auth, pagination, duplicates.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.
- Cache, auth, pagination, duplicates, binary.

Other challengers: GraphQL

Publicly released by Facebook in 2015.

Based on RESTish Graph API and FQL experience.

Trades REST "uniform interface" for transfer efficiency.

At a cost:

- N+1 issue.
- No namespaces, scheme is flat.
- Cache, auth, pagination, duplicates, binary, recursion.

Other challengers: GraphQL

Other challengers: GraphQL

- **Graphene** with **graphene-django**.

Other challengers: GraphQL

Graphene with **graphene-django**.

- **Ariadne**, **Strawberry**, **Tartiflette**, **tartiflette-aiohttp**.

Other challengers: gRPC

Other challengers: gRPC

- Publicly released by Google in 2015.

Other challengers: gRPC

Publicly released by Google in 2015.

- Trades REST "resources" for transfer efficiency.

Other challengers: gRPC

Publicly released by Google in 2015.
Trades REST "resources" for transfer efficiency.

- Fast, low-level, backend-to-backend.

Other challengers: gRPC

Other challengers: gRPC

- Official **grpcio-tools** generator from Google.

Other challengers: gRPC

Official **grpcio-tools** generator from Google.

- **mypy-protobuf** from Dropbox.

/evrone.

Other challengers: HTTP/2

HTTP/2

Other challengers: HTTP/2

- Spec published in 2015.

Other challengers: HTTP/2

Spec published in 2015.

- Fixes TCP and HTTP issues.

Other challengers: HTTP/2

Spec published in 2015.

Fixes TCP and HTTP issues.

- Brings back REST!

Other challengers: HTTP/2

Other challengers: HTTP/2

- **Hypercorn** with ASGI for **Quart**.

Other challengers: HTTP/2

Hypercorn with ASGI for **Quart**.

- **Hyper-h2** or **httpx** for HTTP/2 clients (alpha versions!).

Other challengers: HTTP/2

Hypercorn with ASGI for **Quart**.

Hyper-h2 or **httpx** for HTTP/2 clients (alpha versions!).

- **Django-channels**, **Sanic, Twisted**.

Other challengers: HTTP/2

Hypercorn with ASGI for **Quart**.

Hyper-h2 or **httpx** for HTTP/2 clients (alpha versions!).

Django-channels, **Sanic, Twisted**.

- Or just use the HTTP/2 proxy.

~~Conclusion~~ What I want to discuss at this conference

~~Conclusion~~ What I want to discuss at this conference

- GraphQL and JSON:API are net hacks.

/evrone.

~~Conclusion~~ What I want to discuss at this conference

GraphQL and JSON:API are net hacks.

- Can be replaced with HTTP/2 for some use cases.

PiterPy

~~Conclusion~~ What I want to discuss at this conference

GraphQL and JSON:API are net hacks.

Can be replaced with HTTP/2 for some use cases.

- REST is best with CRUD, but not limited to it.

~~Conclusion~~ What I want to discuss at this conference

GraphQL and JSON:API are net hacks.

Can be replaced with HTTP/2 for some use cases.

REST is best with CRUD, but not limited to it.

- We can mix REST, RPC, gRPC, GraphQL, AMQP.

~~Conclusion~~ What I want to discuss at this conference

GraphQL and JSON:API are net hacks.

    Can be replaced with HTTP/2 for some use cases.

REST is best with CRUD, but not limited to it.

We can mix REST, RPC, gRPC, GraphQL, AMQP.

- Existing environment and business needs matters.

/evrone.

The End

Questions?

PiterPy