🐍 Python in the Land of Serverless

David Przybilla

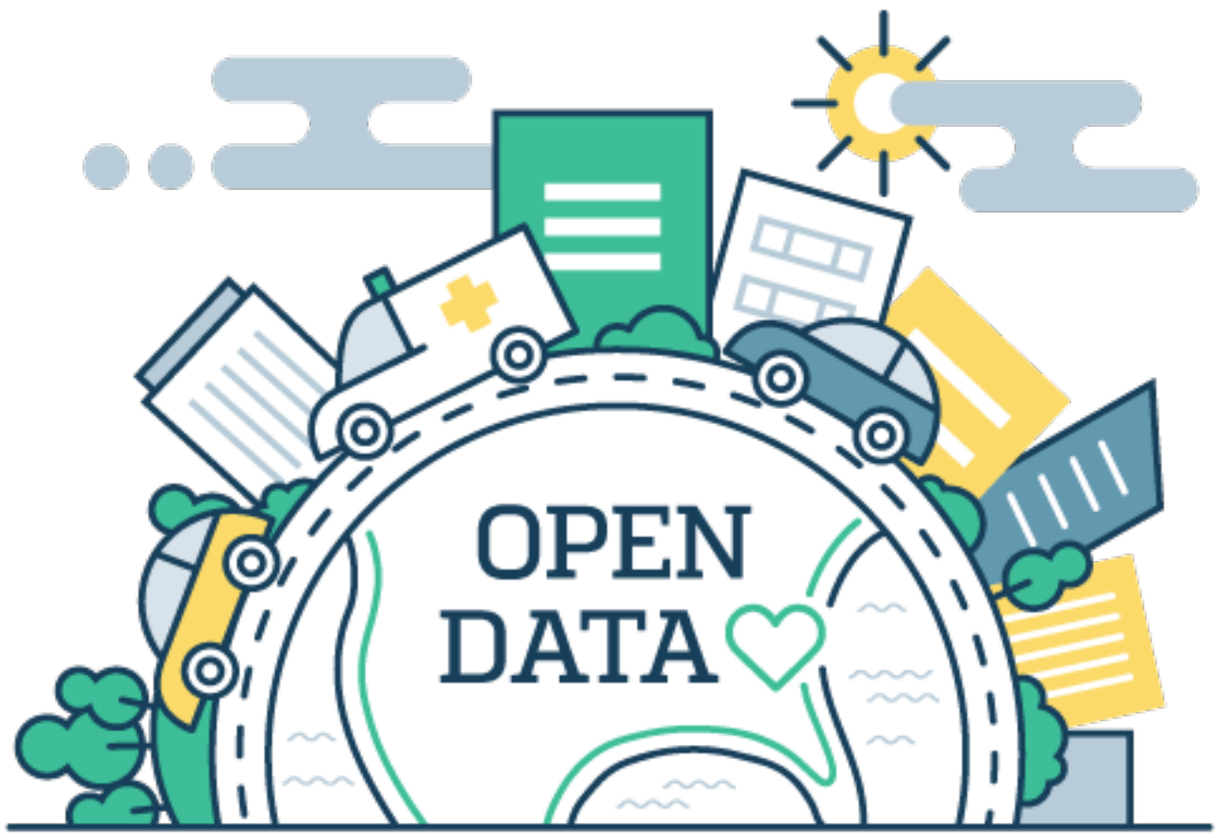
dav009

dav009

🇨🇴 🇨🇴 last time in here was

~9 years ago

NLP
Data Engineering



Gov spending dataset

How to access datasets

ColombiaDev

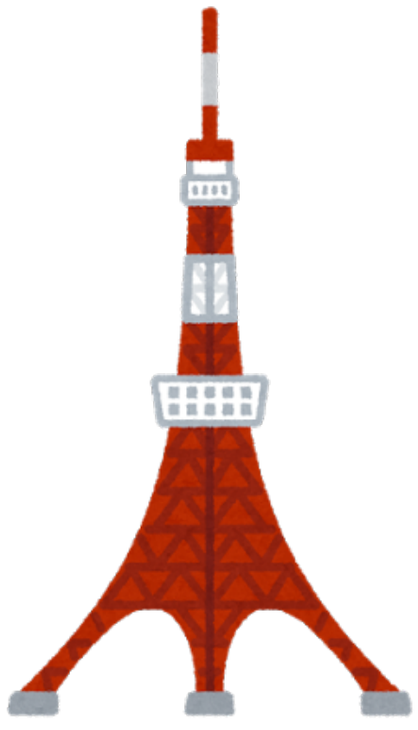
Encuesta de Salarios

..Ops + Golang + Python..


HDF®
Liberation of Technology

HDE®
Liberation of Technology

Tokyo



HDE®
Liberation of Technology

cool problems to solve

