Serverless Python

What is it good for?

Mikhail Novikov

Founder & Dev Lead, fstrk.io

@nooovikov

Plan

Plan

1. What is Serverless / FaaS ?

Plan

1. What is Serverless / FaaS ?
2. Life before serverless

Plan

1. What is Serverless / FaaS ?
2. Life before serverless
3. Vanilla way

Plan

1. What is Serverless / FaaS ?
2. Life before serverless
3. Vanilla way
4. Frameworks

Plan

1. What is Serverless / FaaS ?
2. Life before serverless
3. Vanilla way
4. Frameworks
5. Challenges

Plan

1. What is Serverless / FaaS ?
2. Life before serverless
3. Vanilla way
4. Frameworks
5. Challenges
6. Costs

1. What is Serverless / FaaS ?

What is Serverless / FaaS ?

What is Serverless / FaaS ?

- Upload code without provisioning servers

What is Serverless / FaaS ?

- Upload code without provisioning servers
- Pay per processing time, not per idle time

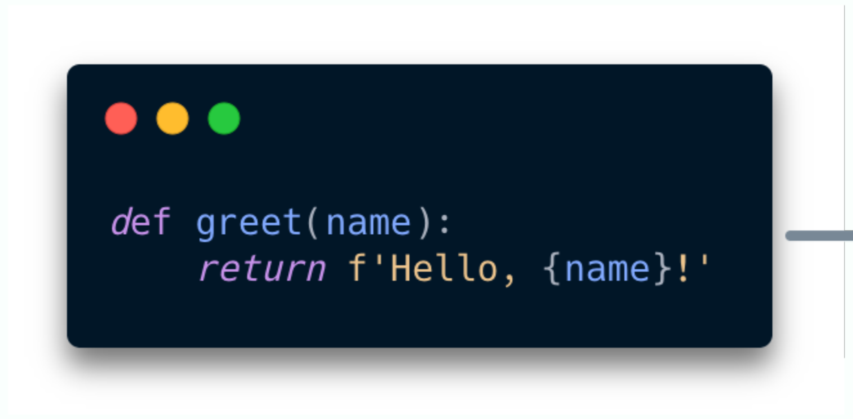
What is Serverless / FaaS ?

- Upload code without provisioning servers
- Pay per processing time, not per idle time
- Use third-party backends (DB, Event bus, Cache, Auth, ...)

What is Serverless / FaaS ?

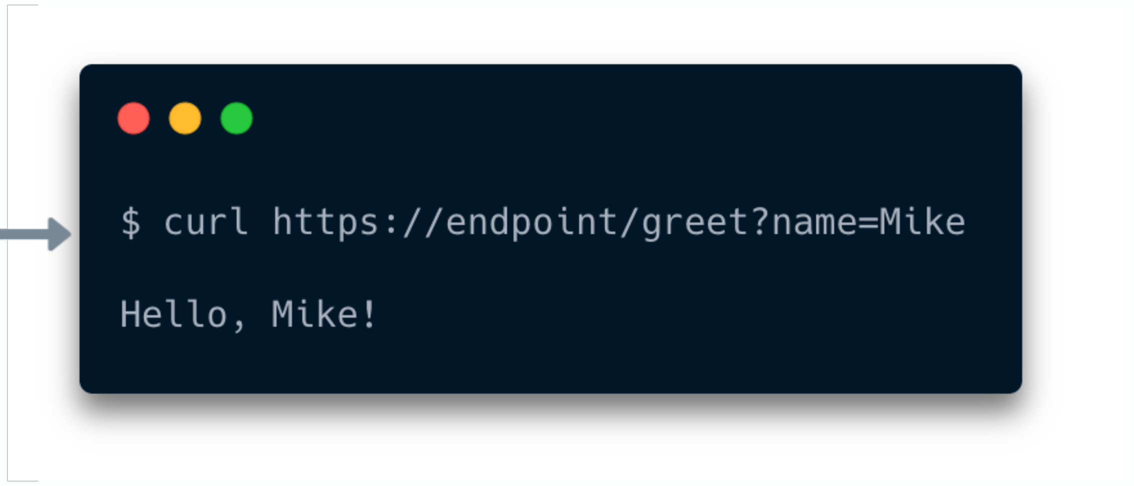
- Upload code without provisioning servers
- Pay per processing time, not per idle time
- Use third-party backends (DB, Event bus, Cache, Auth, ...)
- Focus on your application, not the infrastructure

What is Serverless (really)

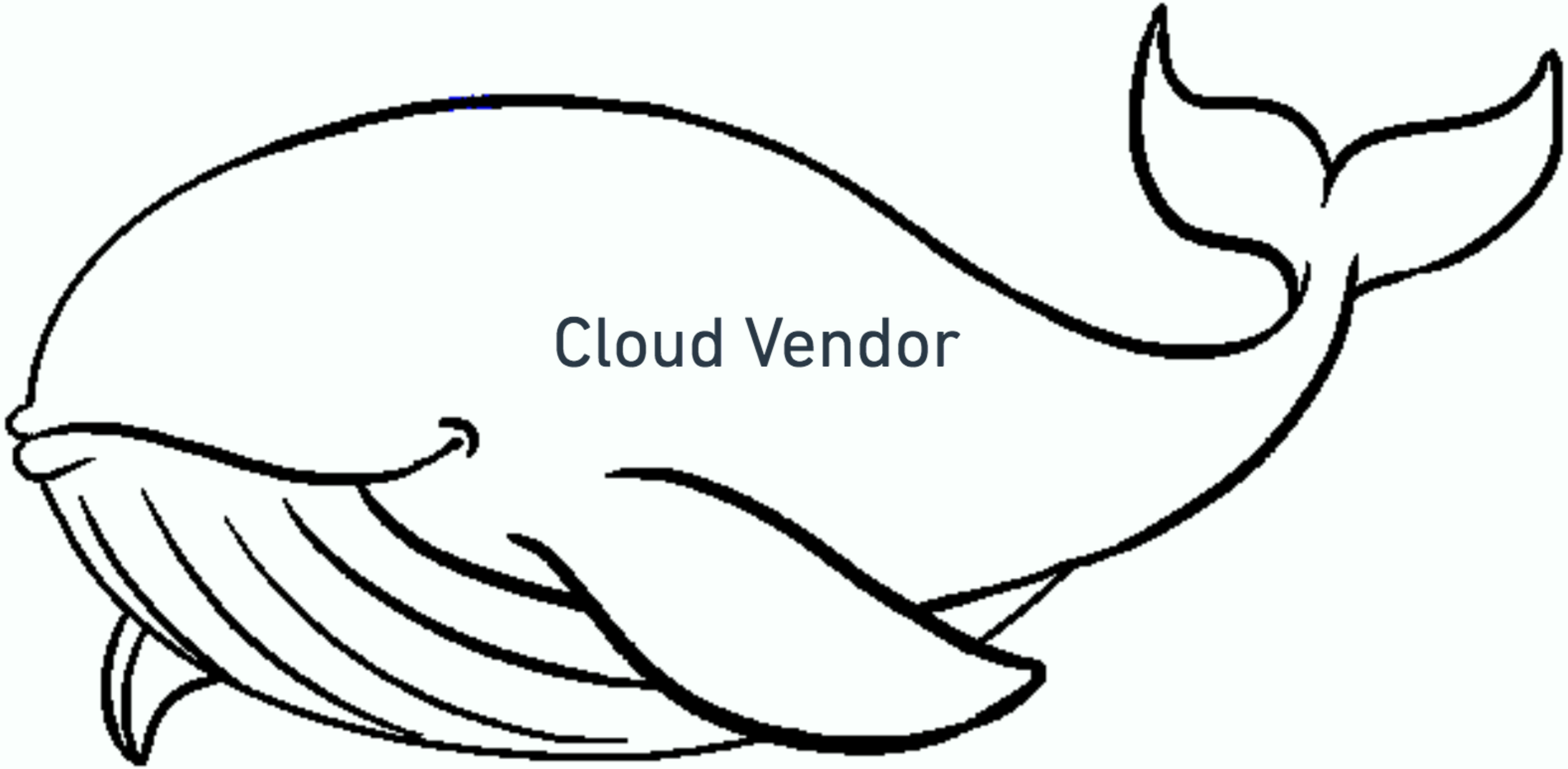


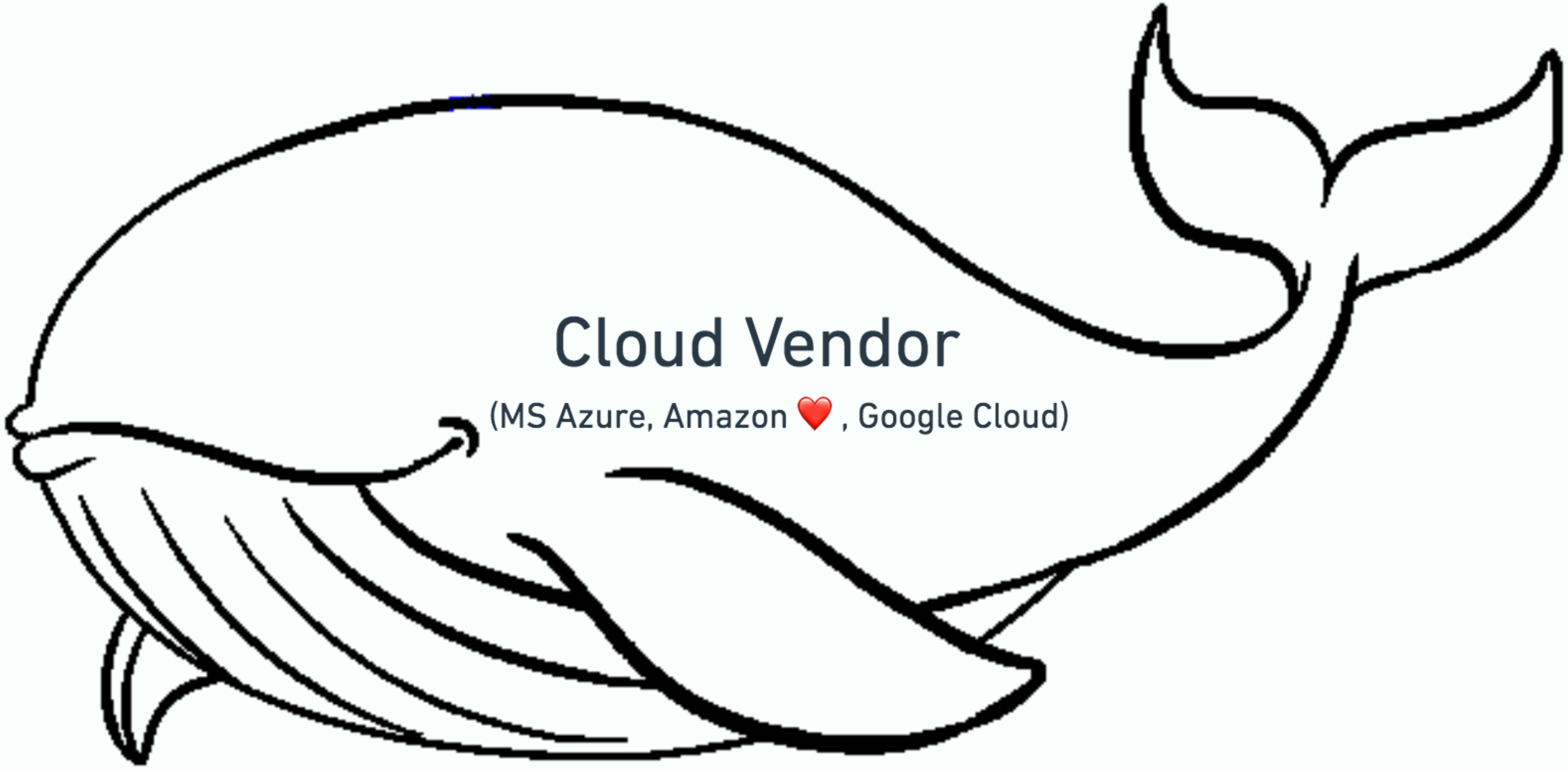
```python
def greet(name):
    return f'Hello, {name}!'
```

Magic



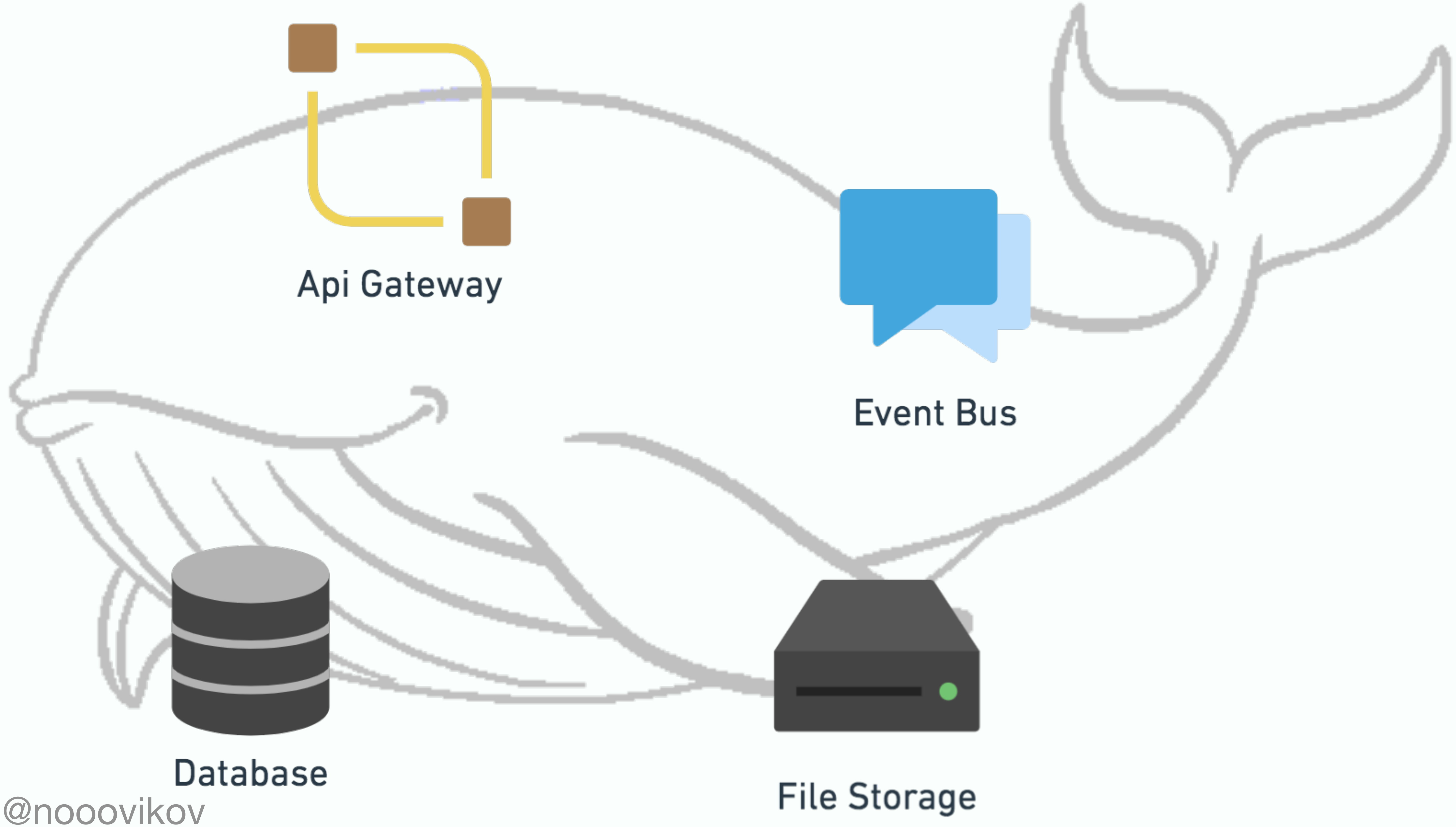
```
$ curl https://endpoint/greet?name=Mike

Hello, Mike!
```

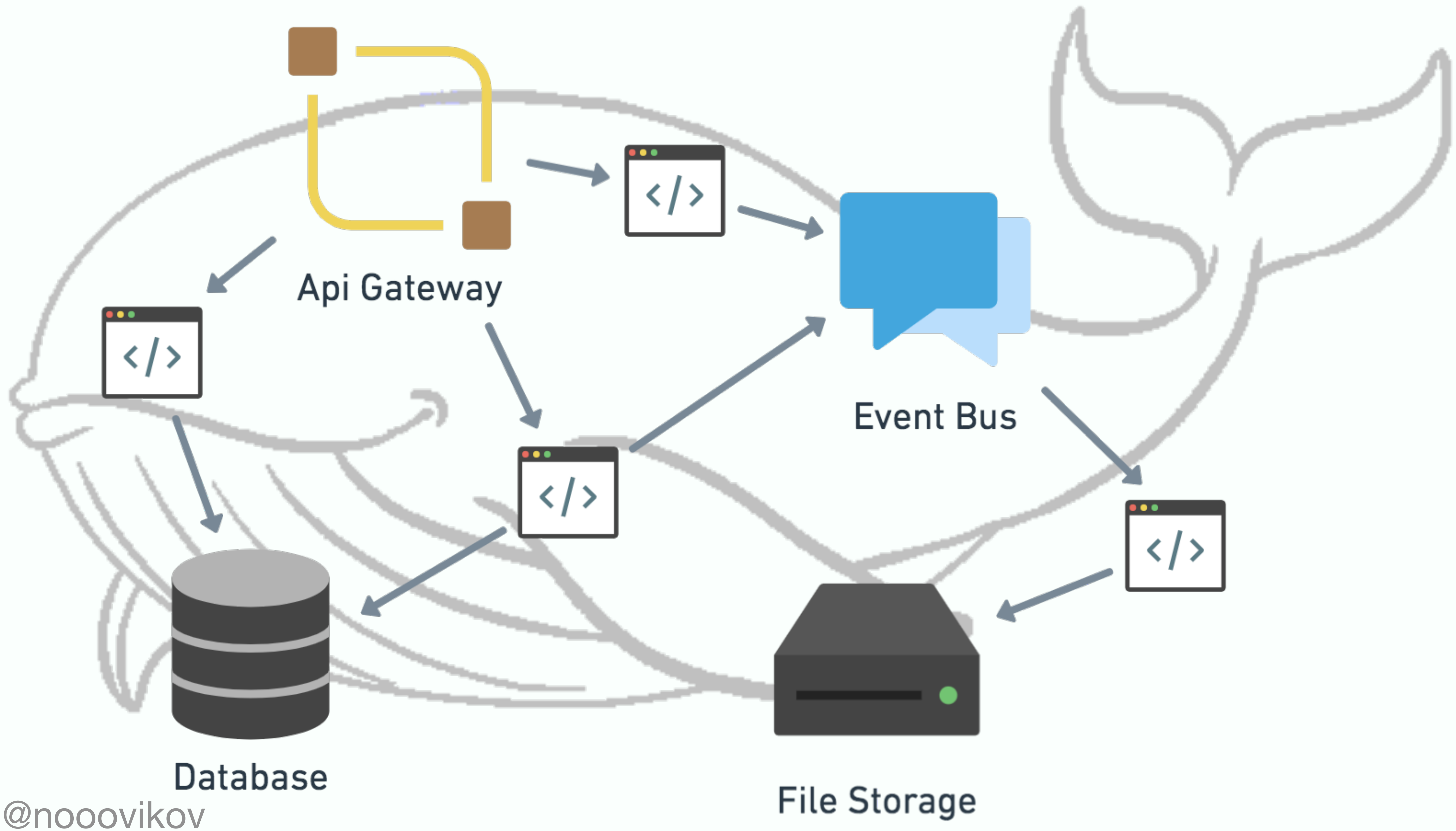

Cloud Vendor



Cloud Vendor

(MS Azure, Amazon ❤️ , Google Cloud)


Api Gateway
Event Bus
Database
File Storage


Api Gateway
Event Bus
Database
File Storage

What is it good for?

What is it good for?

- 📱 Mobile backends & SPAs

What is it good for?

- 📱 Mobile backends & SPAs
- 🔌 APIs & Microservices

What is it good for?

- 📱 Mobile backends & SPAs
- 🔌 APIs & Microservices
- 📦 Data Processing Pipelines

What is it good for?

- 📱 Mobile backends & SPAs
- 🔌 APIs & Microservices
- 📦 Data Processing Pipelines
- ⚡ Webhooks

What is it good for?

- 📱 Mobile backends & SPAs
- 🔌 APIs & Microservices
- 📦 Data Processing Pipelines
- ⚡ Webhooks
- 🤖 Bots and integrations

What is it good for?

- 📱 Mobile backends & SPAs
- 🔌 APIs & Microservices
- 📦 Data Processing Pipelines
- ⚡ Webhooks
- 🤖 Bots and integrations
- 💡 IoT Backends

What is it NOT good for?

What is it NOT good for?

- ⌛ Long tasks

What is it NOT good for?

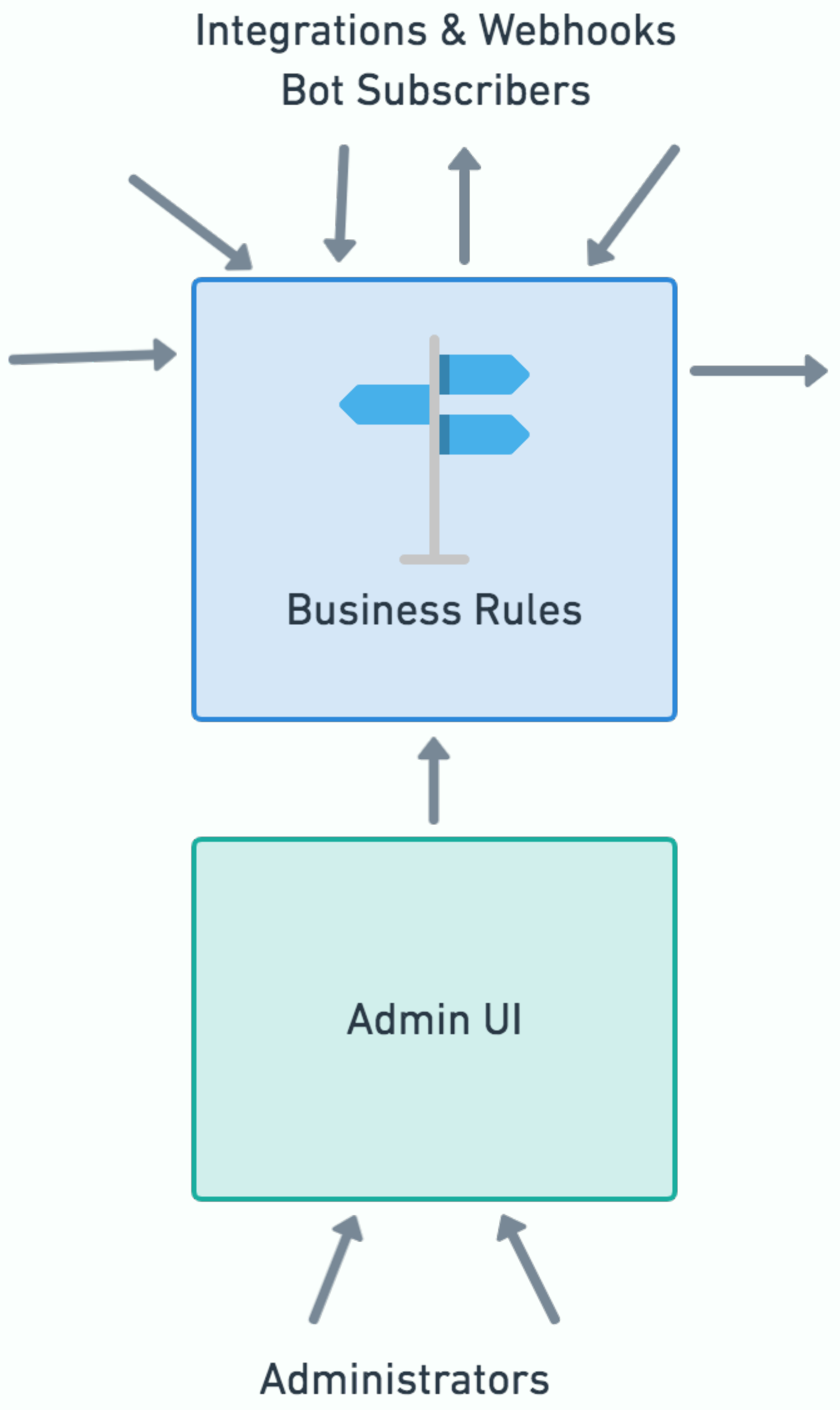
- ⏳ Long tasks
- 🕸️ Apps with complicated dependencies

What is it NOT good for?

- ⏳ Long tasks
- 🕸️ Apps with complicated dependencies
- 💾 Stateful processes

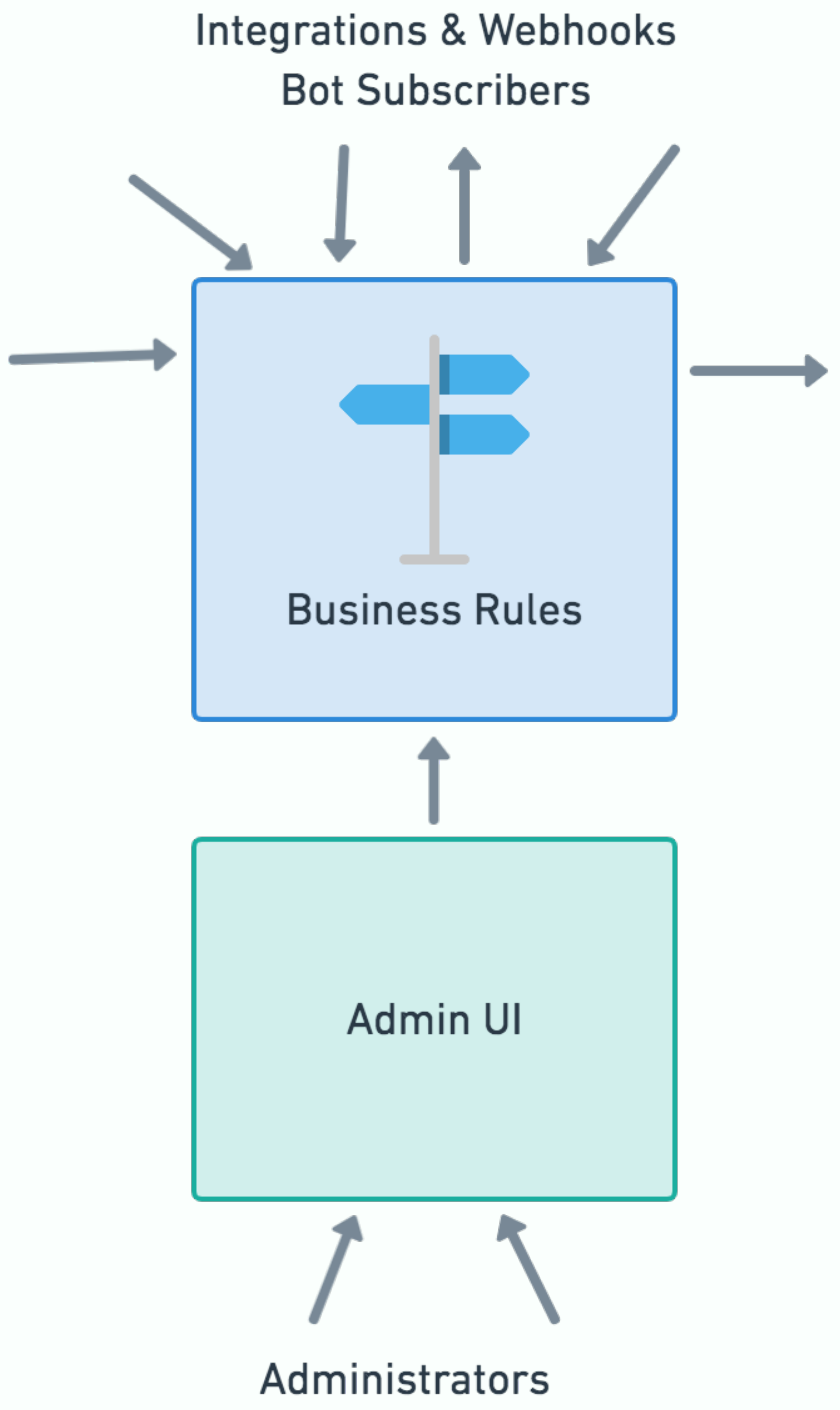
2. Life before Serverless

Our product


Integrations & Webhooks
Bot Subscribers
Business Rules
Admin UI
Administrators

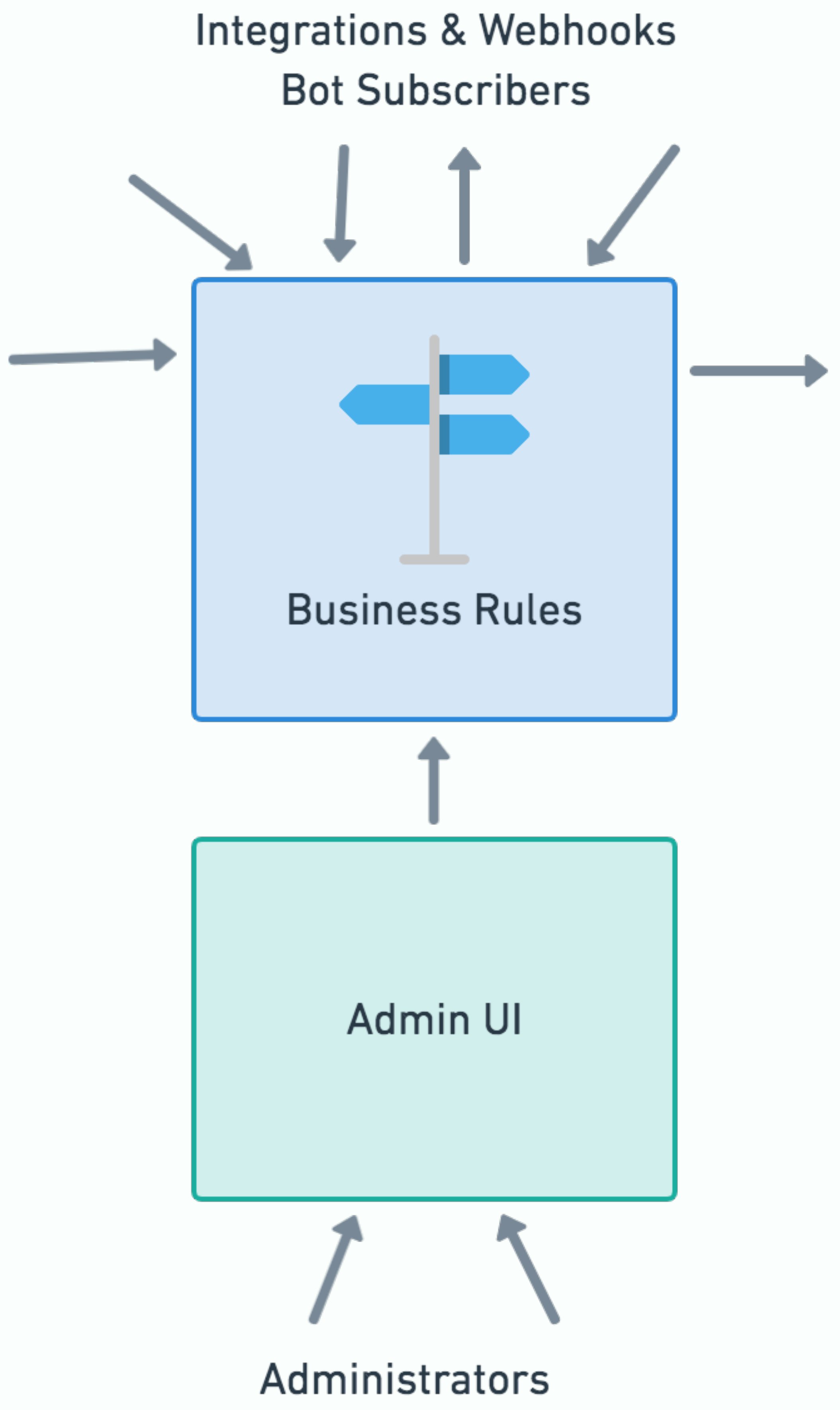
Our product

- Huge monolith, 80k LOC and growing



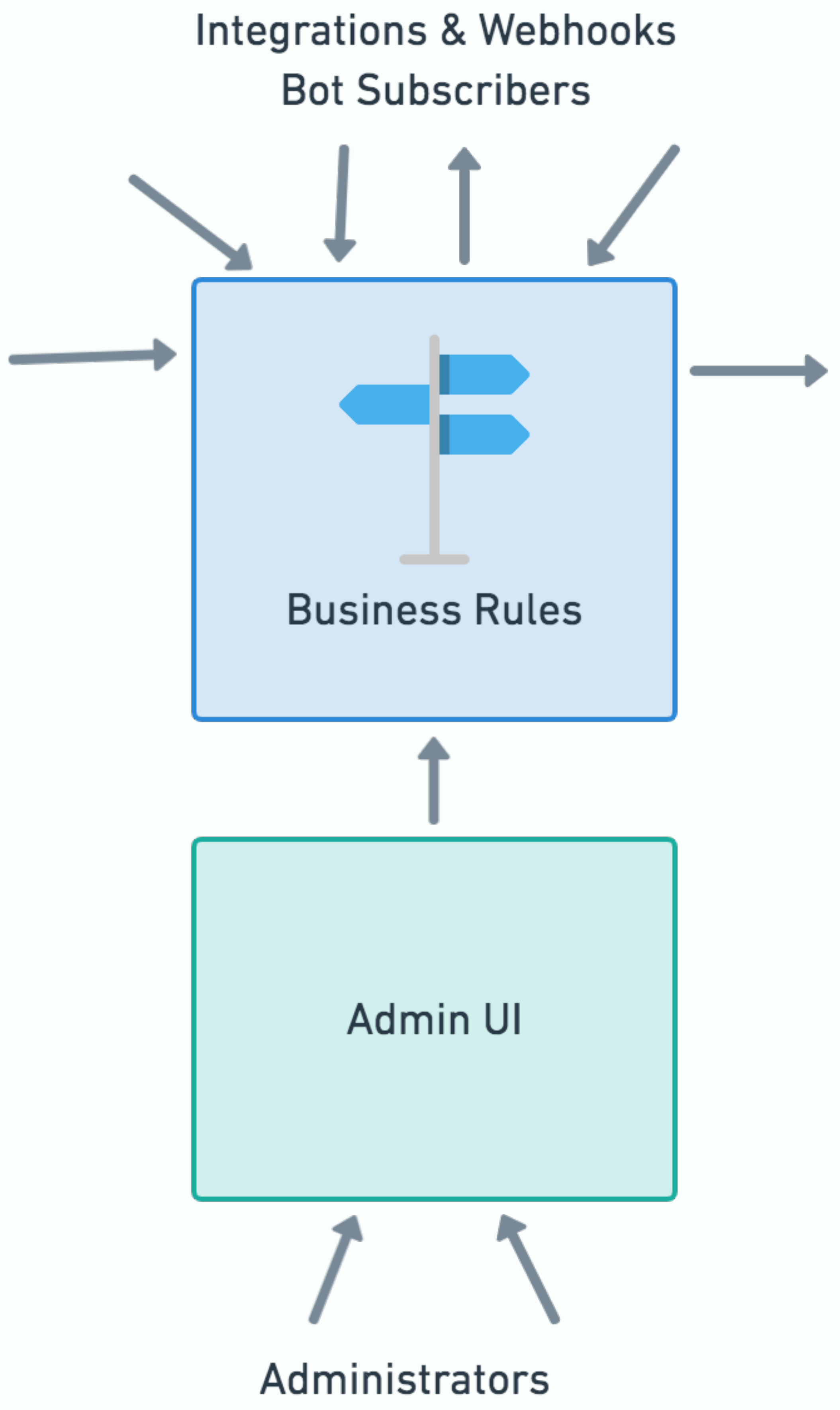
Our product

- Huge monolith, 80k LOC and growing
- Tons of dependencies



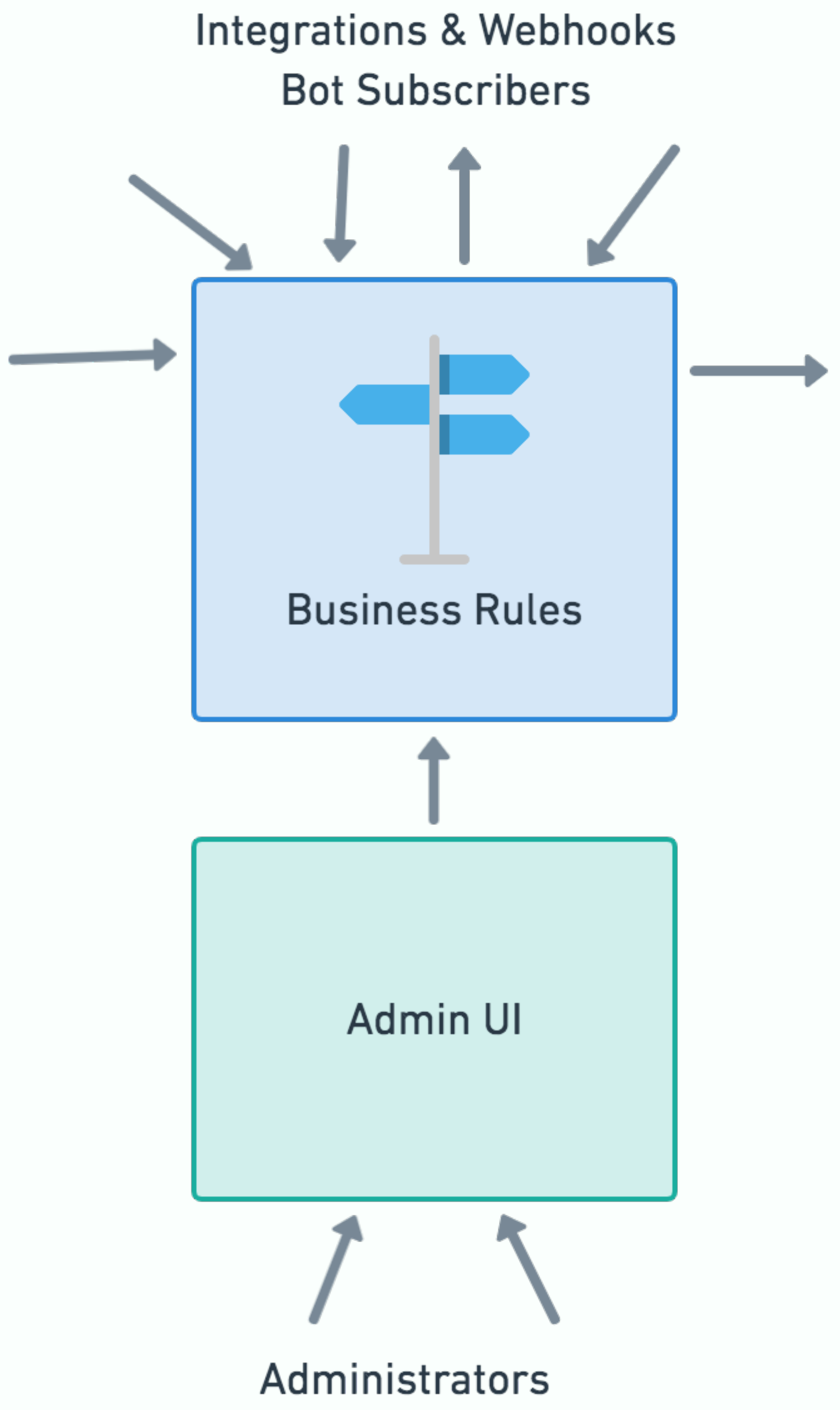
Our product

- Huge monolith, 80k LOC and growing
- Tons of dependencies
- Slow startup time, slow deploy time



Our product

- Huge monolith, 80k LOC and growing
- Tons of dependencies
- Slow startup time, slow deploy time
- One-time customizations, special cases



One-time customizations

One-time customizations

- "Please change your API for us"

One-time customizations

- "Please change your API for us"
- "Please use our SOAP API"

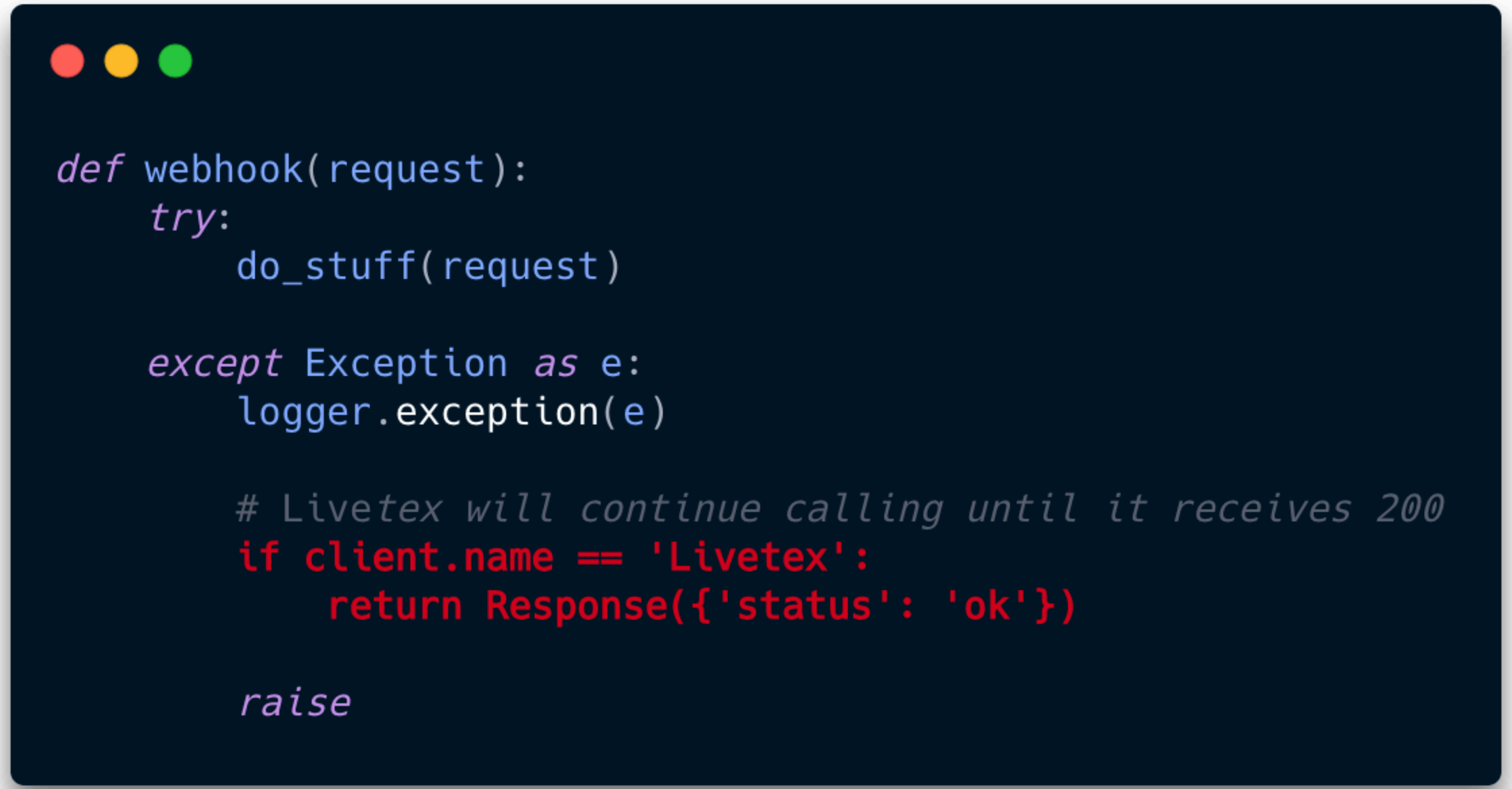
One-time customizations

- "Please change your API for us"
- "Please use our SOAP API"
- "Please call us with this certificate"

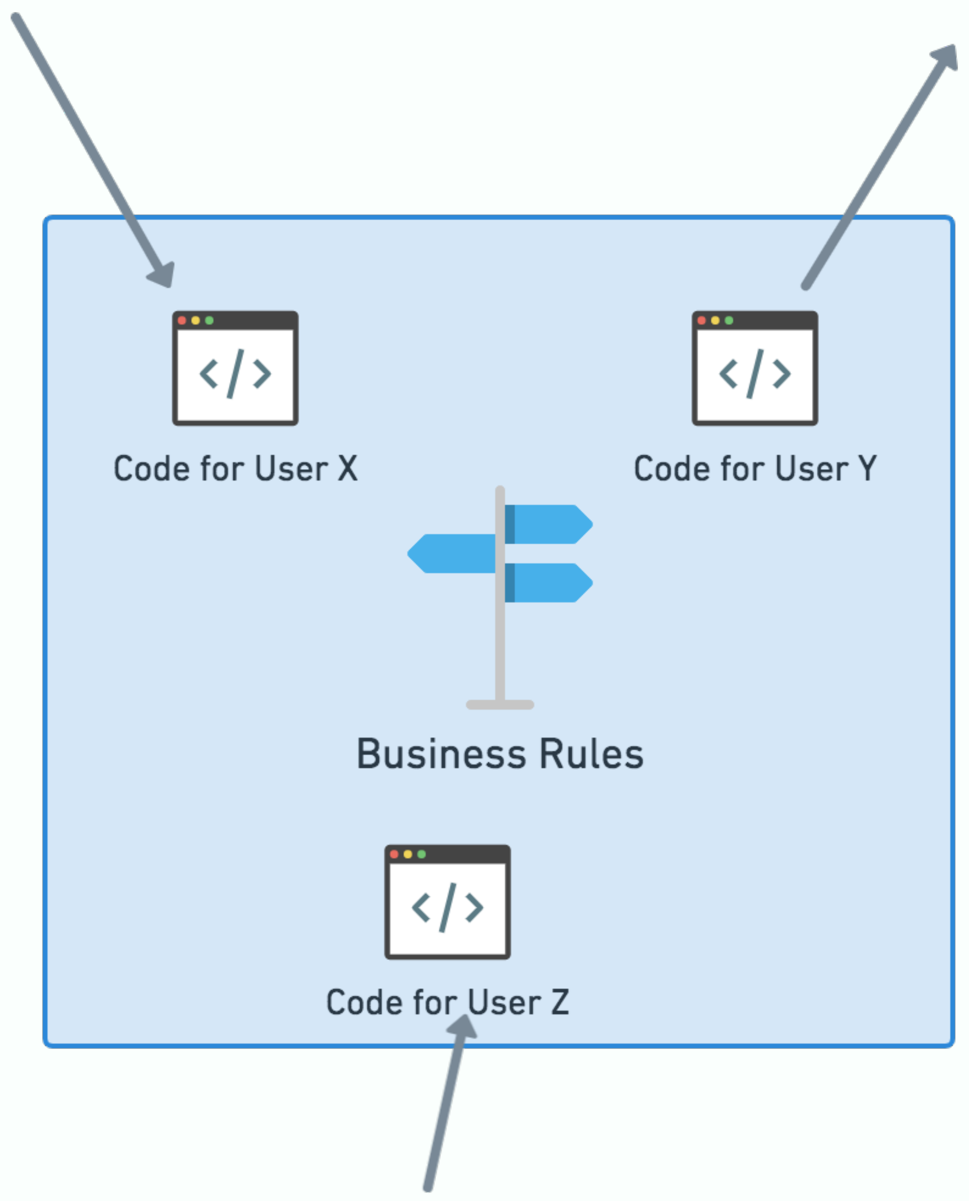
One-time customizations

- "Please change your API for us"
- "Please use our SOAP API"
- "Please call us with this certificate"
- "Please always answer 200"

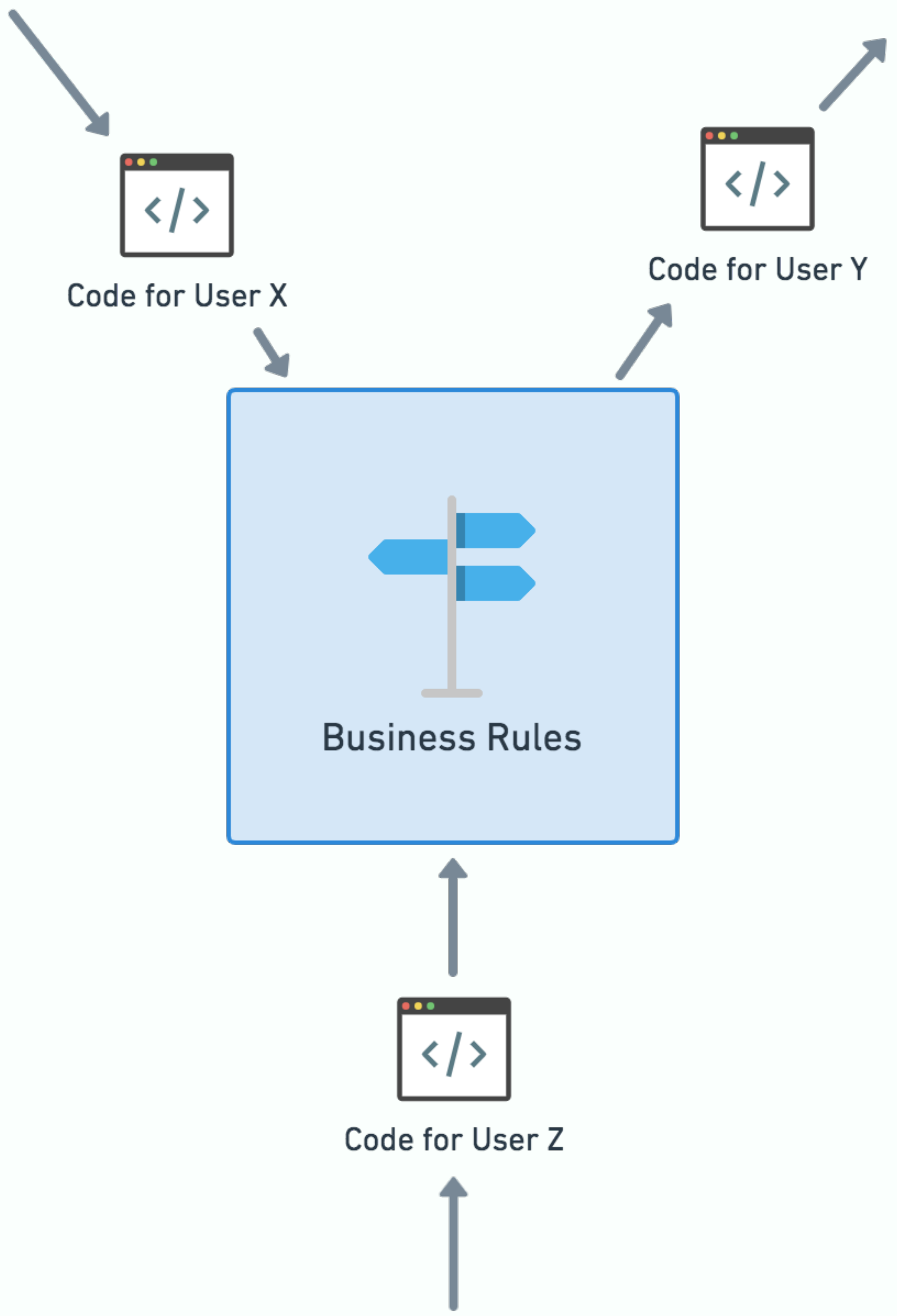
Weird code stuck in master


```
def webhook(request):
    try:
        do_stuff(request)

    except Exception as e:
        logger.exception(e)

        # Livetex will continue calling until it receives 200
        if client.name == 'Livetex':
            return Response({'status': 'ok'})

        raise
```

What we have



What we need



Objective: URL Shortener

Why not use goo.gl / bit.ly / ...?

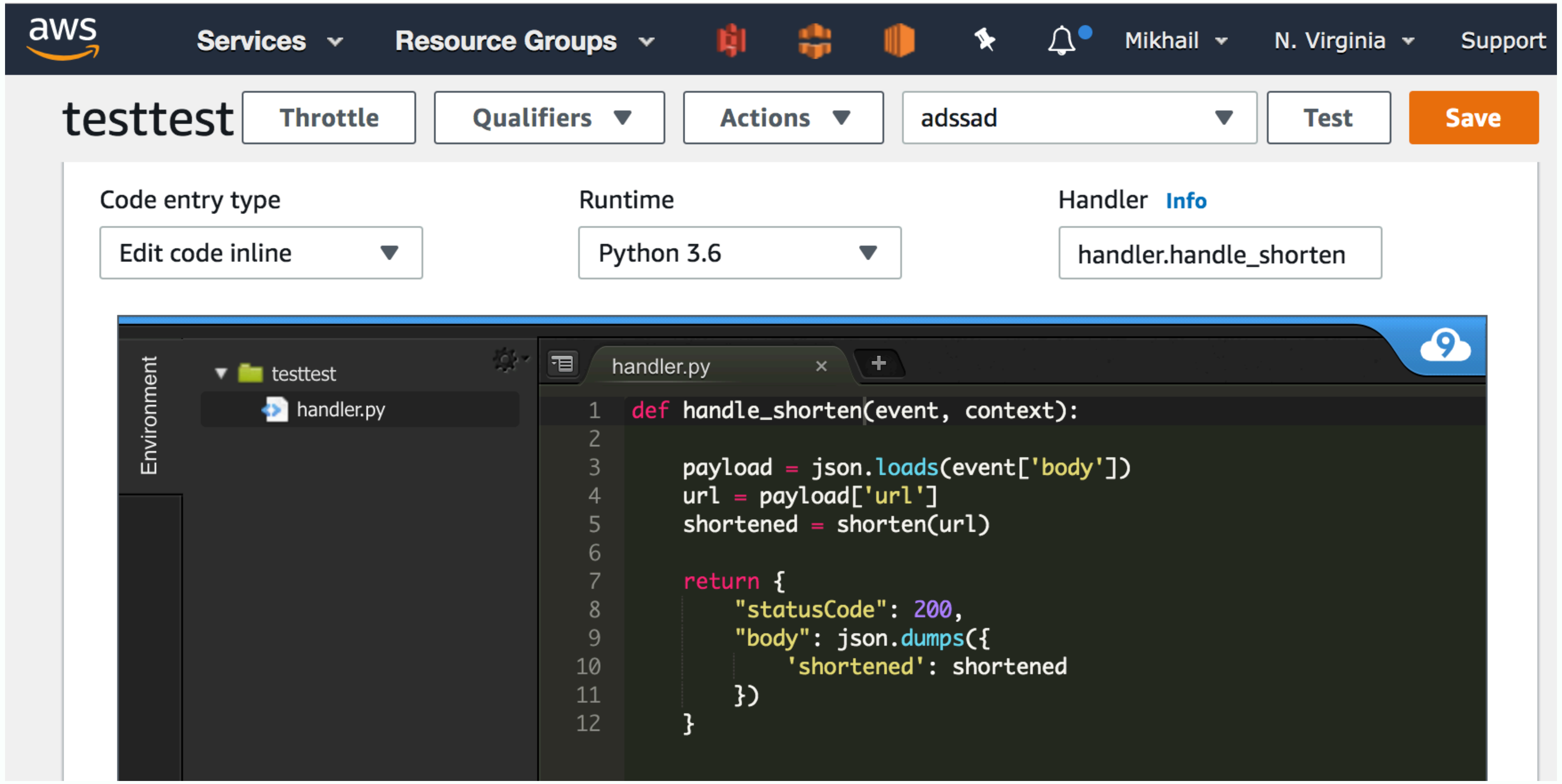
- Shortener APIs are not cheap
- We need callbacks
- We need whitelabelling

Features

- POST: save url in DB, return code
- GET: find code in DB, fire callback, redirect to full url

3. First steps

Step 1: Write a function



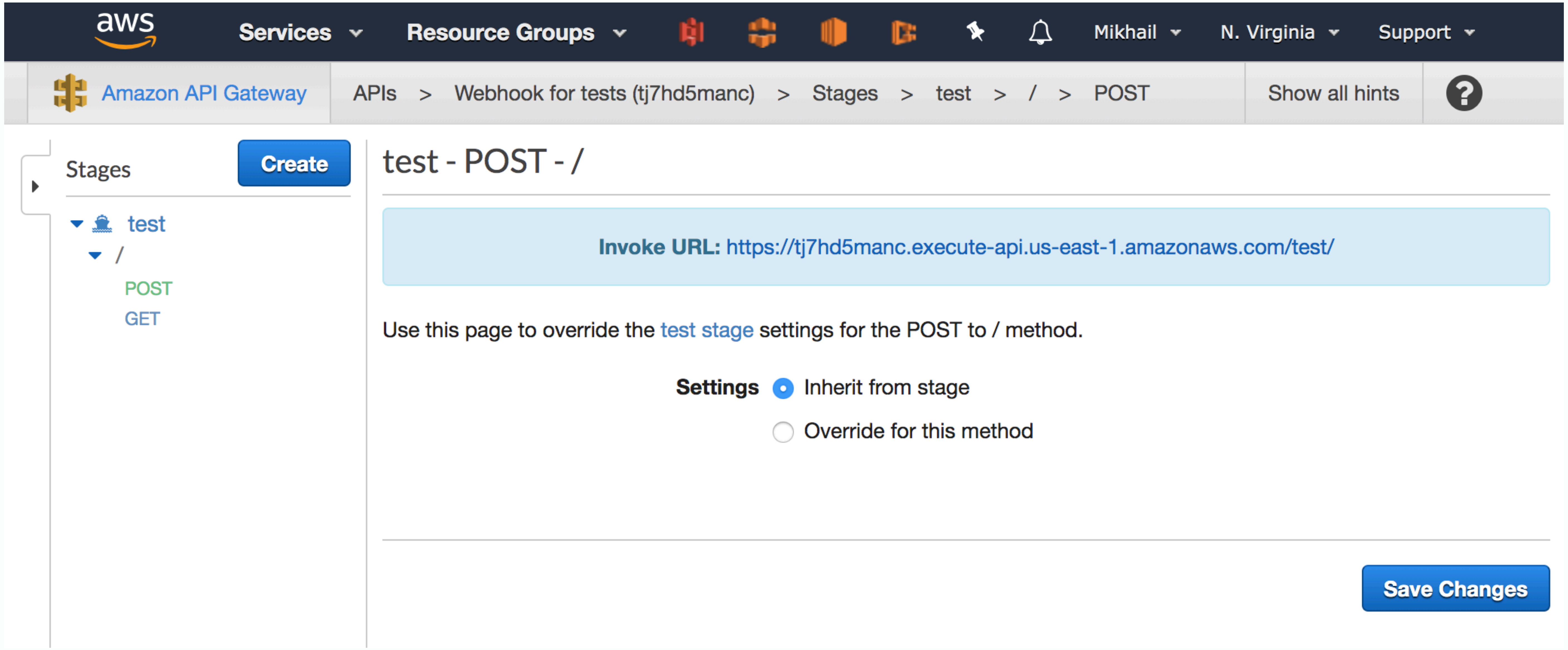
Code entry type: Edit code inline

Runtime: Python 3.6

Handler Info: handler.handle_shorten

```python
def handle_shorten(event, context):

    payload = json.loads(event['body'])
    url = payload['url']
    shortened = shorten(url)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'shortened': shortened
        })
    }
```

Step 2: Add an API



test - POST - /

Invoke URL: https://tj7hd5manc.execute-api.us-east-1.amazonaws.com/test/

Use this page to override the test stage settings for the POST to / method.

Settings [✓ Inherit from stage] Override for this method

Is it scalable?

Is it scalable?

- No!

Is it scalable?

- No!
- Online editor

Is it scalable?

- No!
- Online editor
- No version control

Is it scalable?

- No!
- Online editor
- No version control
- No rollbacks

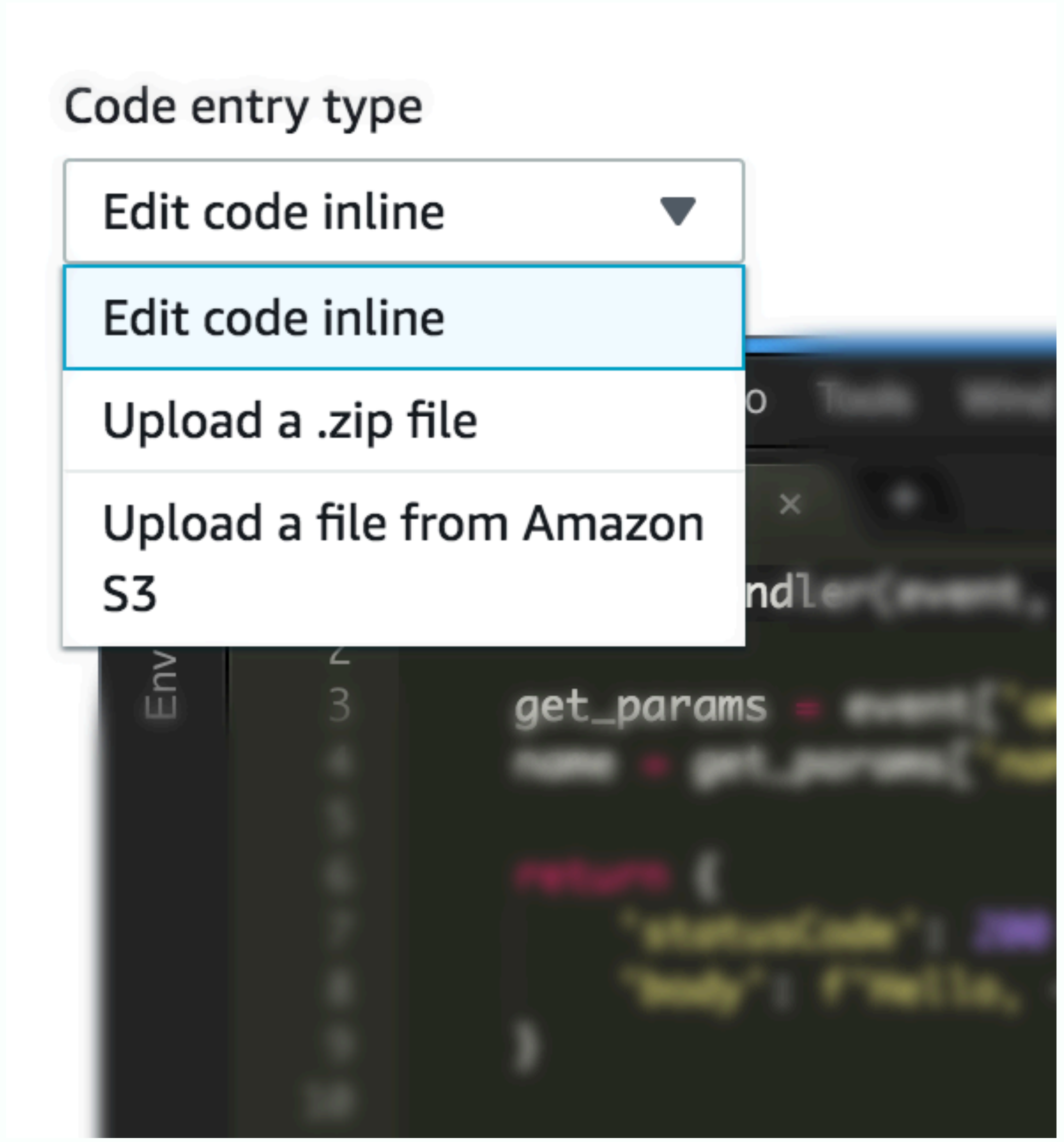
Is it scalable?

- No!
- Online editor
- No version control
- No rollbacks
- No virtualenv

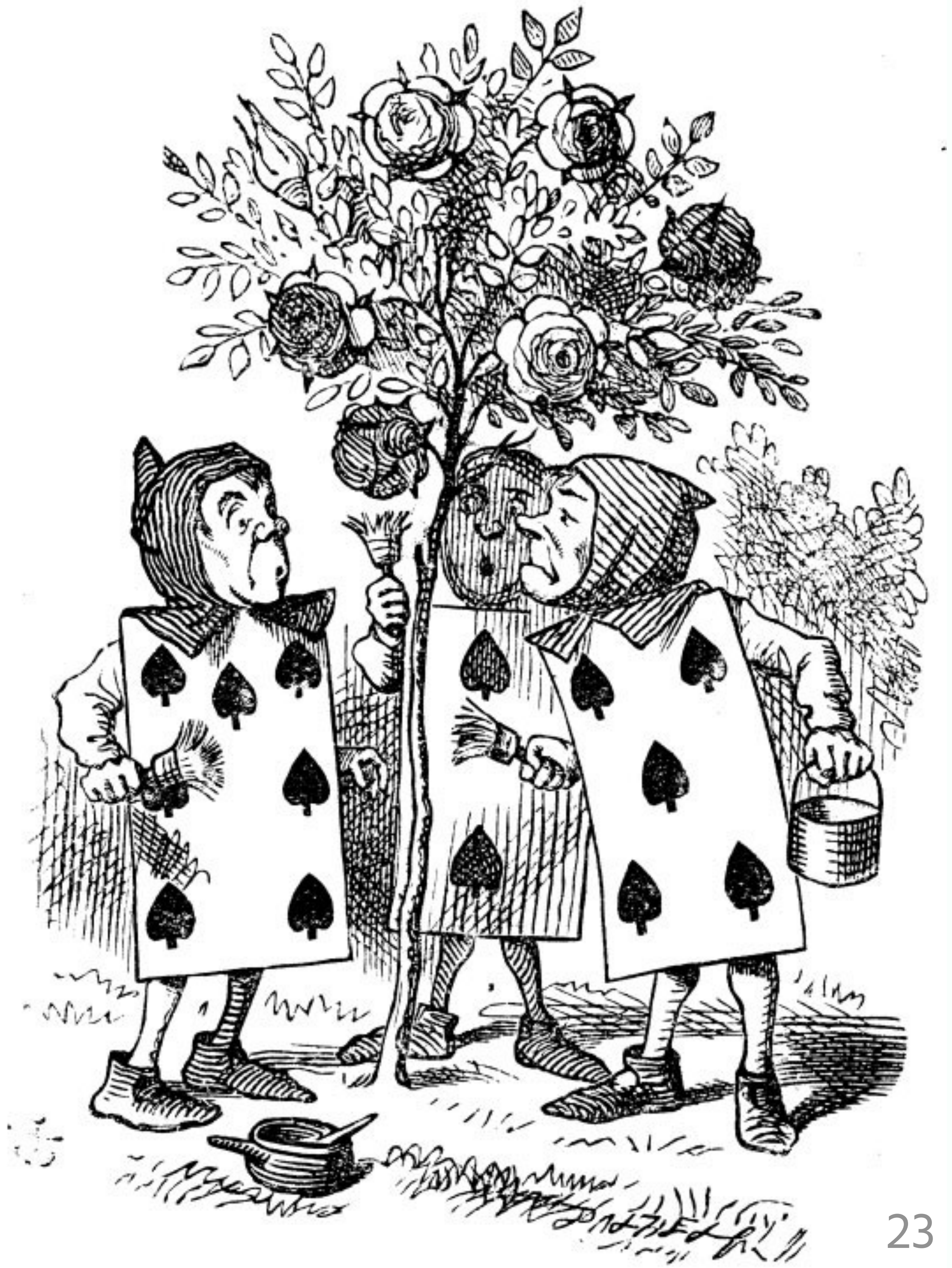
Is it scalable?

No!

- Online editor
- No version control
- No rollbacks
- No virtualenv
- Manual upload with dependencies via ZIP file? Really?



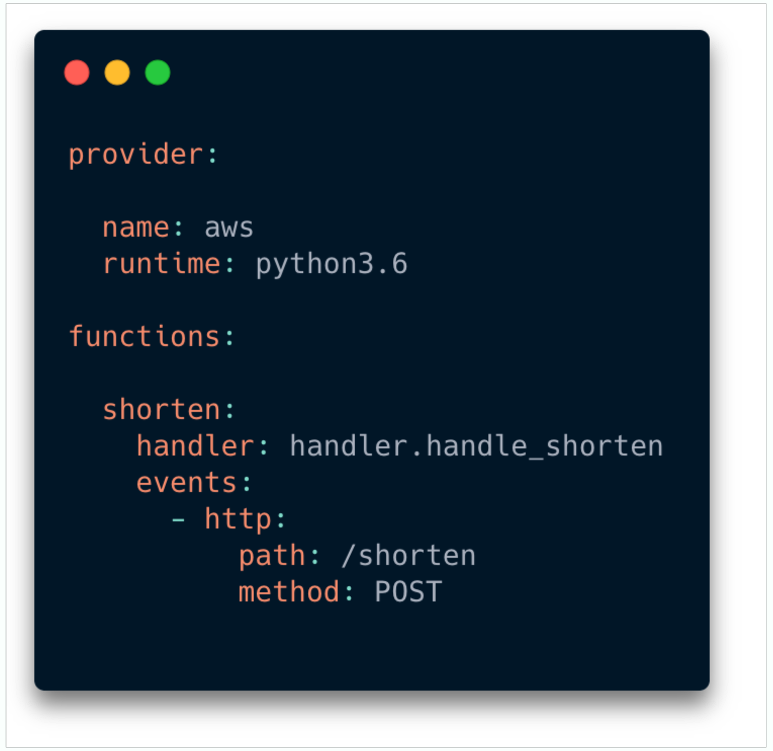
4. Frameworks



4.1. Serverless Framework

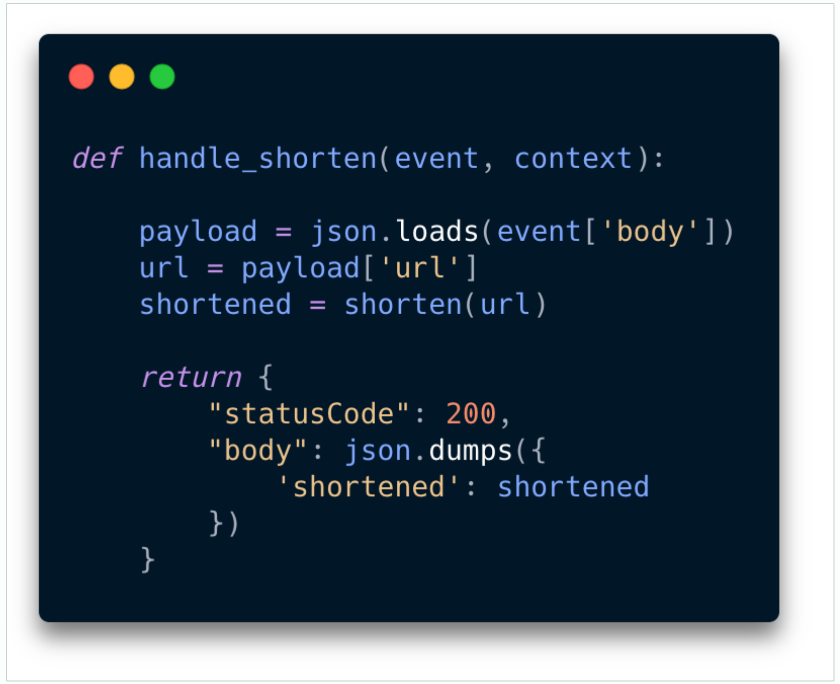
github.com/serverless/serverless

serverless.yml



```yaml
provider:

  name: aws
  runtime: python3.6

functions:

  shorten:
    handler: handler.handle_shorten
    events:
      - http:
          path: /shorten
          method: POST
```

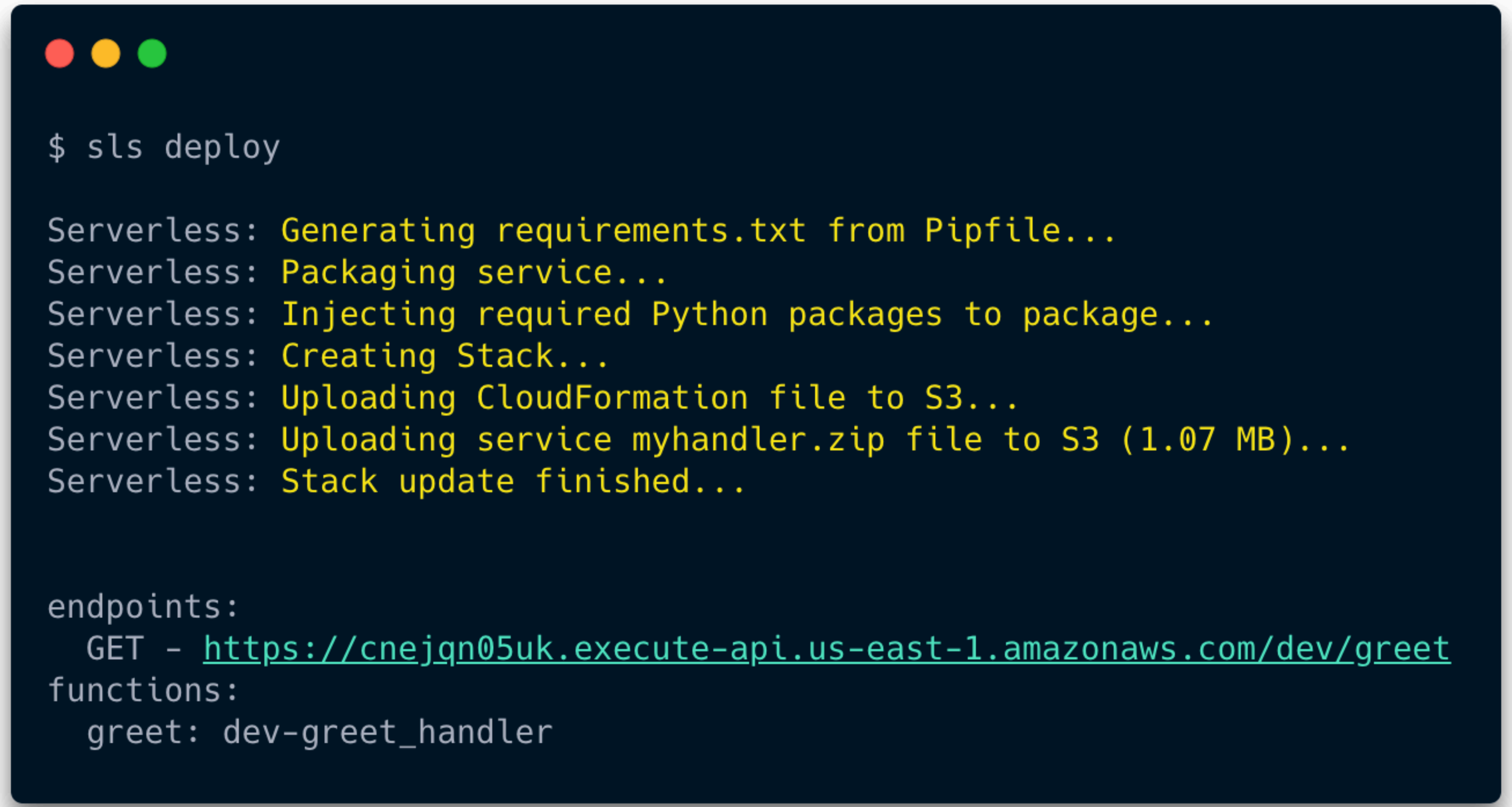
handler.py



```python
def handle_shorten(event, context):

    payload = json.loads(event['body'])
    url = payload['url']
    shortened = shorten(url)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'shortened': shortened
        })
    }
```

Serverless Framework

- YML config + Python code
- Easy deploy
- Tons of settings
- Tons of docs & examples
- Has Enterprise Version! 💰

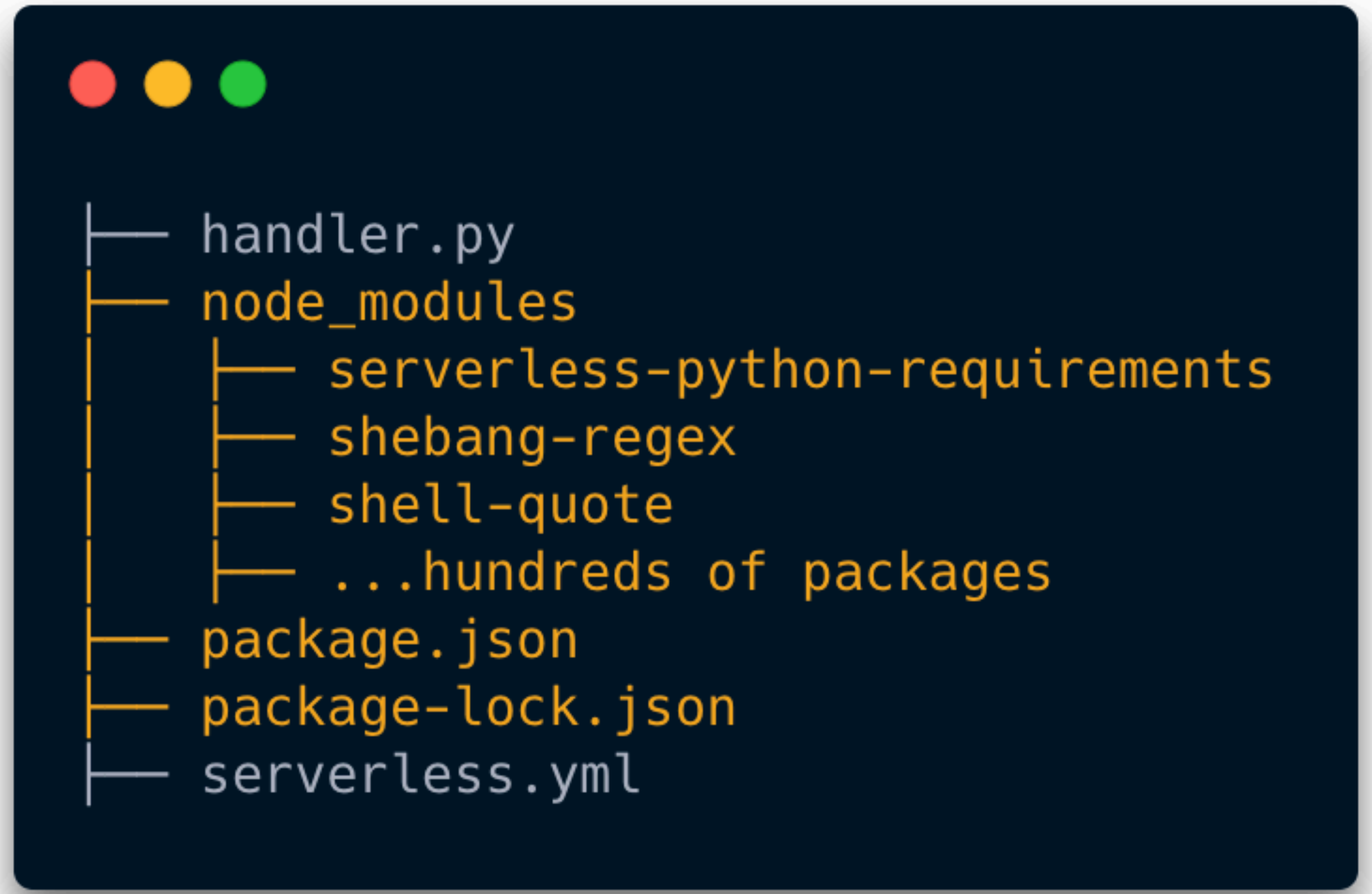


```
$ sls deploy

Serverless: Generating requirements.txt from Pipfile...
Serverless: Packaging service...
Serverless: Injecting required Python packages to package...
Serverless: Creating Stack...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service myhandler.zip file to S3 (1.07 MB)...
Serverless: Stack update finished...


endpoints:
  GET - https://cnejqn05uk.execute-api.us-east-1.amazonaws.com/dev/greet
functions:
  greet: dev-greet_handler
```

But...

We'll need NPM.

```
npm install serverless
npm install serverless-python-requirements
```

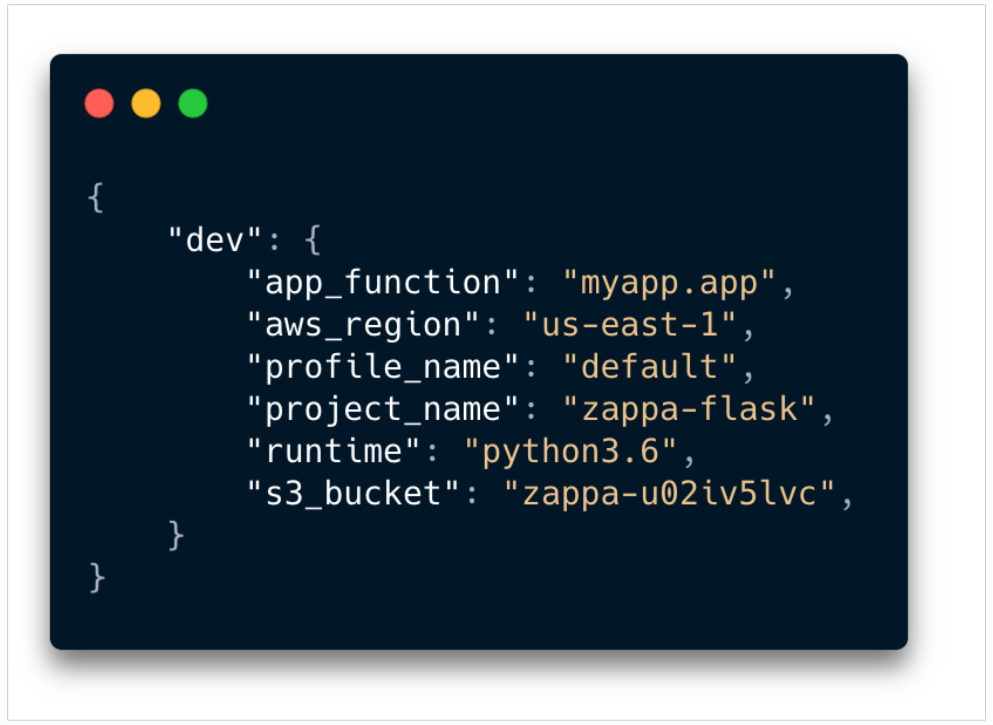


```
├── handler.py
├── node_modules
│    ├── serverless-python-requirements
│    ├── shebang-regex
│    ├── shell-quote
│    ├── ...hundreds of packages
├── package.json
├── package-lock.json
├── serverless.yml
```

4.2. Zappa

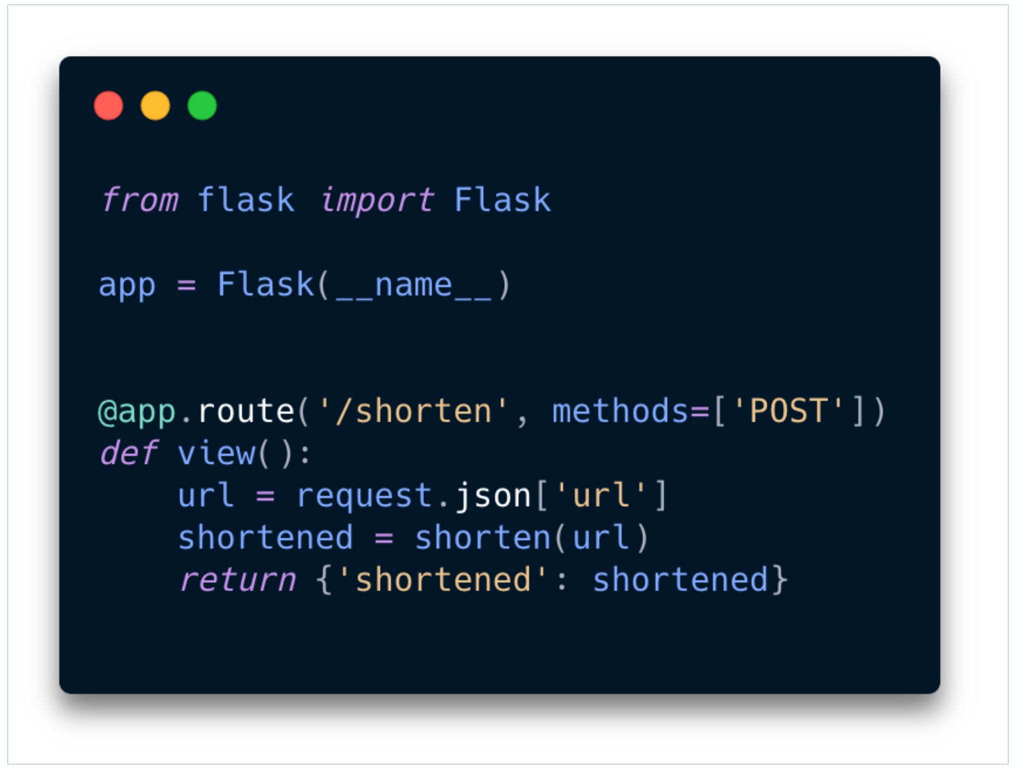
github.com/Miserlou/Zappa

zappa_settings.json



```json
{
    "dev": {
        "app_function": "myapp.app",
        "aws_region": "us-east-1",
        "profile_name": "default",
        "project_name": "zappa-flask",
        "runtime": "python3.6",
        "s3_bucket": "zappa-u02iv5lvc",
    }
}
```

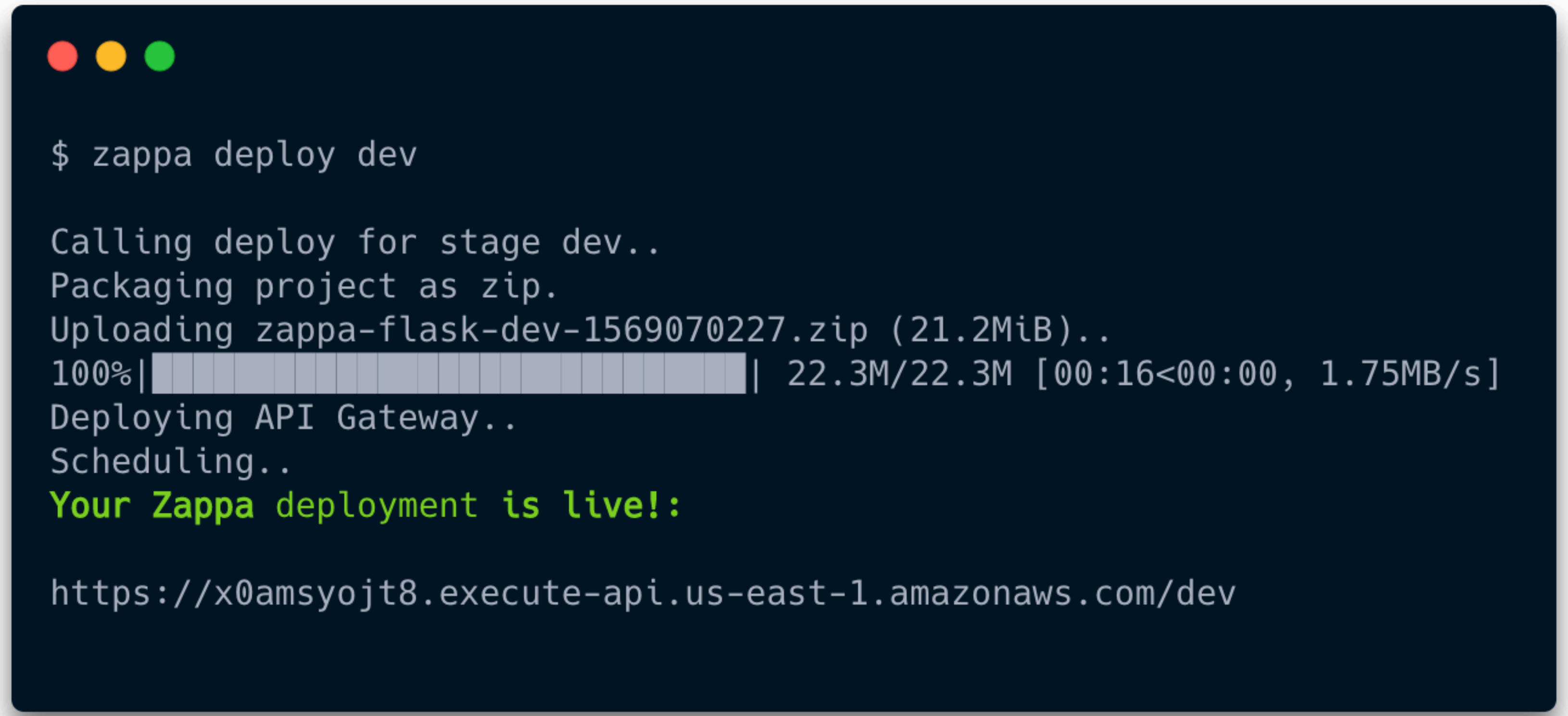
myapp.py



```python
from flask import Flask

app = Flask(__name__)


@app.route('/shorten', methods=['POST'])
def view():
    url = request.json['url']
    shortened = shorten(url)
    return {'shortened': shortened}
```

Zappa

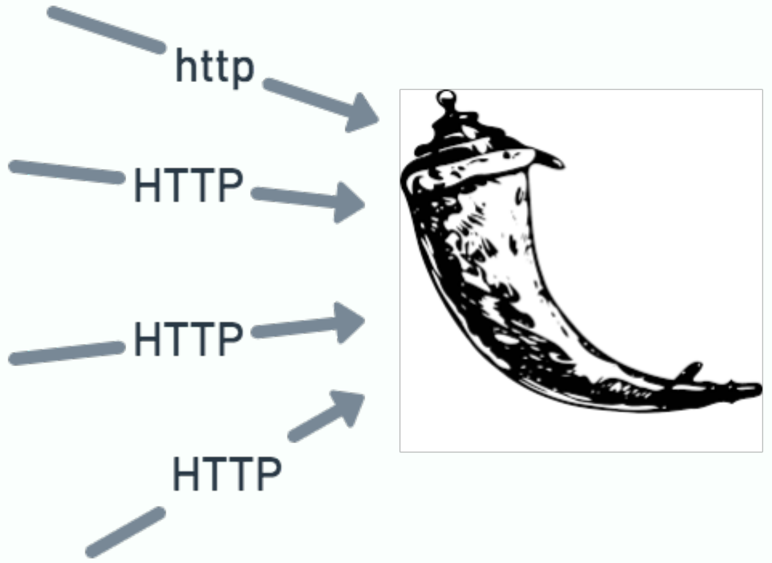
- Supports Django & Flask
- One extra dependency + **`zappa_settings.json`**
- Deploy existing projects with (almost) zero config
- Migrate pet projects to Lambda!



```
$ zappa deploy dev

Calling deploy for stage dev..
Packaging project as zip.
Uploading zappa-flask-dev-1569070227.zip (21.2MiB)..
100%|████████████████████████████| 22.3M/22.3M [00:16<00:00, 1.75MB/s]
Deploying API Gateway..
Scheduling..
Your Zappa deployment is live!:

https://x0amsyojt8.execute-api.us-east-1.amazonaws.com/dev
```

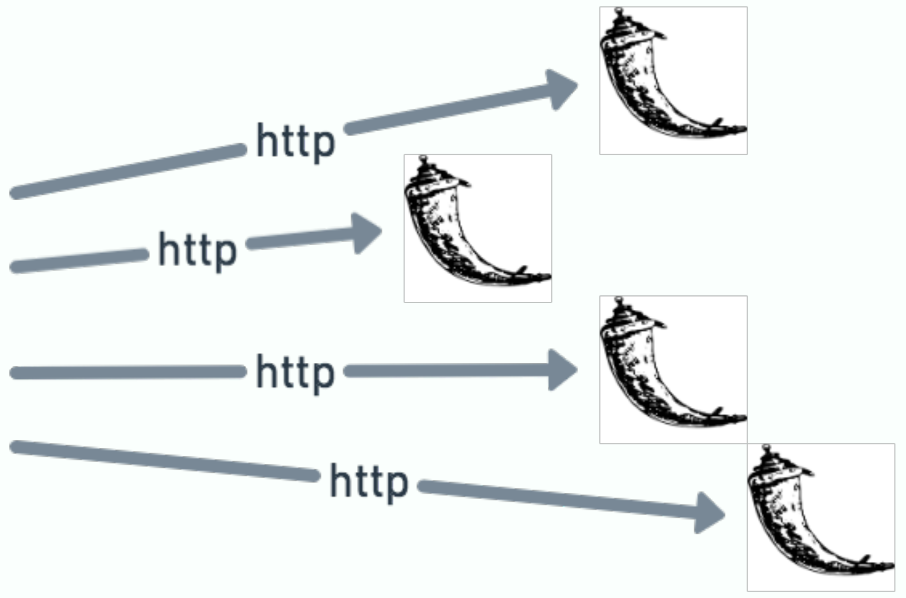
Flask

All requests are processed by the same server



Zappa

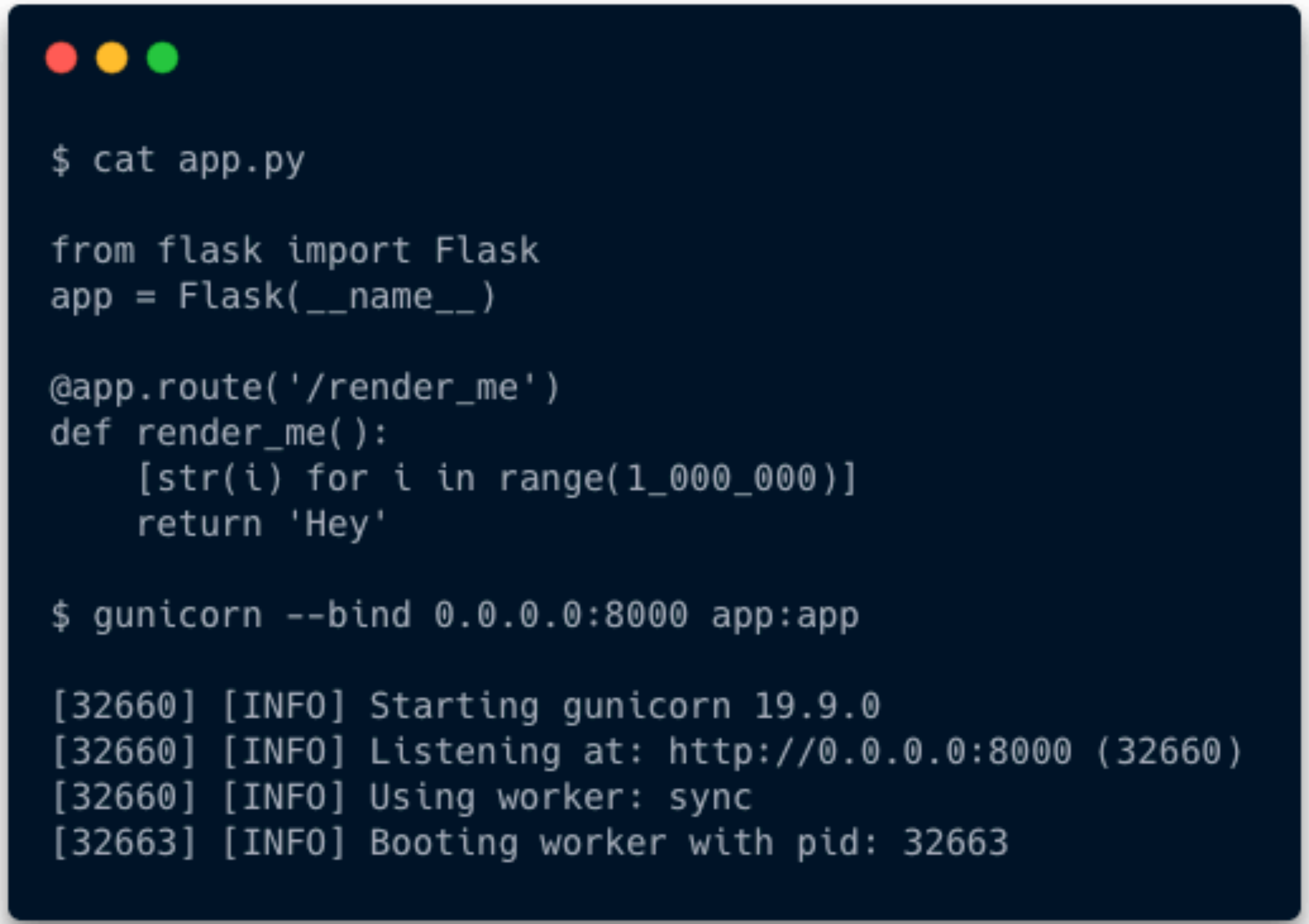
Every request deserves its own server!



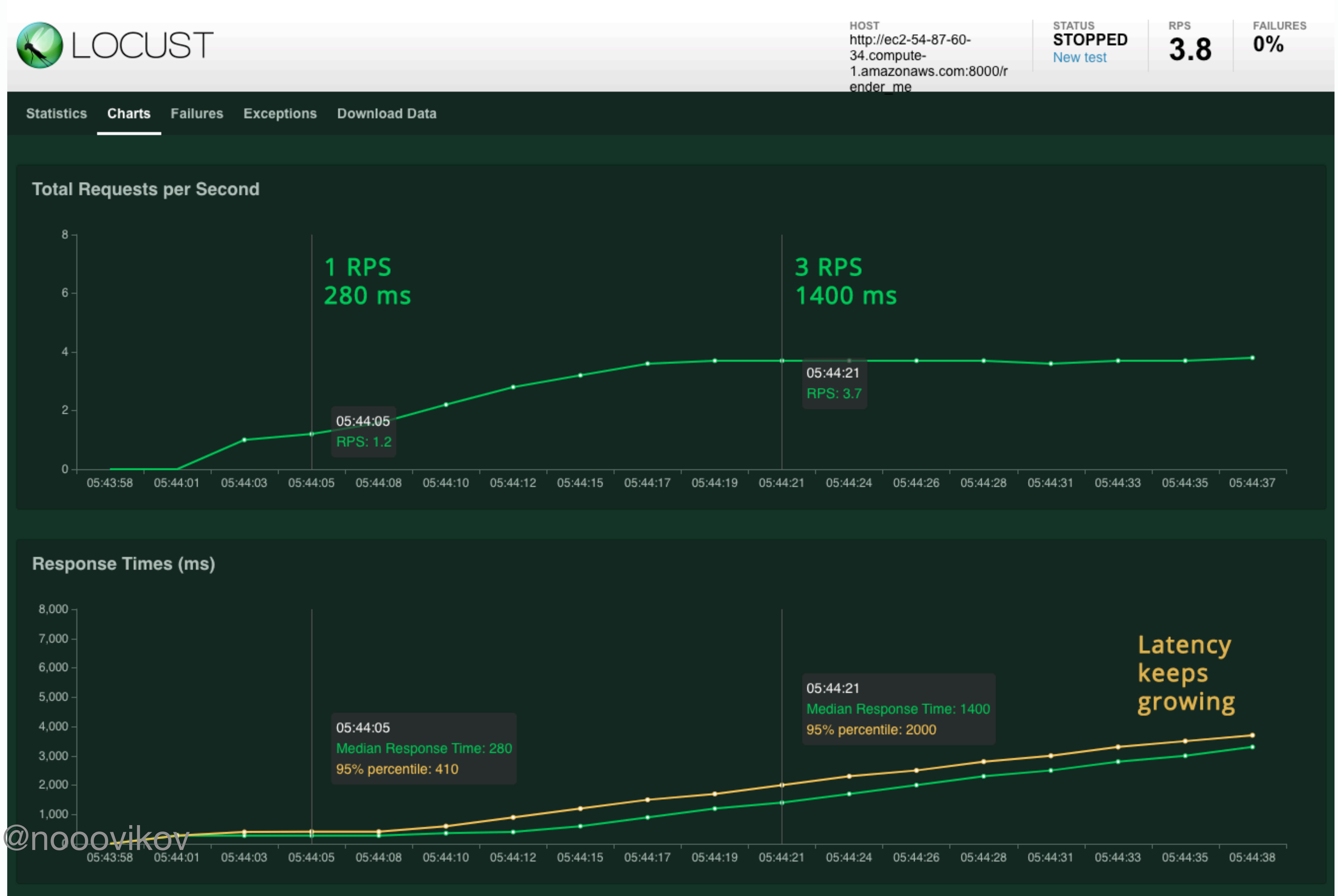
How about load?

Let's test it!

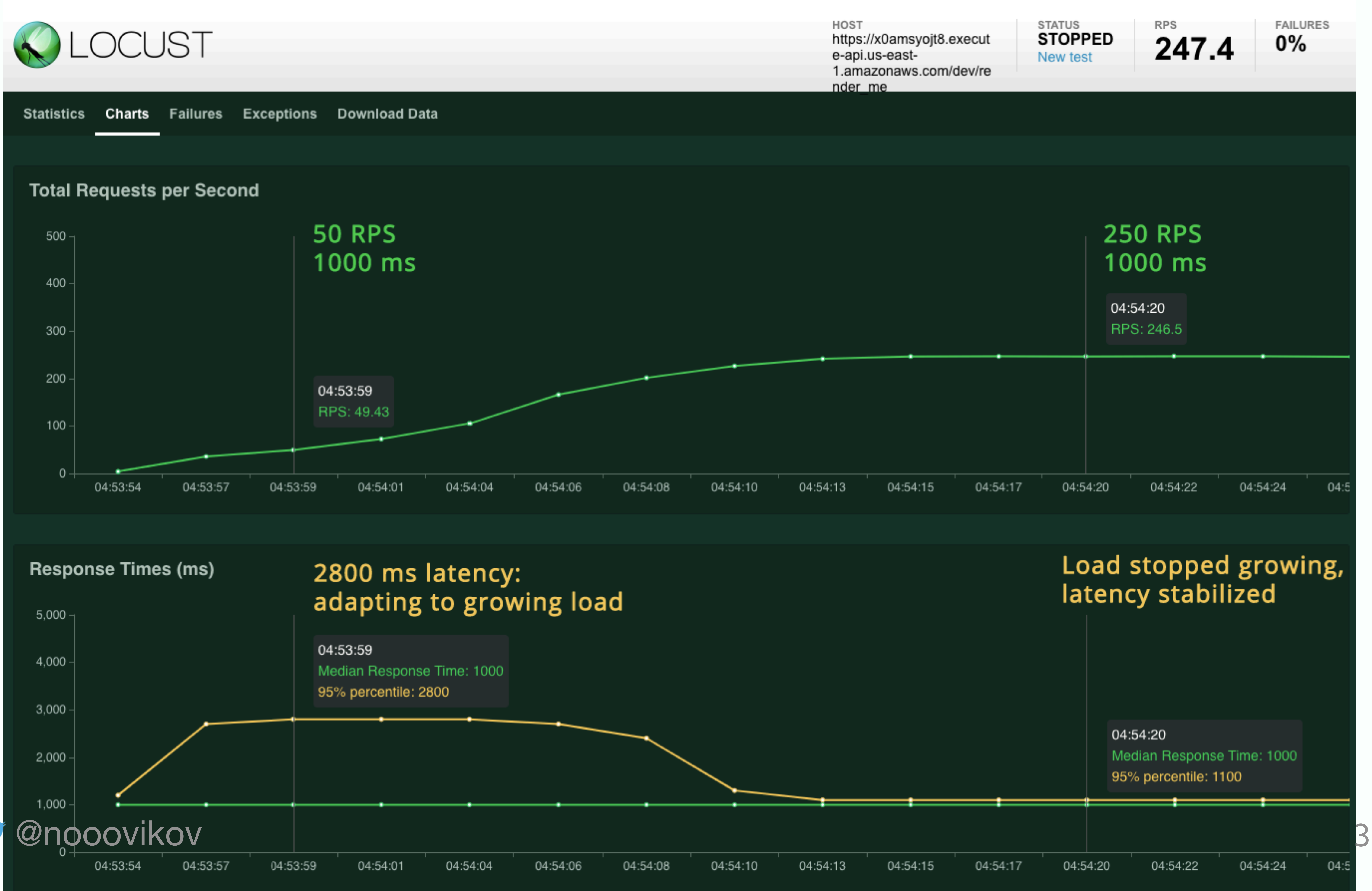
- We'll create a slow handler
- Deploy it to a 512MB EC2 server
- Deploy it to a 512MB Lambda
- And test both with **Locust**



```
$ cat app.py

from flask import Flask
app = Flask(__name__)

@app.route('/render_me')
def render_me():
    [str(i) for i in range(1_000_000)]
    return 'Hey'

$ gunicorn --bind 0.0.0.0:8000 app:app

[32660] [INFO] Starting gunicorn 19.9.0
[32660] [INFO] Listening at: http://0.0.0.0:8000 (32660)
[32660] [INFO] Using worker: sync
[32663] [INFO] Booting worker with pid: 32663
```


LOCUST
HOST
http://ec2-54-87-60-34.compute-1.amazonaws.com:8000/render_me
STATUS
STOPPED
New test
RPS
3.8
FAILURES
0%
Statistics
Charts
Failures
Exceptions
Download Data
Total Requests per Second
1 RPS
280 ms
3 RPS
1400 ms
05:44:05
RPS: 1.2
05:44:21
RPS: 3.7
Response Times (ms)
05:44:05
Median Response Time: 280
95% percentile: 410
05:44:21
Median Response Time: 1400
95% percentile: 2000
Latency keeps growing

@nooovikov


LOCUST

HOST https://x0amsyojt8.execute-api.us-east-1.amazonaws.com/dev/render_me
STATUS STOPPED New test
RPS 247.4
FAILURES 0%

Statistics Charts Failures Exceptions Download Data

Total Requests per Second

50 RPS
1000 ms

250 RPS
1000 ms

04:53:59
RPS: 49.43

04:54:20
RPS: 246.5

Response Times (ms)

2800 ms latency:
adapting to growing load

Load stopped growing,
latency stabilized

04:53:59
Median Response Time: 1000
95% percentile: 2800

04:54:20
Median Response Time: 1000
95% percentile: 1100

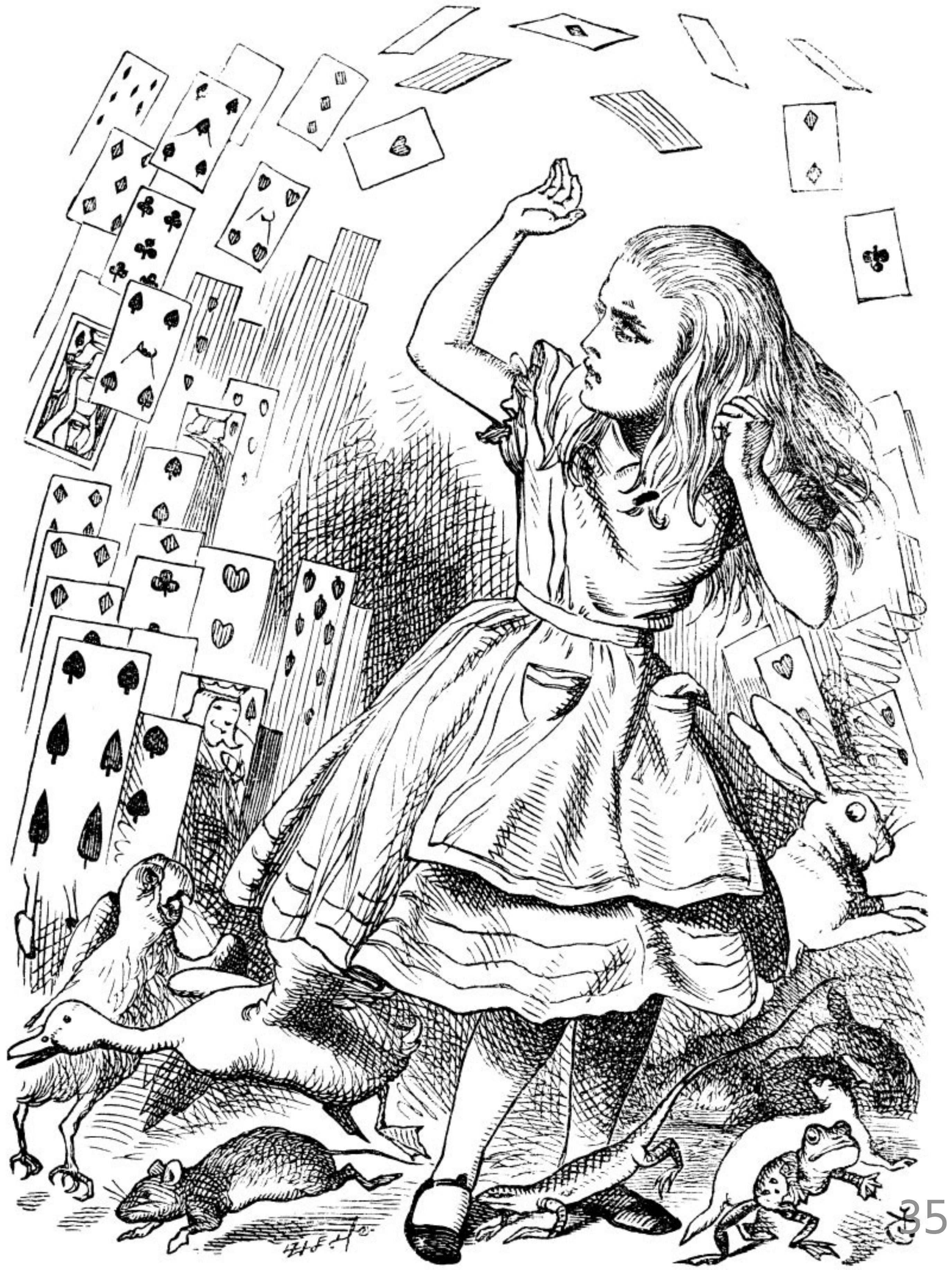
Test results

- Lambda is **extremely scalable**
  - ***Low load:*** Lambda @ 1000 ms, server @ 300 ms
  - ***High load:*** Lambda @ 1000 ms, server dies
- Lambda is **greedy**
  - If you want better latency (300 ms) - give it more resources
  - (If you don't do useless loops - it's pretty fast)
- Lambda **adapts to load**
  - ***Cold start:*** startup time + processing time
  - ***Hot start:*** processing time

We have conquered Serverless!

...Or have we?

5. Challenges

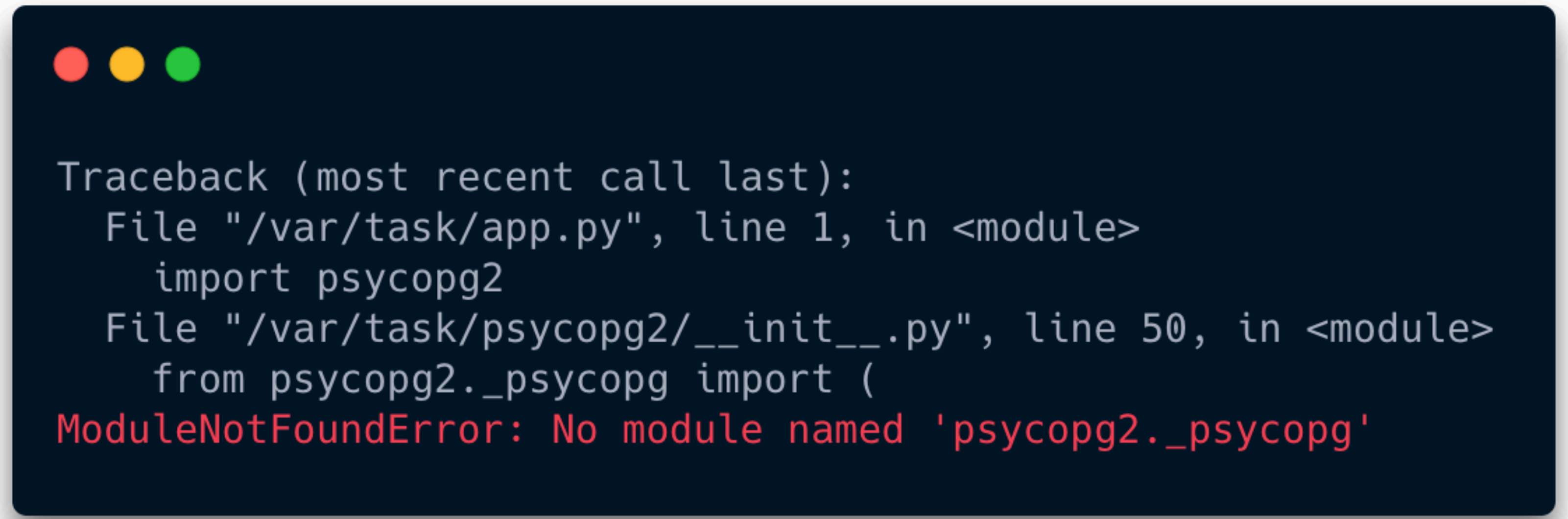


First problem

- We need a DB for our urls. Let's see how Lambda handles Postgres!
- `pip install psycopg2`
- `zappa deploy dev`

First problem

- We need a DB for our urls. Let's see how Lambda handles Postgres!
- `pip install psycopg2`
- `zappa deploy dev`
- ...Crashes with code 500



```
Traceback (most recent call last):
  File "/var/task/app.py", line 1, in <module>
    import psycopg2
  File "/var/task/psycopg2/__init__.py", line 50, in <module>
    from psycopg2._psycopg import (
ModuleNotFoundError: No module named 'psycopg2._psycopg'
```

5.1. Binaries

5.1. Binaries

- No **`pip install`** on Lambda

5.1. Binaries

- No **`pip install`** on Lambda
- Zappa uploads all dependencies from localhost

5.1. Binaries

- No **`pip install`** on Lambda
- Zappa uploads all dependencies from localhost
- My **`psycopg2`** was compiled for **Mac OS**

5.1. Binaries

- No **`pip install`** on Lambda
- Zappa uploads all dependencies from localhost
- My **`psycopg2`** was compiled for **Mac OS**
- Lambda runs on **Linux AMI**

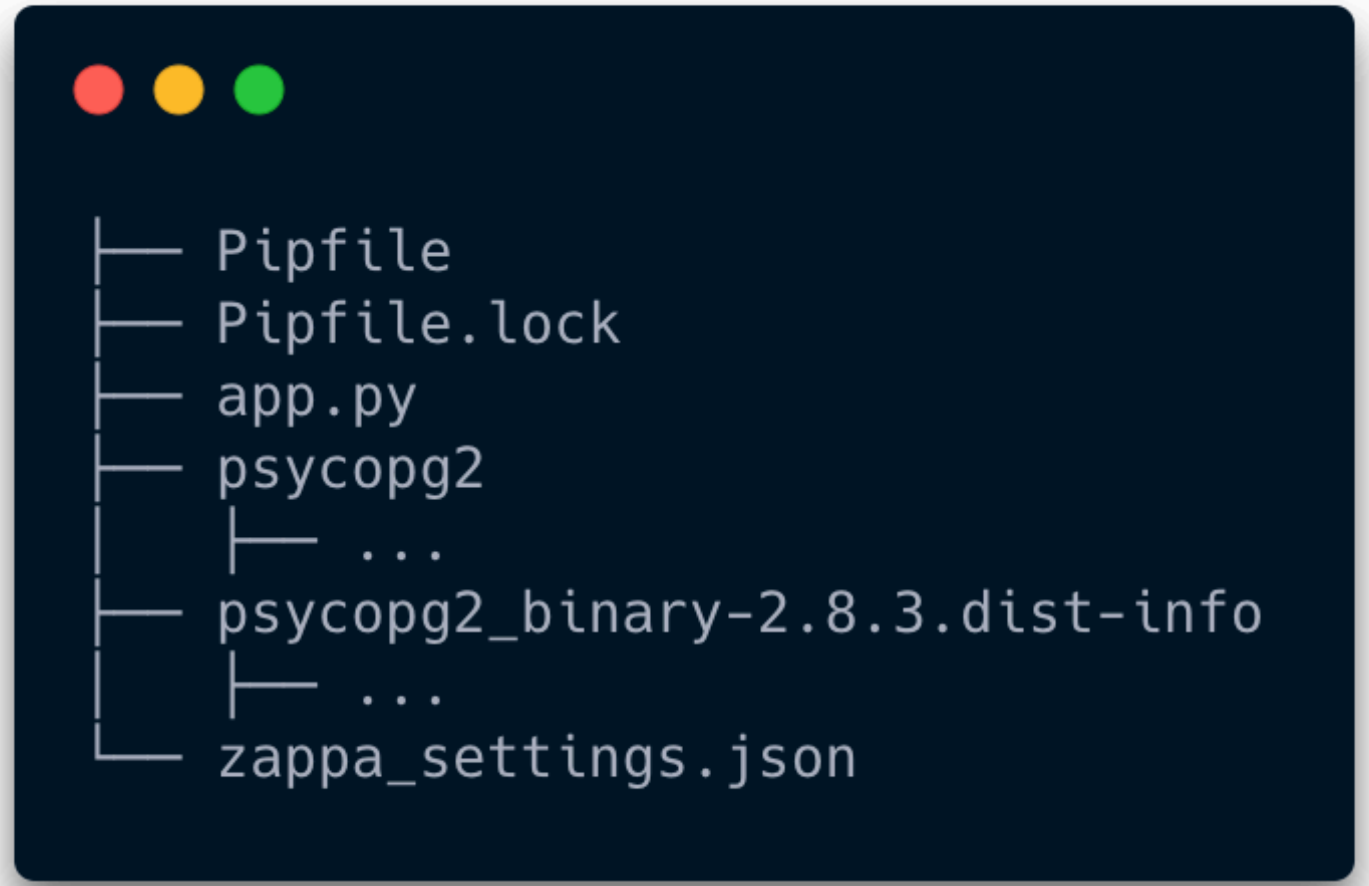
5.1. Binaries

- No `pip install` on Lambda
- Zappa uploads all dependencies from localhost
- My **`psycopg2`** was compiled for **Mac OS**
- Lambda runs on **Linux AMI**
- **`psycopg2`** compiled for **Mac OS** won't work on **Linux AMI!**

Workaround?

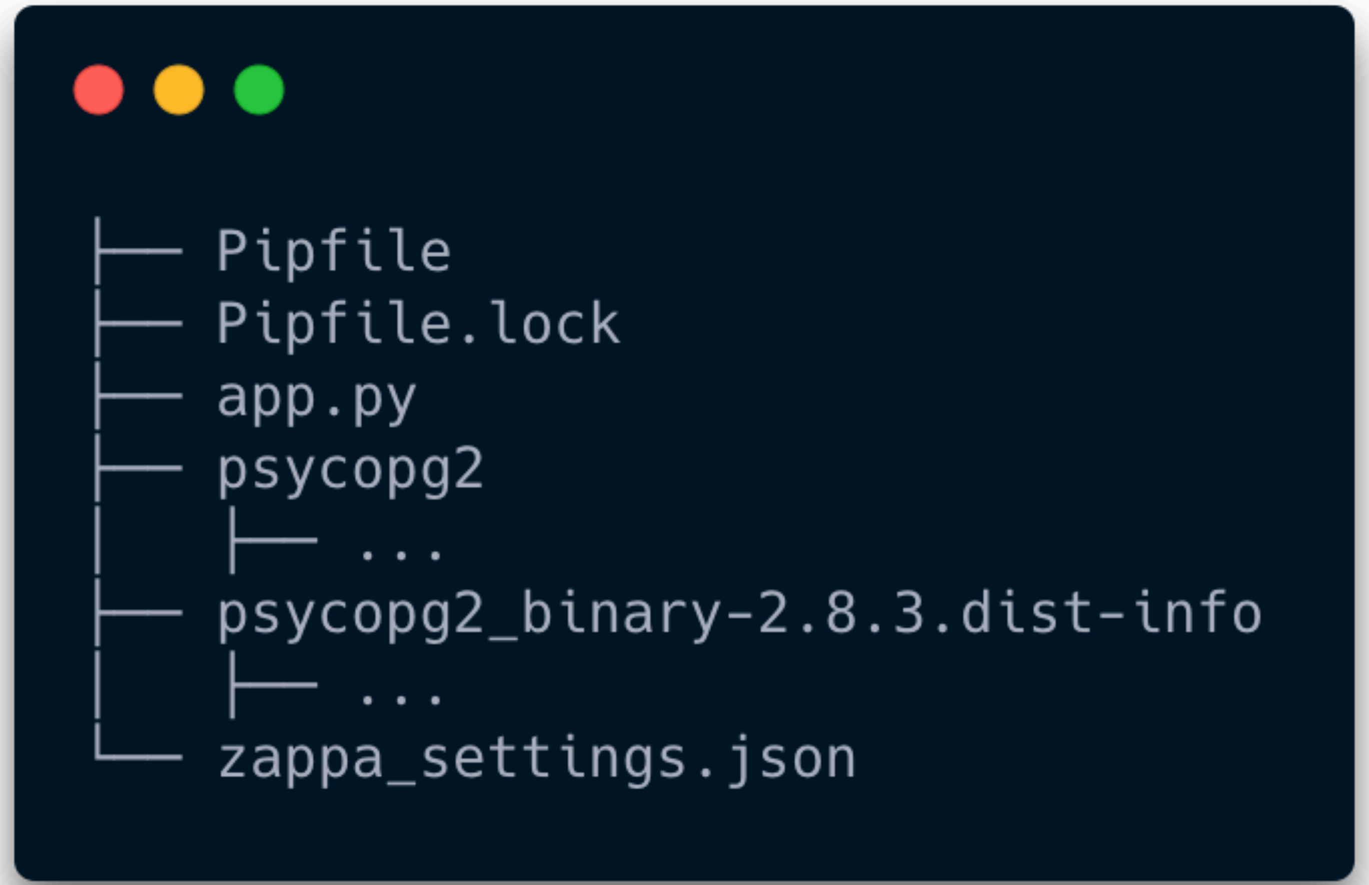
Build package in Lambda environment,
Put package in project folder

```
docker run --rm -v $PWD/:/var/task \
    lambci/lambda:build-python3.6 \
    pip install psycopg2_binary \
    -t /var/task
```



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

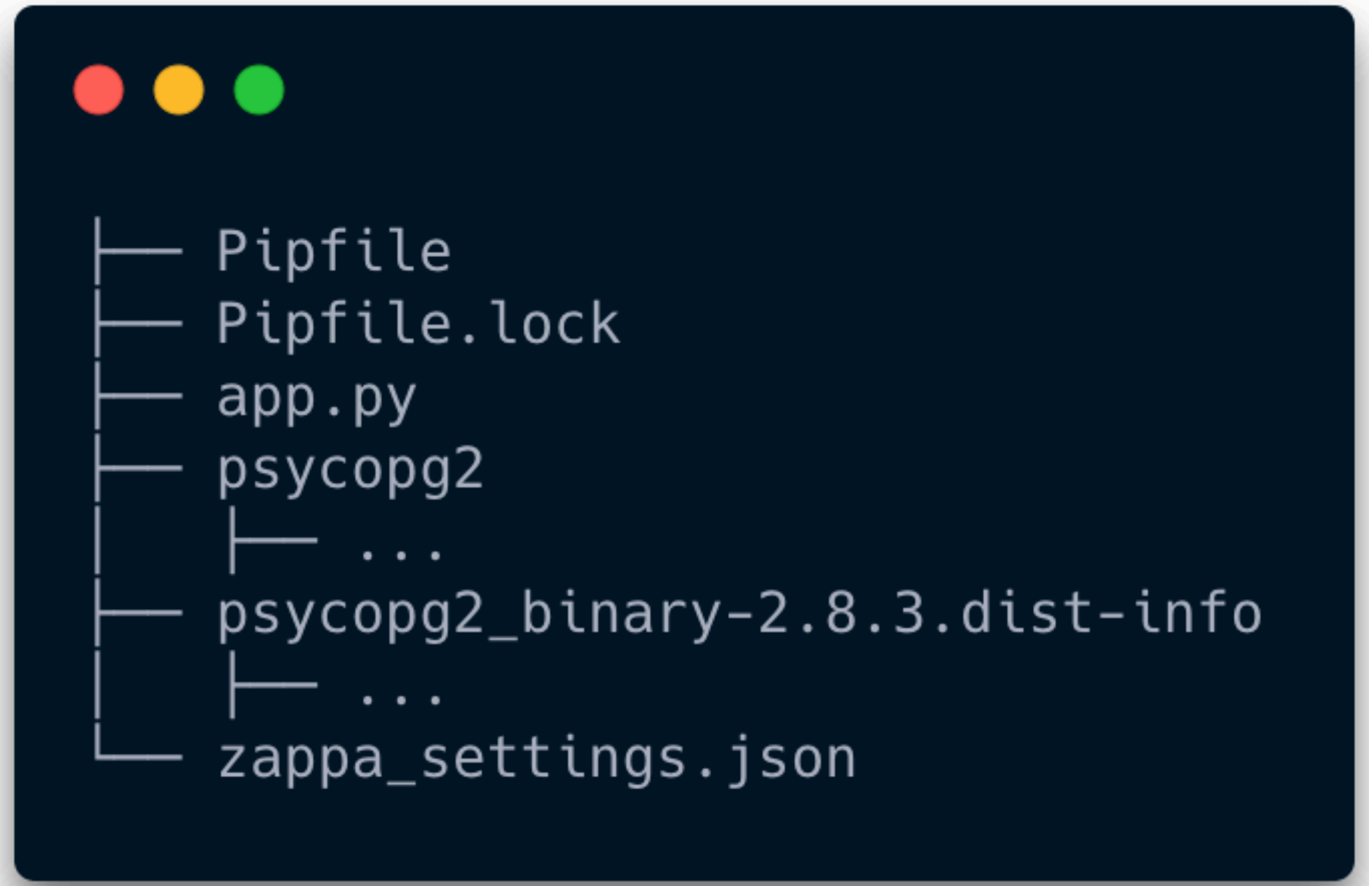
But...



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

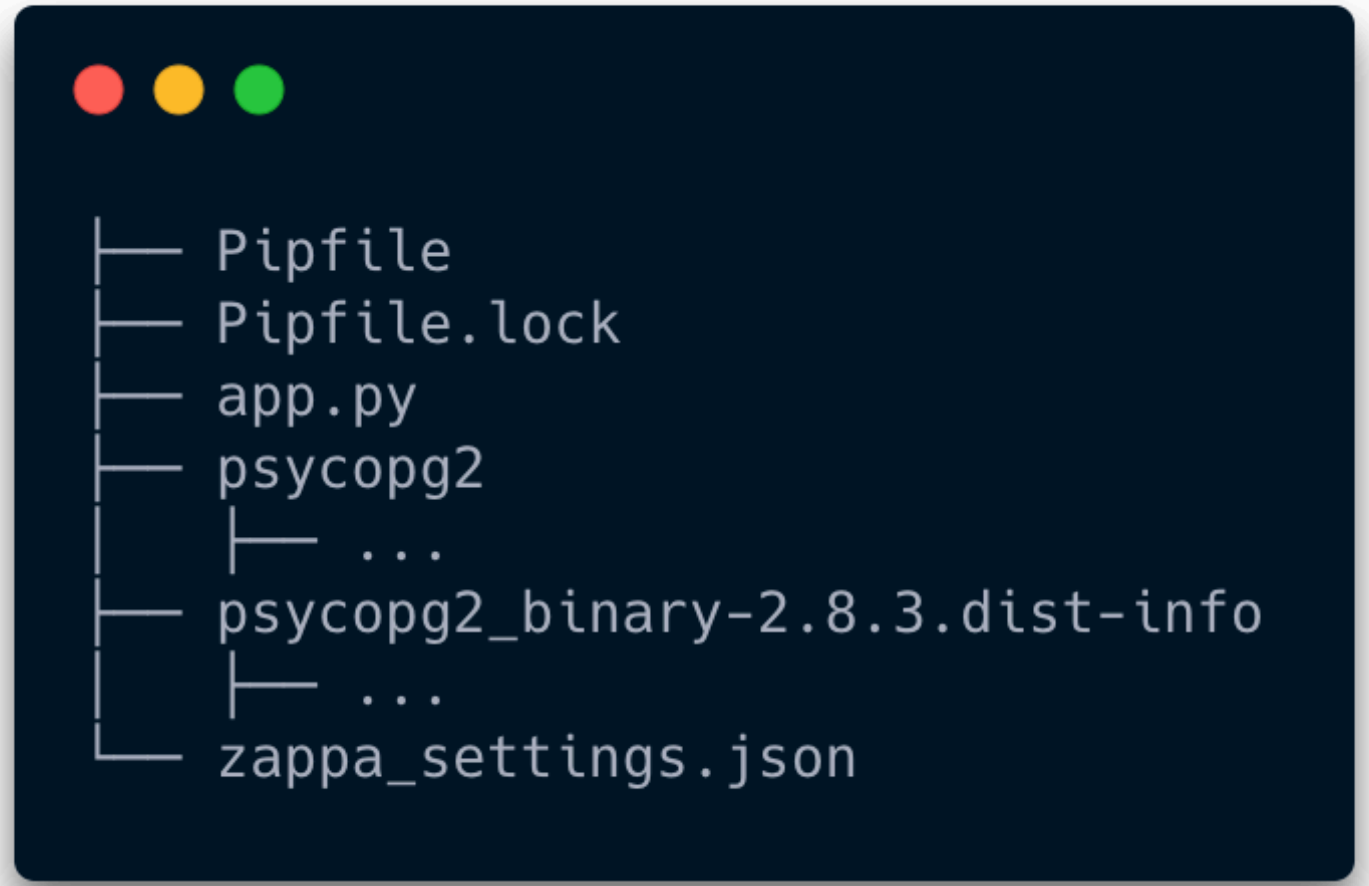
- Now it's not working locally!



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

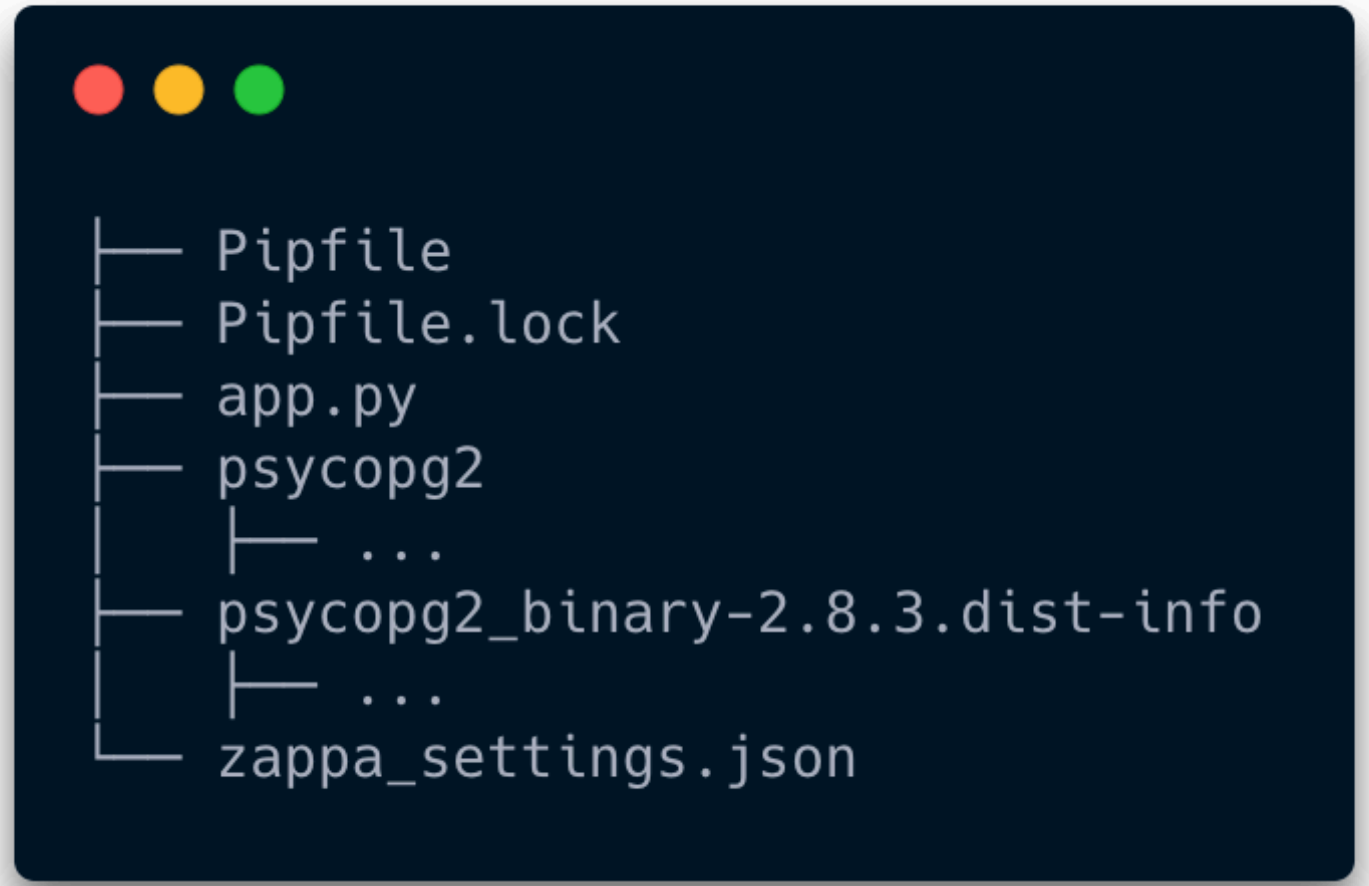
- Now it's not working locally!
- It's hard to use Postgres with Zappa.



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

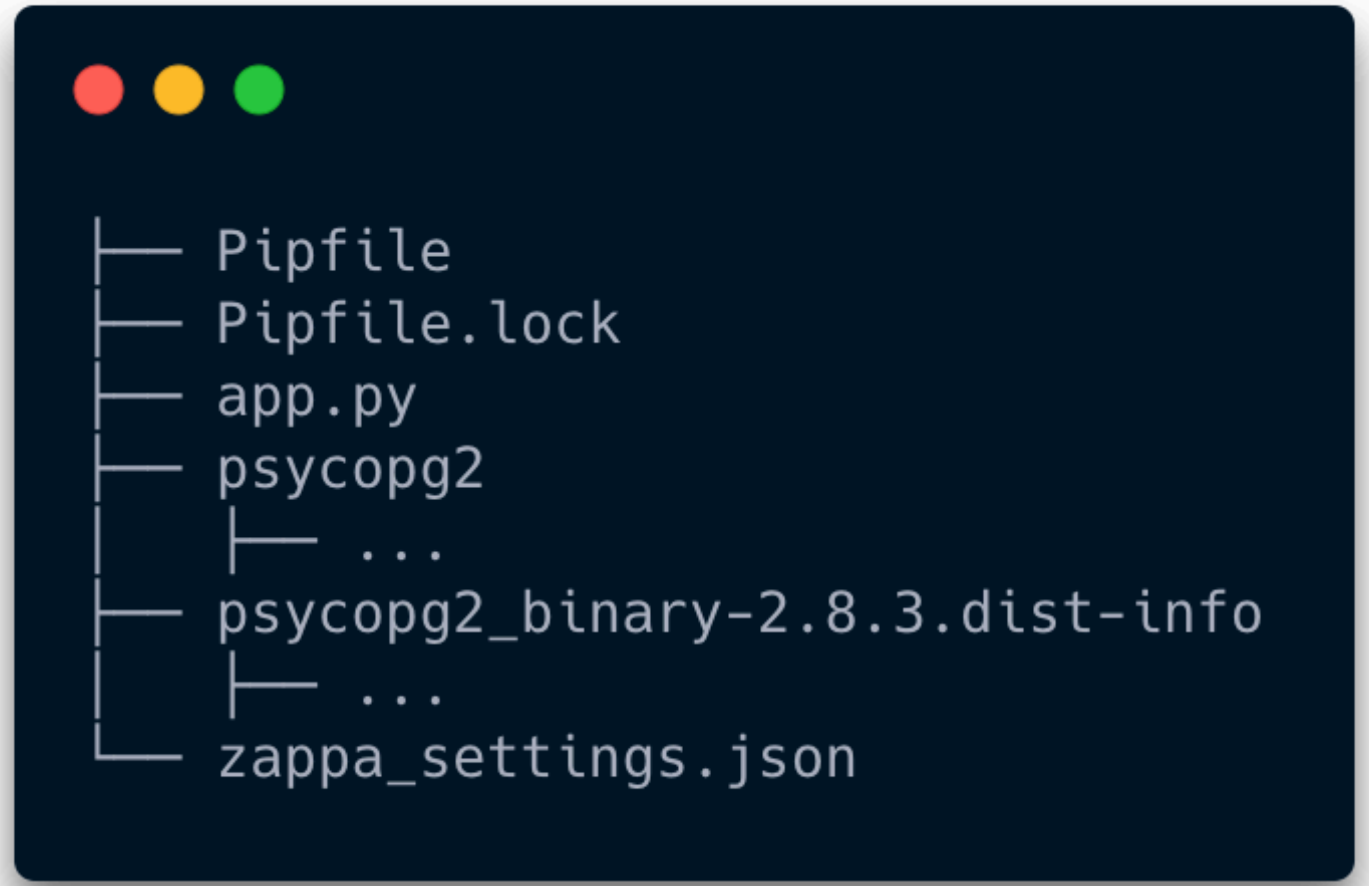
- Now it's not working locally!
- **It's hard to use Postgres with Zappa.**
- Or any other binary:



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

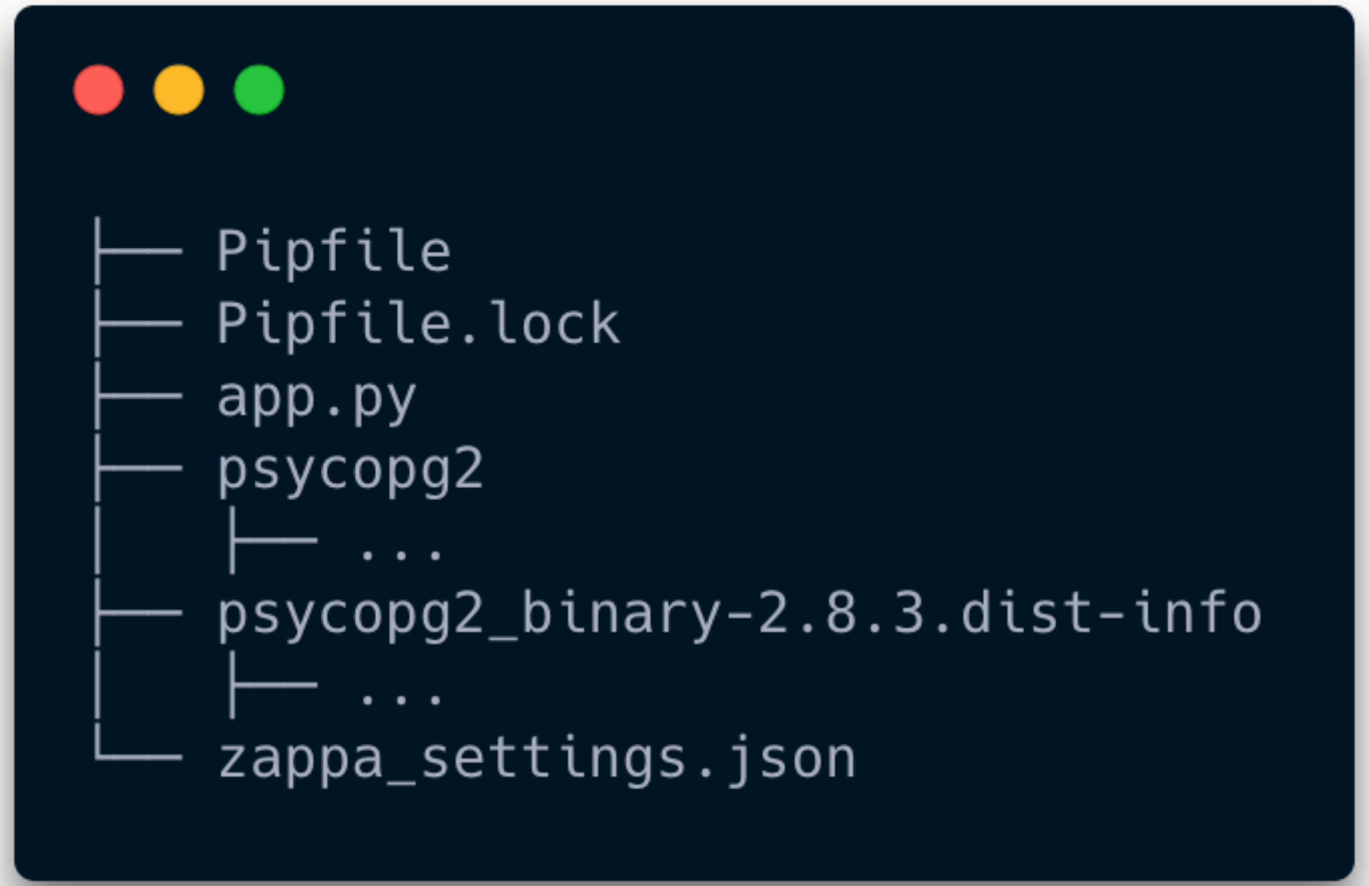
- Now it's not working locally!
- **It's hard to use Postgres with Zappa.**
- Or any other binary:
  - numpy



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

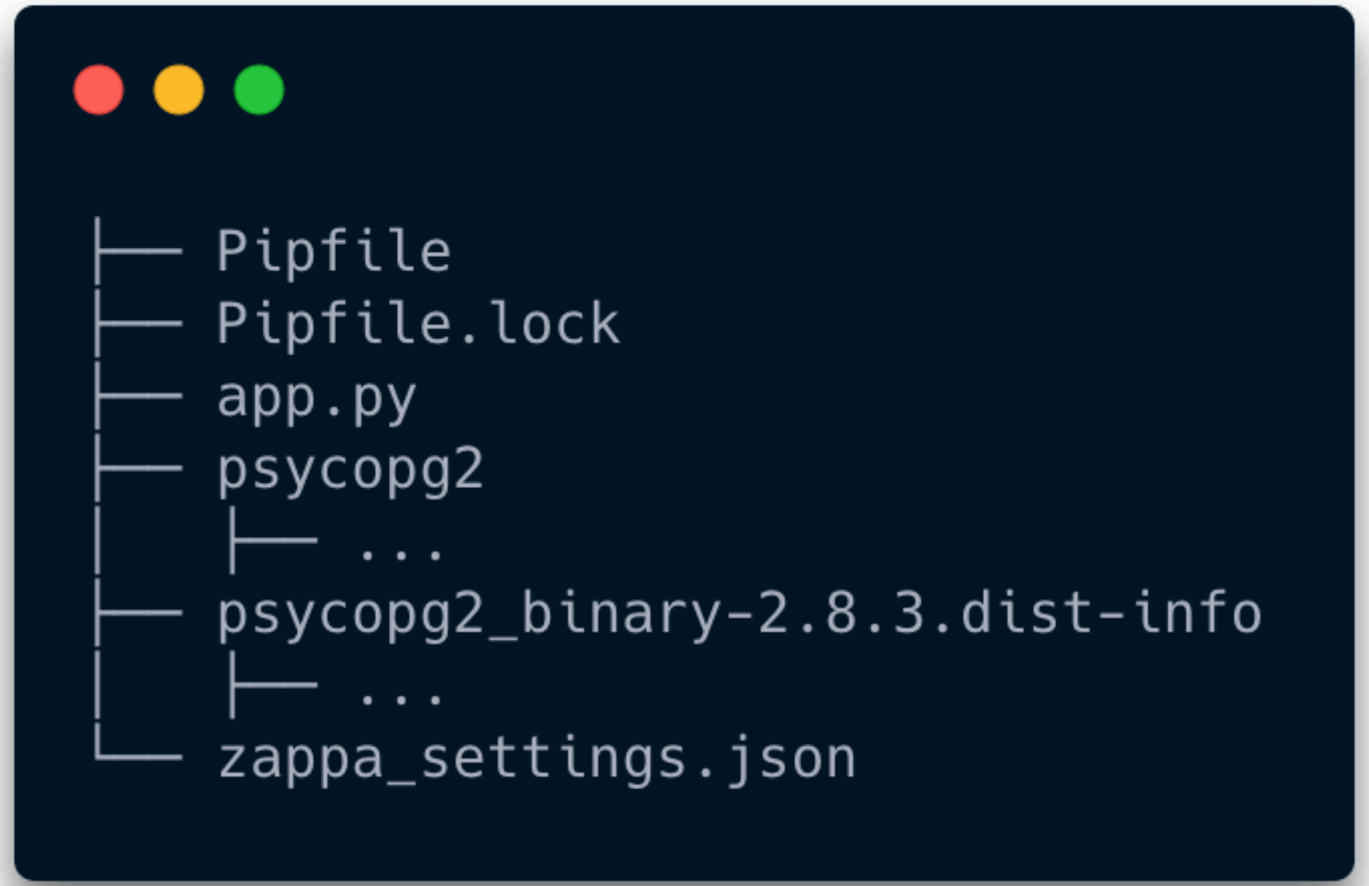
- Now it's not working locally!
- **It's hard to use Postgres with Zappa.**
- Or any other binary:
  - numpy
  - Pillow



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

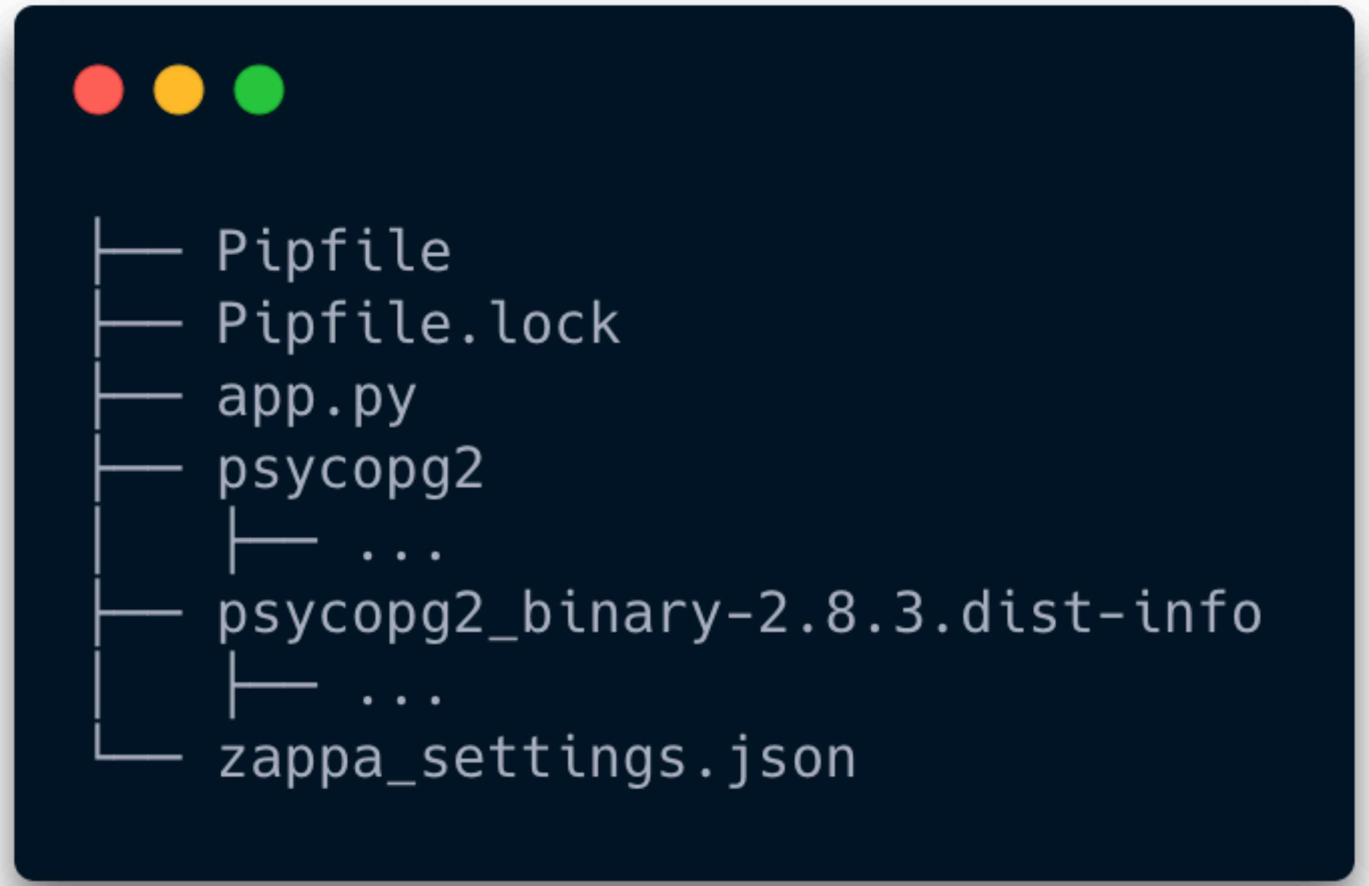
- Now it's not working locally!
- **It's hard to use Postgres with Zappa.**
- Or any other binary:
  - numpy
  - Pillow
  - cryptography



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

But...

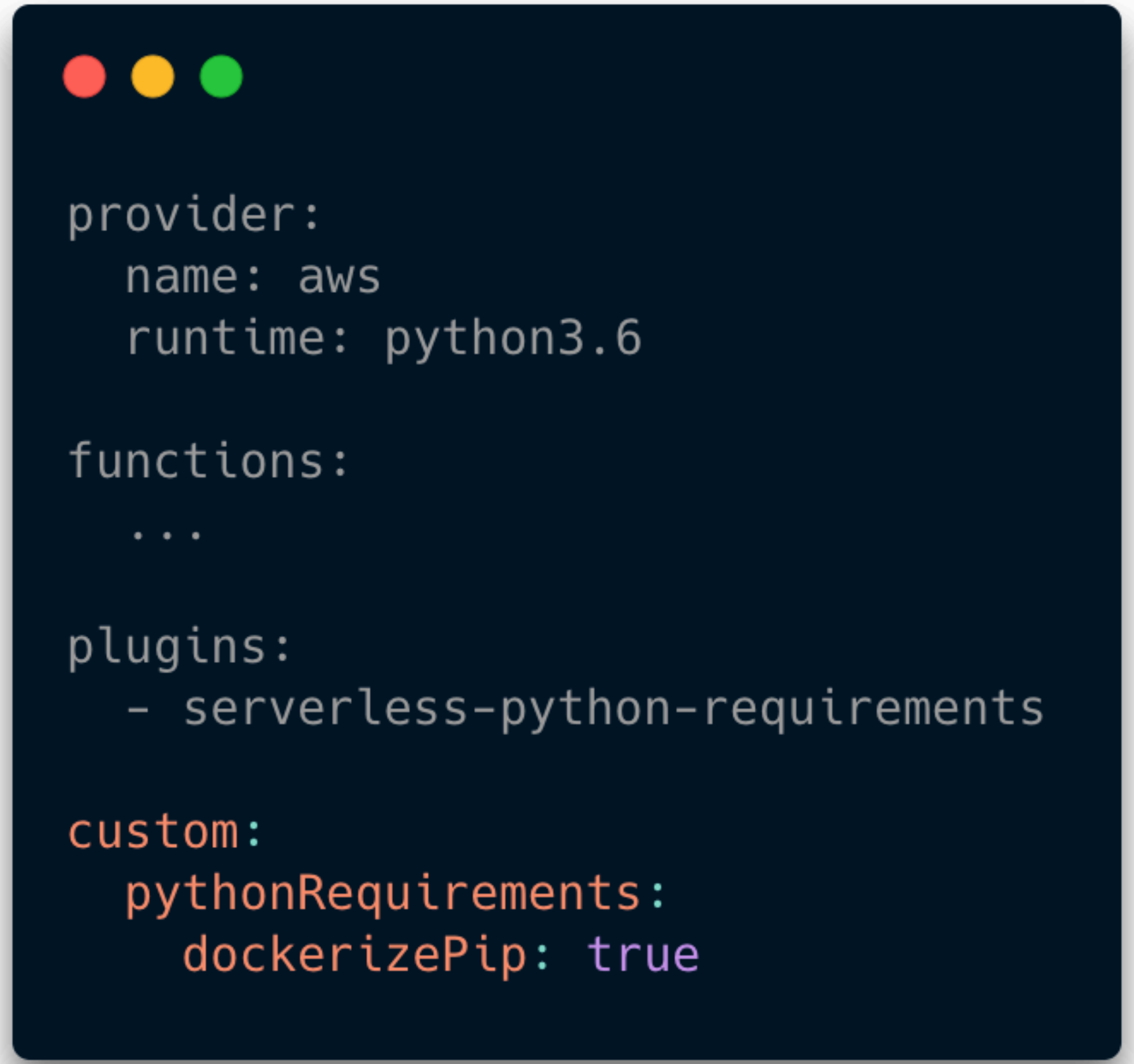
- Now it's not working locally!
- **It's hard to use Postgres with Zappa.**
- Or any other binary:
    - numpy
    - Pillow
    - cryptography
    - ...



```
├── Pipfile
├── Pipfile.lock
├── app.py
├── psycopg2
│   ├── ...
├── psycopg2_binary-2.8.3.dist-info
│   ├── ...
└── zappa_settings.json
```

Back to Serverless Framework

That's all folks! ----->



```
provider:
  name: aws
  runtime: python3.6

functions:
  ...

plugins:
  - serverless-python-requirements

custom:
  pythonRequirements:
    dockerizePip: true
```

5.2. Connections & Concurrency

- OK so we have a Postgres facing app.
- Let's test its throughput!

Our model

```
class Url(Base):

    __tablename__ = 'url'

    id = Column(Integer, primary_key=True)
    code = Column(String, index=True)
    origin = Column(String)
```

Our handler

```python
def handler(event, context):
    ...
    url = payload['url']
    code = make_code(url)

    url = Url(origin=origin, code=code)
    session.add(url)
    session.commit()

    return code
```

```sql
INSERT INTO url (code, origin)
  VALUES (?, ?)
  ('6d8eef', 'https://long_url.com')
```

Our handler

```
def handler(event, context):
    ...
    url = payload['url']
    code = make_code(url)

    url = Url(origin=origin, code=code)
    session.add(url)
    session.commit()

    return code

INSERT INTO url (code, origin)
  VALUES (?, ?)
  ('6d8eef', 'https://long_url.com')
```

Our handler

```
def handler(event, context):
    ...
    url = payload['url']
    code = make_code(url)

    url = Url(origin=origin, code=code)
    session.add(url)
    session.commit()

    return code

INSERT INTO url (code, origin)
  VALUES (?, ?)
  ('6d8eef', 'https://long_url.com')
```

Our handler

```
def handler(event, context):
    ...
    url = payload['url']
    code = make_code(url)

    url = Url(origin=origin, code=code)
    session.add(url)
    session.commit()

    return code

INSERT INTO url (code, origin)
  VALUES (?, ?)
  ('6d8eef', 'https://long_url.com')
```

Our handler

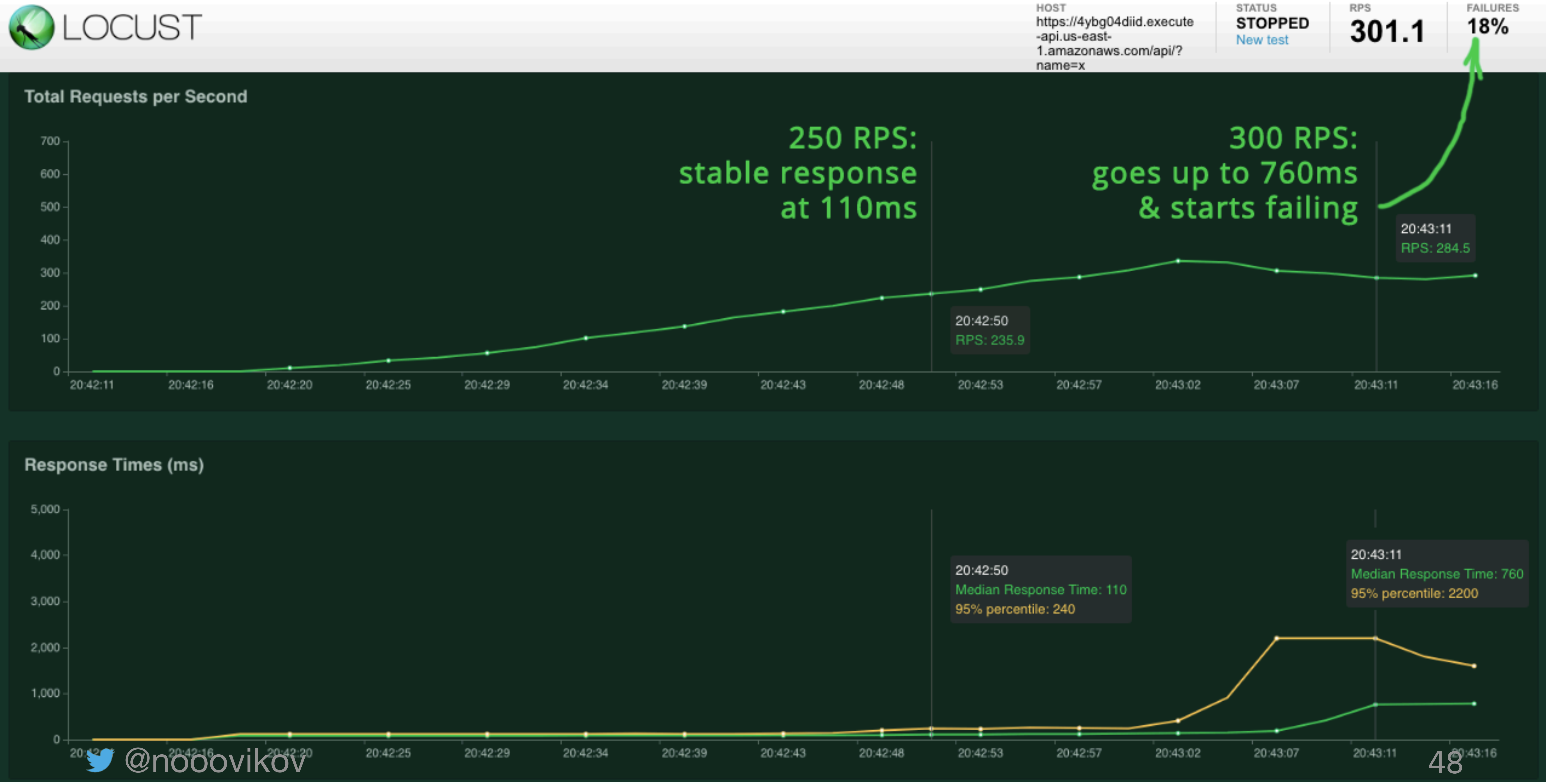
```python
def handler(event, context):
    ...
    url = payload['url']
    code = make_code(url)

    url = Url(origin=origin, code=code)
    session.add(url)
    session.commit()

    return code
```

```sql
INSERT INTO url (code, origin)
  VALUES (?, ?)
  ('6d8eef', 'https://long_url.com')
```

Our handler

```
def handler(event, context):
    ...
    url = payload['url']
    code = make_code(url)

    url = Url(origin=origin, code=code)
    session.add(url)
    session.commit()

    return code
```

```
INSERT INTO url (code, origin)
  VALUES (?, ?)
  ('6d8eef', 'https://long_url.com')
```



LOCUST

HOST https://4ybg04diid.execute-api.us-east-1.amazonaws.com/api/?name=x

STATUS STOPPED New test

RPS 301.1

FAILURES 18%

250 RPS:
stable response
at 110ms

300 RPS:
goes up to 760ms
& starts failing

What happens at 300 RPS?

```
(psycopg2.OperationalError) FATAL:
remaining connection slots are reserved
for non-replication superuser connections
```

Why?

What happens at 300 RPS?

```
(psycopg2.OperationalError) FATAL:
remaining connection slots are reserved
for non-replication superuser connections
```

Why?

- Postgres connections are limited

What happens at 300 RPS?

```
(psycopg2.OperationalError) FATAL:
remaining connection slots are reserved
for non-replication superuser connections
```

Why?

- Postgres connections are limited
- Each Lambda invocation spawns a Postgres connection

What happens at 300 RPS?

```
(psycopg2.OperationalError) FATAL:
remaining connection slots are reserved
for non-replication superuser connections
```

Why?

- Postgres connections are limited
- Each Lambda invocation spawns a Postgres connection
- N invocations => N connections

What happens at 300 RPS?

```
(psycopg2.OperationalError) FATAL:
remaining connection slots are reserved
for non-replication superuser connections
```

Why?

- Postgres connections are limited
- Each Lambda invocation spawns a Postgres connection
- N invocations => N connections
- We hit our DB connection limit

Solution: PGBouncer

Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa

Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**

Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**
- Or use DynamoDB

Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**
- Or use DynamoDB
- And other AWS tools:

Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**
- Or use DynamoDB
- And other AWS tools:
  - SQS for queues

Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**
- Or use DynamoDB
- And other AWS tools:
    - SQS for queues
    - S3 for static hosting

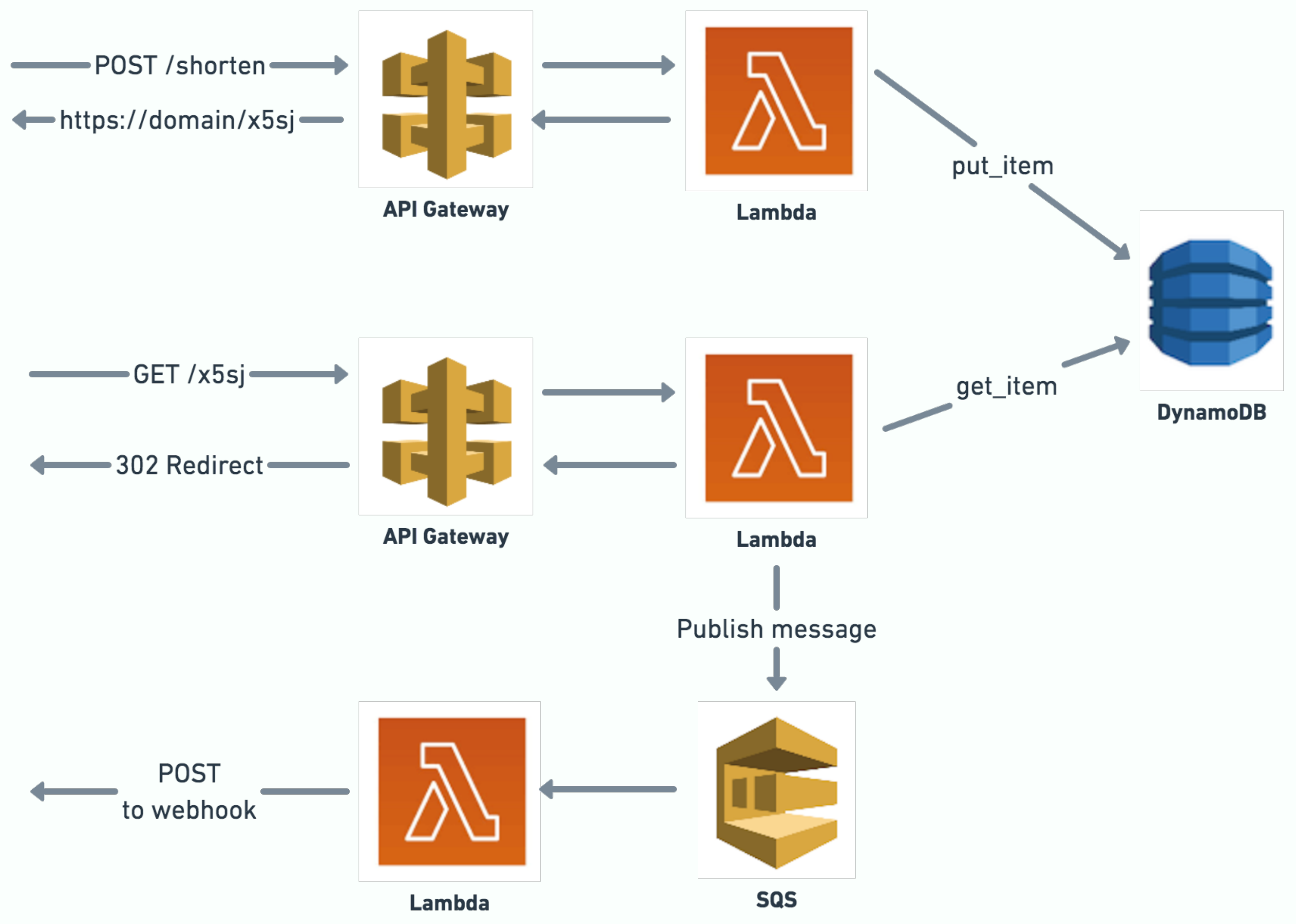
Solution: PGBouncer

- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**
- Or use DynamoDB
- And other AWS tools:
  - SQS for queues
  - S3 for static hosting
  - Cloudfront for CDN

Solution: PGBouncer

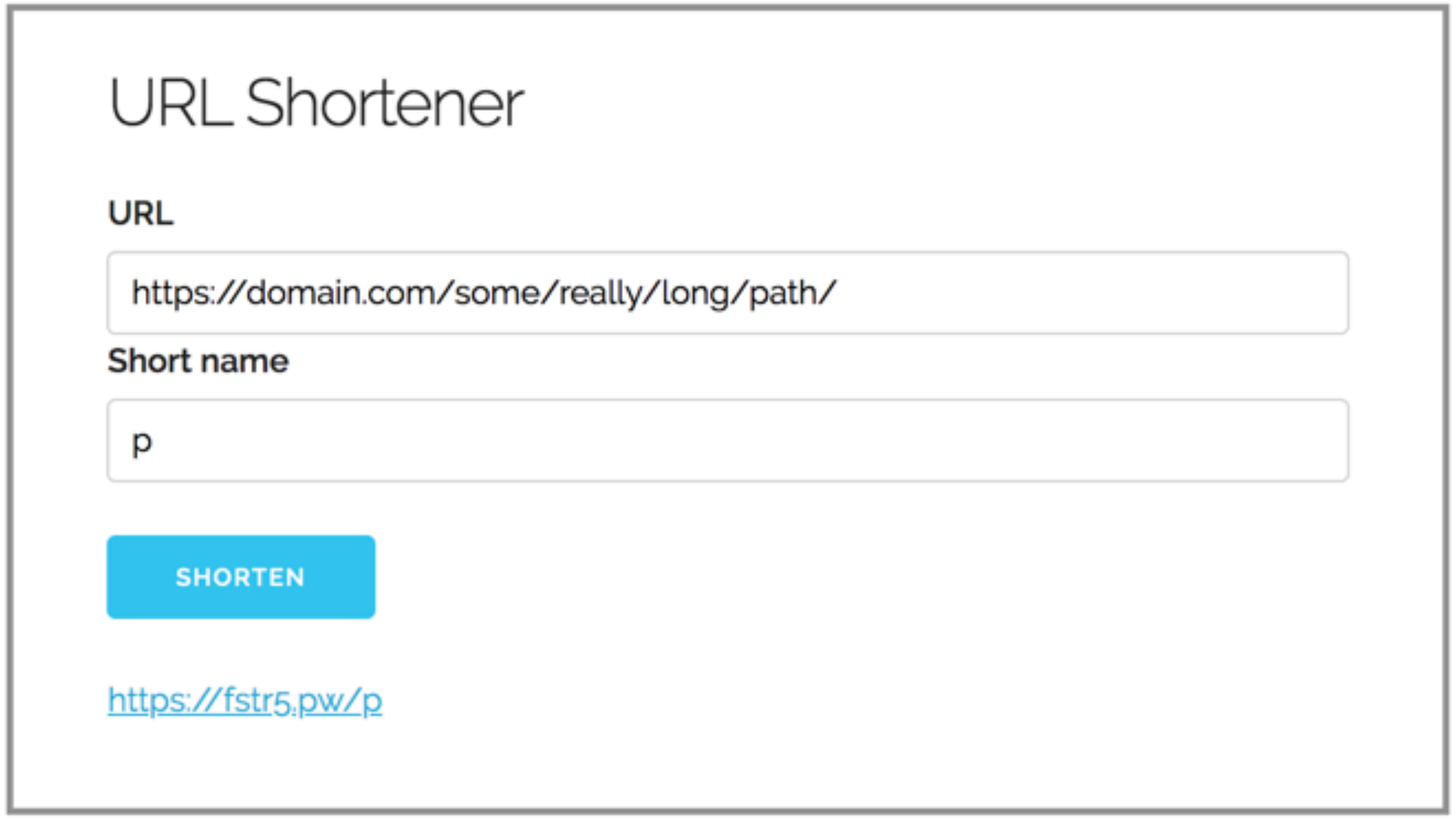
- Postgres Connection Pooling: Tutorial for Zappa
- **You need a server to run Serverless. ¯_(ツ)_/¯**
- Or use DynamoDB
- And other AWS tools:
    - SQS for queues
    - S3 for static hosting
    - Cloudfront for CDN
    - ...

URL shortener with DynamoDB and 3 lambdas



Example: Our shortener

https://fstr5.pw/piterpy



URL Shortener

URL

https://domain.com/some/really/long/path/

Short name

p

https://fstr5.pw/p

Boring stuff

- Logging
- Debugging
- Exception handling
- Testing
- …

5.3. Logging

This is easy.

```
zappa tail --since 1m
```

```
serverless logs -f myfunction
```

Or just open AWS web console. (Cloudwatch)

5.4. Exceptions

Good old Sentry

```
import sentry_sdk
from sentry_sdk.integrations.aws_lambda import AwsLambdaIntegration

sentry_sdk.init(
    dsn="https://xxx@sentry.io/123",
    integrations=[AwsLambdaIntegration()]
)

def my_function(event, context):
    ...
```

5.5. Uploading dependencies

```
./requests      376K
./psycopg2      564K
./sentry_sdk    596K
./PIL           11M
./numpy         23M
./pandas        65M
```

Do we really have to upload them every time we deploy code?

5.5. Uploading dependencies

- Not always.
- We have lambda layers! docs
- Supported in Serverless Framework: docs
- Not supported in Zappa.

5.6. Running locally

- **Zappa**
  - Just run your Flask server
  - Debug mode: as always
- **Serverless Framework**
  - `npm install serverless-offline`
  - run `sls offline`
  - No debug mode :(

5.7. Testing

- You can test your logic.
- But how do you test your side-effects?

5.7. Testing

You have three options.

5.7. Testing

You have three options.

1. Just don't test it. :)

5.7. Testing

You have three options.

1. Just don't test it. :)
2. Run cloud services locally.

5.7. Testing

You have three options.

1. Just don't test it. :)
2. Run cloud services locally.
3. Mock out calls to the cloud.

Local cloud

Just DynamoDB: use DynamoDB Local

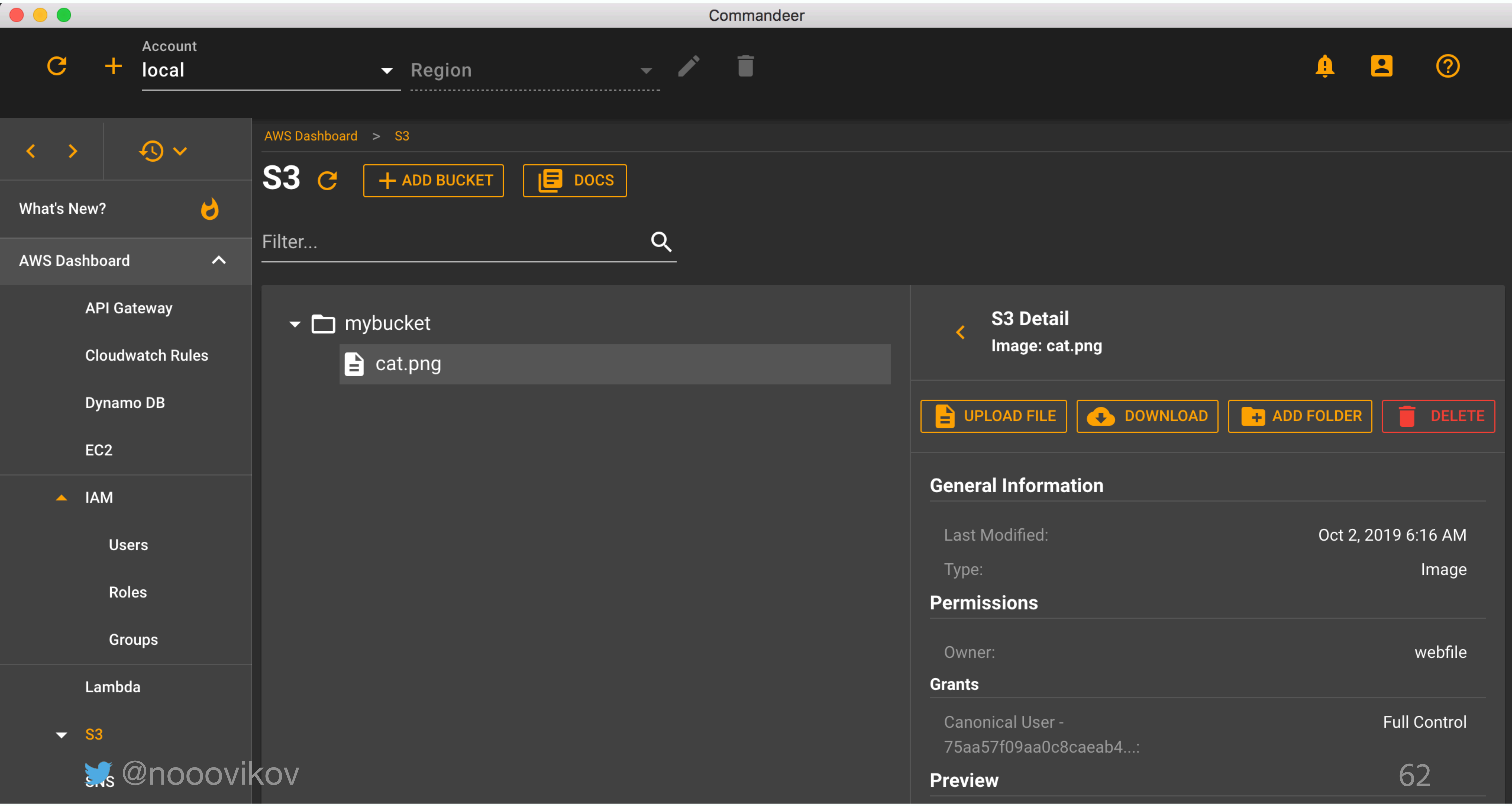
- `docker run -p 8000:8000 amazon/dynamodb-local`

Just S3: use Minio

- `docker run -p 9000:9000 minio/minio server /data`

(Almost) all AWS Stack: use Localstack

- https://github.com/localstack/localstack
- Start with **`docker-compose`**


Commandeer

Account
local
Region

AWS Dashboard > S3

S3
ADD BUCKET
DOCS

Filter...

What's New?
AWS Dashboard
API Gateway
Cloudwatch Rules
Dynamo DB
EC2
IAM
Users
Roles
Groups
Lambda
S3
SNS

mybucket
cat.png

S3 Detail
Image: cat.png

UPLOAD FILE
DOWNLOAD
ADD FOLDER
DELETE

General Information

Last Modified:	Oct 2, 2019 6:16 AM
Type:	Image

Permissions

Owner:	webfile

Grants

Canonical User - 75aa57f09aa0c8caeab4...:	Full Control

Preview

@nooovikov

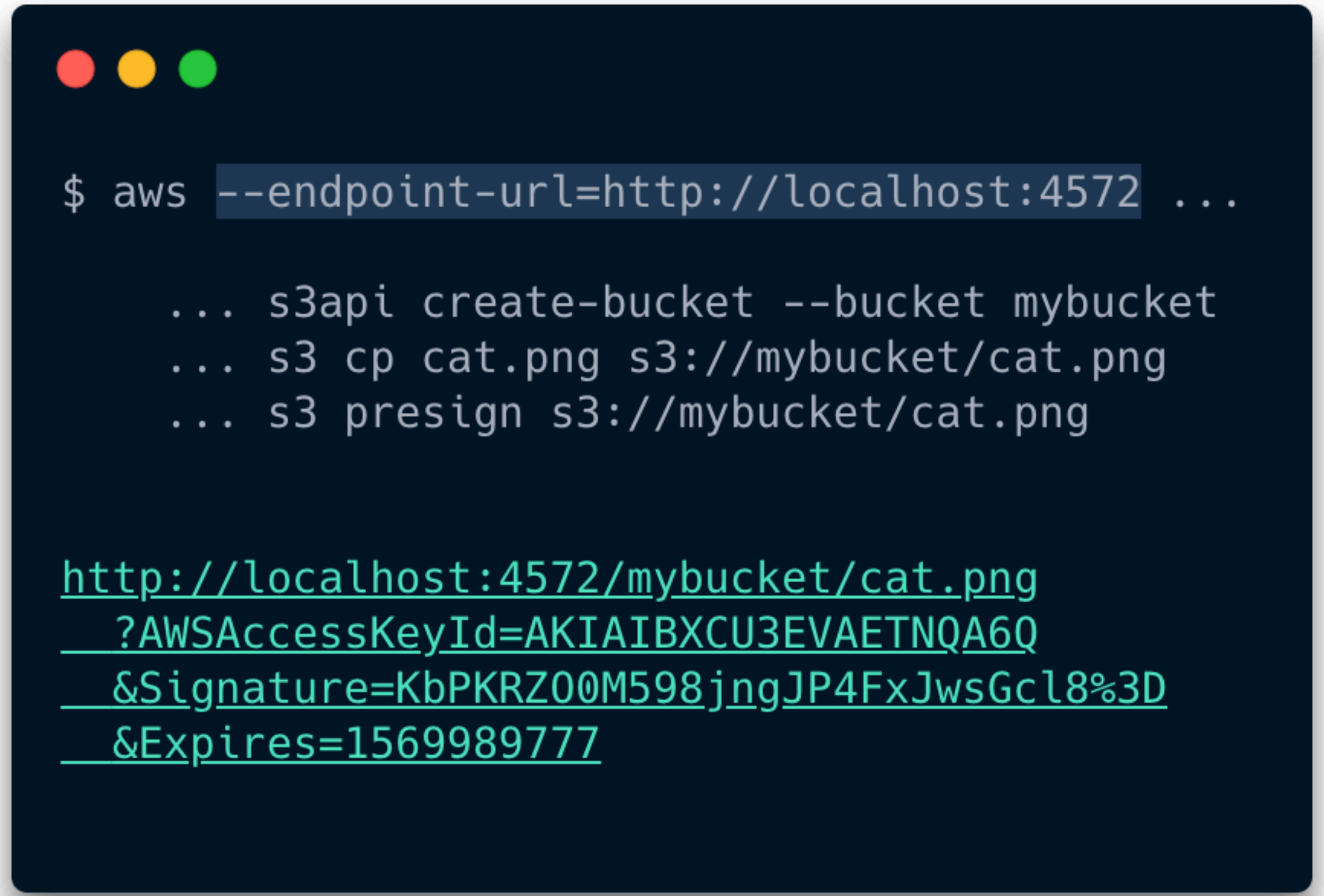
62

Localstack

- Runs most of AWS services locally:
  - Port 4572: S3
  - Port 4569: DynamoDB
  - etc.
- Use regular AWS client, but specify

`--endpoint-url`

- Has a sepatate UI (getcommandeer.com)



```
$ aws --endpoint-url=http://localhost:4572 ...

    ... s3api create-bucket --bucket mybucket
    ... s3 cp cat.png s3://mybucket/cat.png
    ... s3 presign s3://mybucket/cat.png


http://localhost:4572/mybucket/cat.png
  ?AWSAccessKeyId=AKIAIBXCU3EVAETNQA6Q
  &Signature=KbPKRZO0M598jngJP4FxJwsGcl8%3D
  &Expires=1569989777
```

Mocking calls

```python
import boto3

from moto import mock_dynamodb2

@mock_dynamodb2
def test_moto():
    client = boto3.resource('dynamodb')

    table = client.Table('urls')

    table.put_item(
        Item={'url': 'https://domain.com', 'code': 'x7sta9'}
    )
    item = table.get_item(Key={'code': 'x7sta9'})

    assert item['Item']['url'] == 'https://domain.com'
```

Mocking calls

```
import boto3

from moto import mock_dynamodb2

@mock_dynamodb2
def test_moto():
    client = boto3.resource('dynamodb')

    table = client.Table('urls')

    table.put_item(
        Item={'url': 'https://domain.com', 'code': 'x7sta9'}
    )
    item = table.get_item(Key={'code': 'x7sta9'})

    assert item['Item']['url'] == 'https://domain.com'
```

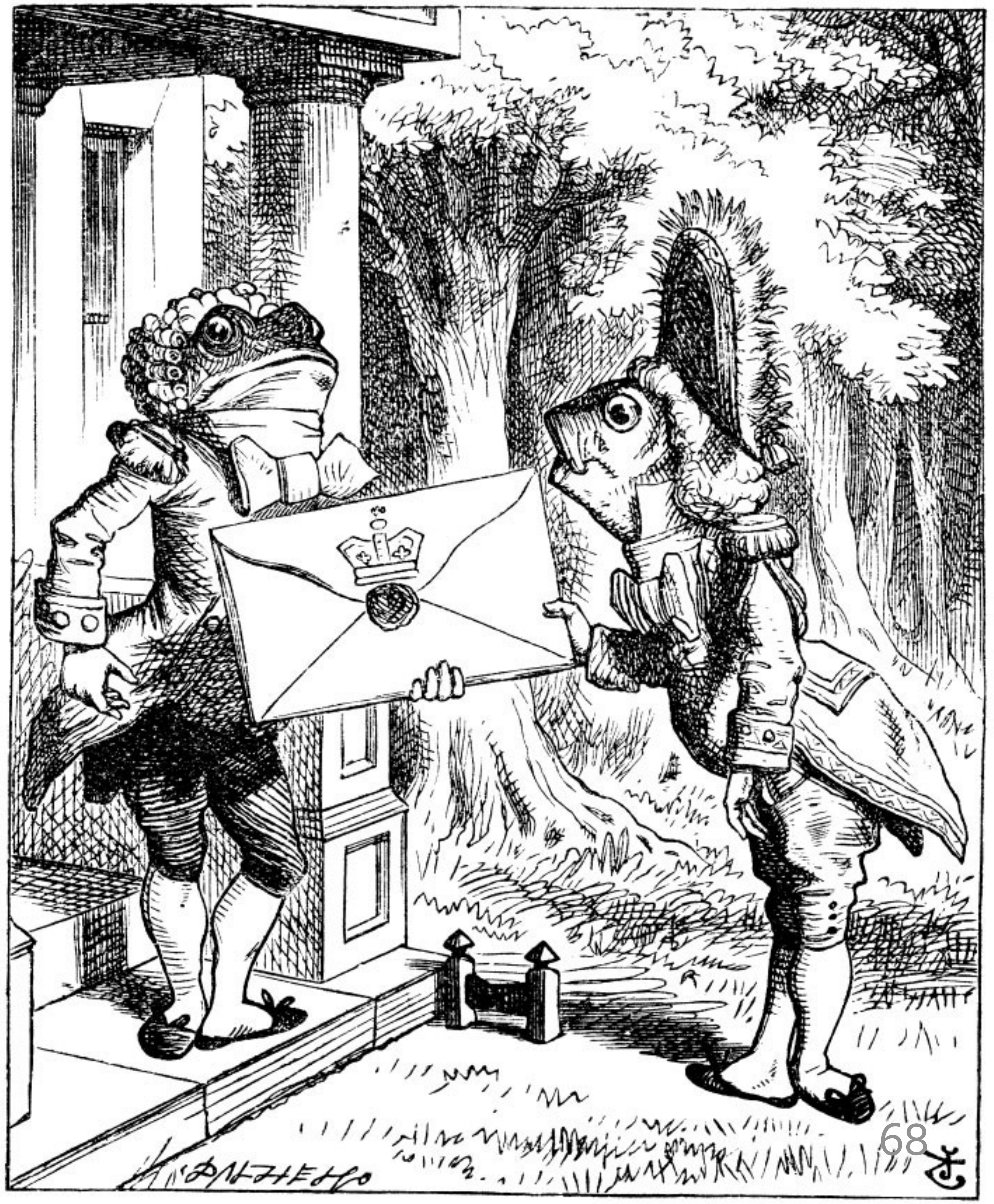
Mocking calls

```
import boto3

from moto import mock_dynamodb2

@mock_dynamodb2
def test_moto():
    client = boto3.resource('dynamodb')

    table = client.Table('urls')

    table.put_item(
        Item={'url': 'https://domain.com', 'code': 'x7sta9'}
    )
    item = table.get_item(Key={'code': 'x7sta9'})

    assert item['Item']['url'] == 'https://domain.com'
```

Mocking calls

```
import boto3

from moto import mock_dynamodb2

@mock_dynamodb2
def test_moto():
    client = boto3.resource('dynamodb')

    table = client.Table('urls')

    table.put_item(
        Item={'url': 'https://domain.com', 'code': 'x7sta9'}
    )
    item = table.get_item(Key={'code': 'x7sta9'})

    assert item['Item']['url'] == 'https://domain.com'
```

6. Costs

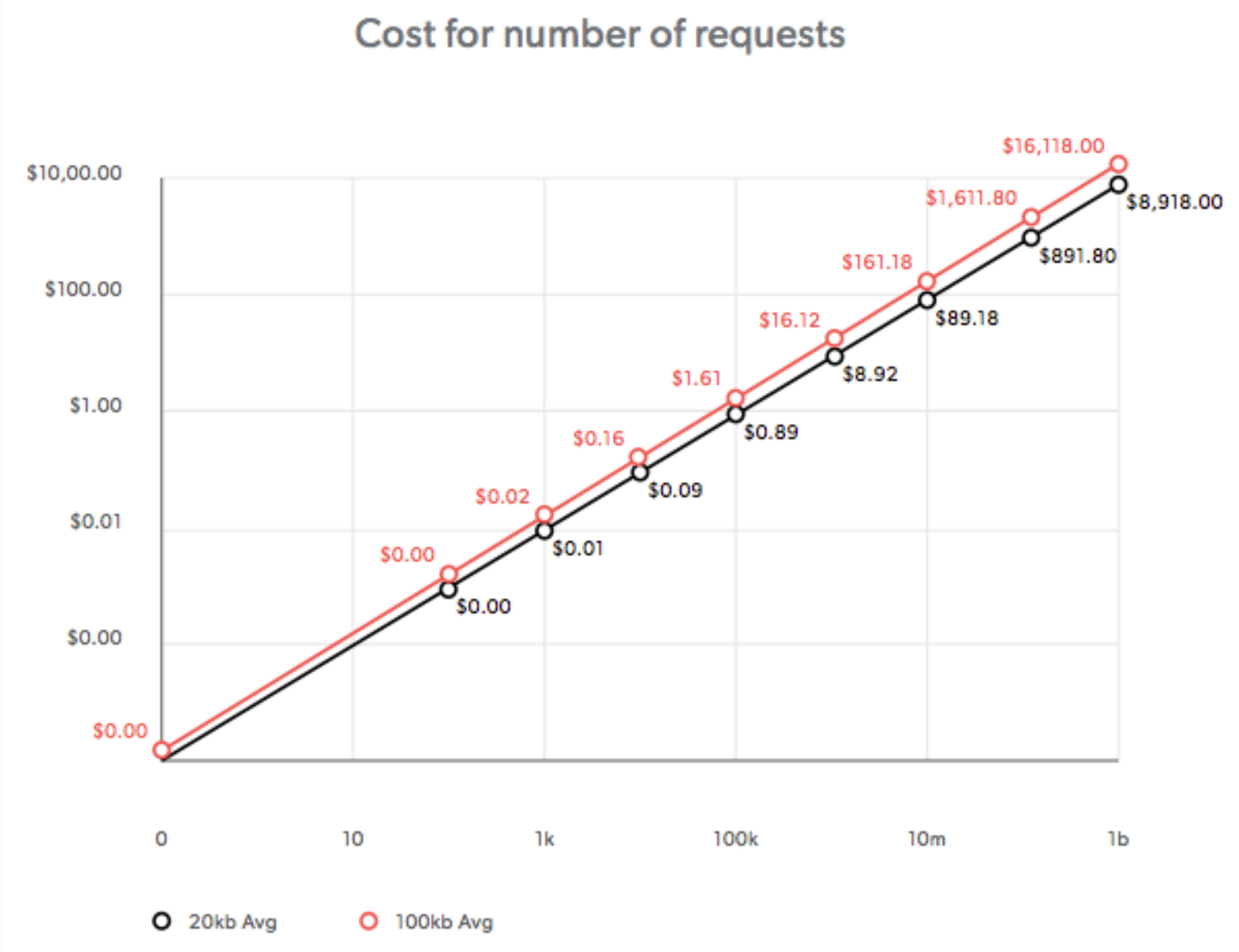


@nooovikov

Service	*Units*	*Price per unit*	*Free quota*
Lambda	GB-second	$0.00001667	✅
	1 million requests	$0.2	✅
	GB of data out	$0.09	✅
RDS Postgres	Instance hour	$0.021(μ)/$0.164(L)	✅ (1y)
	GB-month of storage	$0.133	✅
DynamoDB	1 million PUT	$1.25	
	1 million GET	$0.25	
	GB-month of storage	$0.25	✅

Example

- Each lambda runs in 200ms at 512mb Memory
- Each request writes once and reads twice from DynamoDB
- Each request is 20kb or 100kb on average



7. Conclusion

Is it worth it?

Is it worth it?

It depends.

Is it worth it?

It depends.

- ❌ Not for enterprises

Is it worth it?

It depends.

- ❌ Not for enterprises
- ❌ Not for long-running tasks

Is it worth it?

It depends.

- ❌ Not for enterprises
- ❌ Not for long-running tasks
- ❌ Not for ***constant*** high load

Is it worth it?

It depends.

- ❌ Not for enterprises
- ❌ Not for long-running tasks
- ❌ Not for ***constant*** high load
- ✅ Yes for SAAS products

Is it worth it?

It depends.

- ❌ Not for enterprises
- ❌ Not for long-running tasks
- ❌ Not for *constant* high load
- ✅ Yes for SAAS products
- ✅ Yes for data engineering

Is it worth it?

It depends.

- ❌ Not for enterprises
- ❌ Not for long-running tasks
- ❌ Not for ***constant*** high load
- ✅ Yes for SAAS products
- ✅ Yes for data engineering
- ✅ Yes for MVPs, pet projects

Python ❤️ Serverless

Python ❤️ Serverless

- Stop worrying about performance

Python ❤️ Serverless

- Stop worrying about performance
- Simple is better than complex. Serverless is simple.

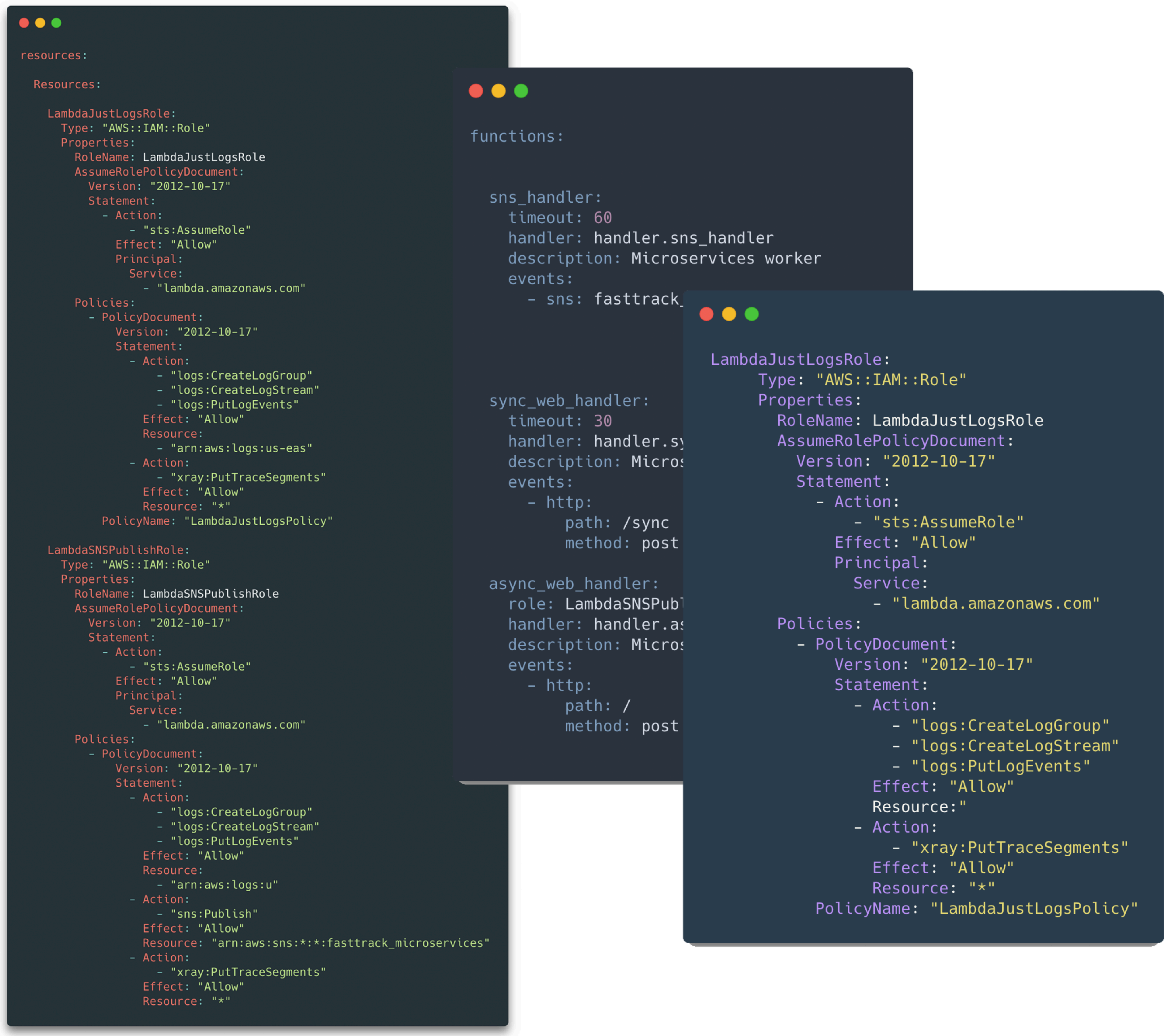
Zen of nginx?

```
server {
    location / {
        fastcgi_pass  localhost:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param QUERY_STRING    $query_string;
    }

    location ~ \.(gif|jpg|png)$ {
        root /data/images;
    }
}
```

Zen of pg_hba.conf?

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

With Serverless, you just write code that you love.



```
resources:

  Resources:

    LambdaJustLogsRole:
      Type: "AWS::IAM::Role"
      Properties:
        RoleName: LambdaJustLogsRole
        AssumeRolePolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Action:
                - "sts:AssumeRole"
              Effect: "Allow"
              Principal:
                Service:
                  - "lambda.amazonaws.com"
        Policies:
          - PolicyDocument:
              Version: "2012-10-17"
              Statement:
                - Action:
                    - "logs:CreateLogGroup"
                    - "logs:CreateLogStream"
                    - "logs:PutLogEvents"
                  Effect: "Allow"
                  Resource:
                    - "arn:aws:logs:us-eas"
                - Action:
                    - "xray:PutTraceSegments"
                  Effect: "Allow"
                  Resource: "*"
            PolicyName: "LambdaJustLogsPolicy"

    LambdaSNSPublishRole:
      Type: "AWS::IAM::Role"
      Properties:
        RoleName: LambdaSNSPublishRole
        AssumeRolePolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Action:
                - "sts:AssumeRole"
              Effect: "Allow"
              Principal:
                Service:
                  - "lambda.amazonaws.com"
        Policies:
          - PolicyDocument:
              Version: "2012-10-17"
              Statement:
                - Action:
                    - "logs:CreateLogGroup"
                    - "logs:CreateLogStream"
                    - "logs:PutLogEvents"
                  Effect: "Allow"
                  Resource:
                    - "arn:aws:logs:u"
                - Action:
                    - "sns:Publish"
                  Effect: "Allow"
                  Resource: "arn:aws:sns:*:*:fasttrack_microservices"
                - Action:
                    - "xray:PutTraceSegments"
                  Effect: "Allow"
                  Resource: "*"
```

```
functions:


  sns_handler:
    timeout: 60
    handler: handler.sns_handler
    description: Microservices worker
    events:
      - sns: fasttrack_



  sync_web_handler:
    timeout: 30
    handler: handler.sy
    description: Micros
    events:
      - http:
          path: /sync
          method: post

  async_web_handler:
    role: LambdaSNSPubl
    handler: handler.as
    description: Micros
    events:
      - http:
          path: /
          method: post
```

```
LambdaJustLogsRole:
     Type: "AWS::IAM::Role"
     Properties:
       RoleName: LambdaJustLogsRole
       AssumeRolePolicyDocument:
         Version: "2012-10-17"
         Statement:
           - Action:
               - "sts:AssumeRole"
             Effect: "Allow"
             Principal:
               Service:
                 - "lambda.amazonaws.com"
       Policies:
         - PolicyDocument:
             Version: "2012-10-17"
             Statement:
               - Action:
                   - "logs:CreateLogGroup"
                   - "logs:CreateLogStream"
                   - "logs:PutLogEvents"
                 Effect: "Allow"
                 Resource:"
               - Action:
                   - "xray:PutTraceSegments"
                 Effect: "Allow"
                 Resource: "*"
           PolicyName: "LambdaJustLogsPolicy"
```

@nooovikov

With Serverless, you just write code that you love.

Thanks!

Mikhail Novikov
Founder & Dev Lead, ***fstrk.io***

t.me/pyshorts

A little Github demo

Telegram: @rekog_bot

- Receives selfies and detects emotions using AWS Rekognition
- Stores data in DynamoDB

github.com/kurtgn/zappa-rekognition-bot