HDE®
Liberation of Technology

check our booth

..a bit of context…

Roughly 1.5 years ago…

I joined HDE

first week..started looking at projects..

first week..started looking at projects..

Github repos…

first week..started looking at projects..

Github repos…

New project I had to work on…

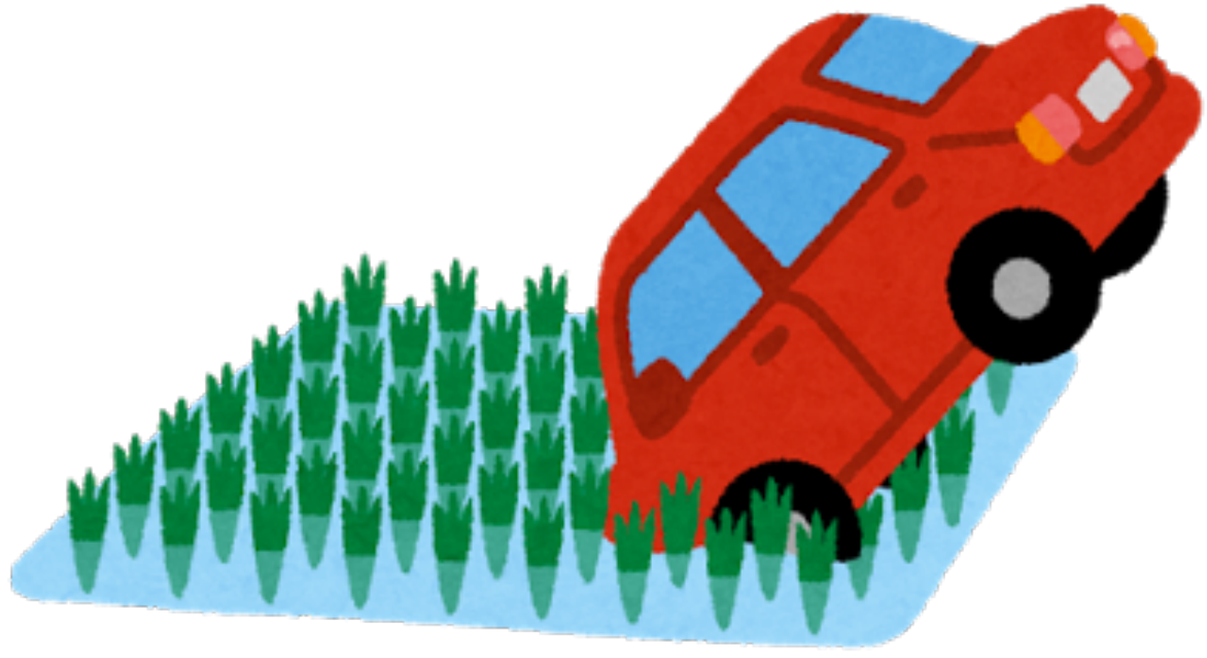
turns out some projects were built on “serverless”

so in this talk I will guide you through:

some of the tools

some of the use cases

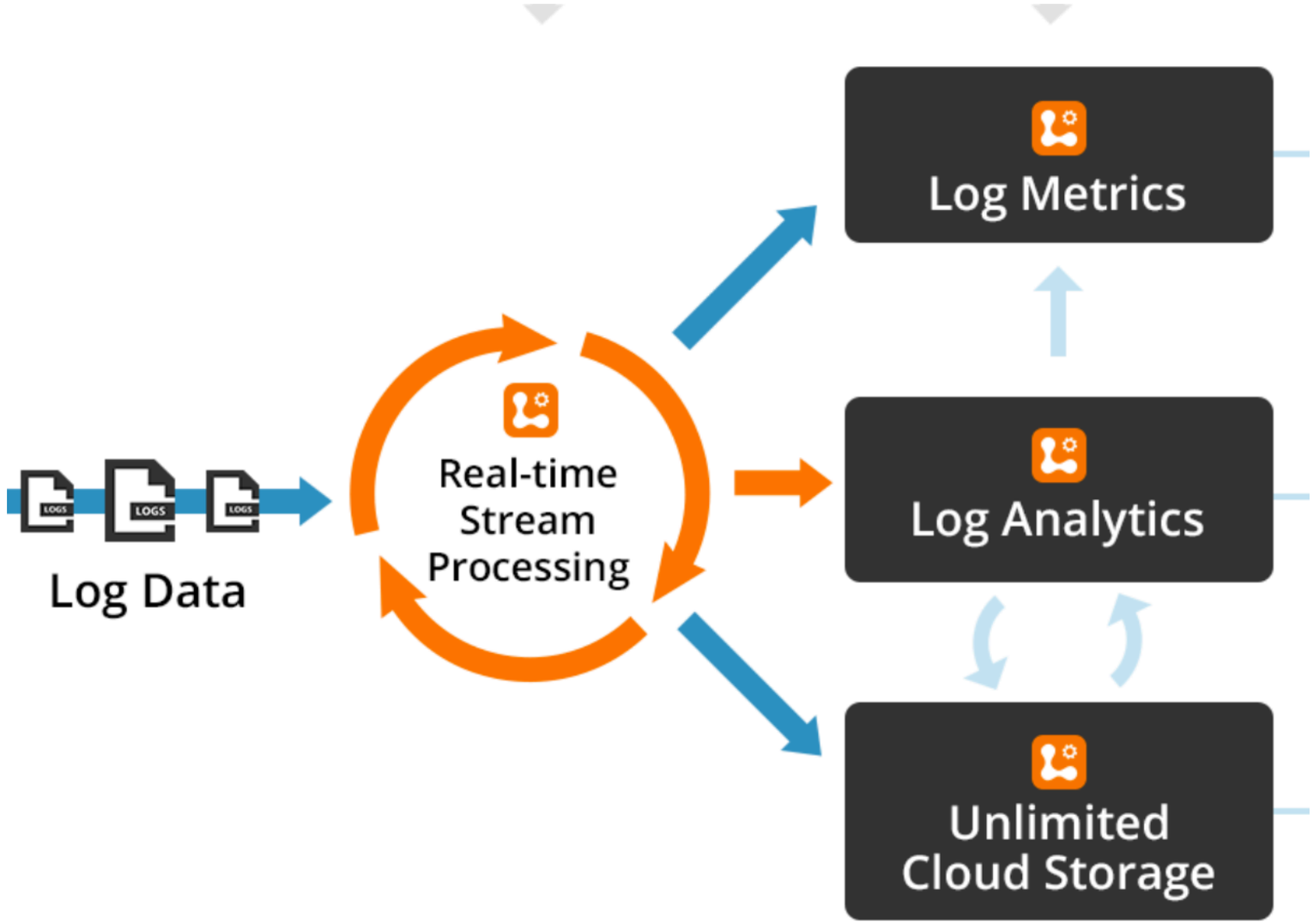
some common mistakes

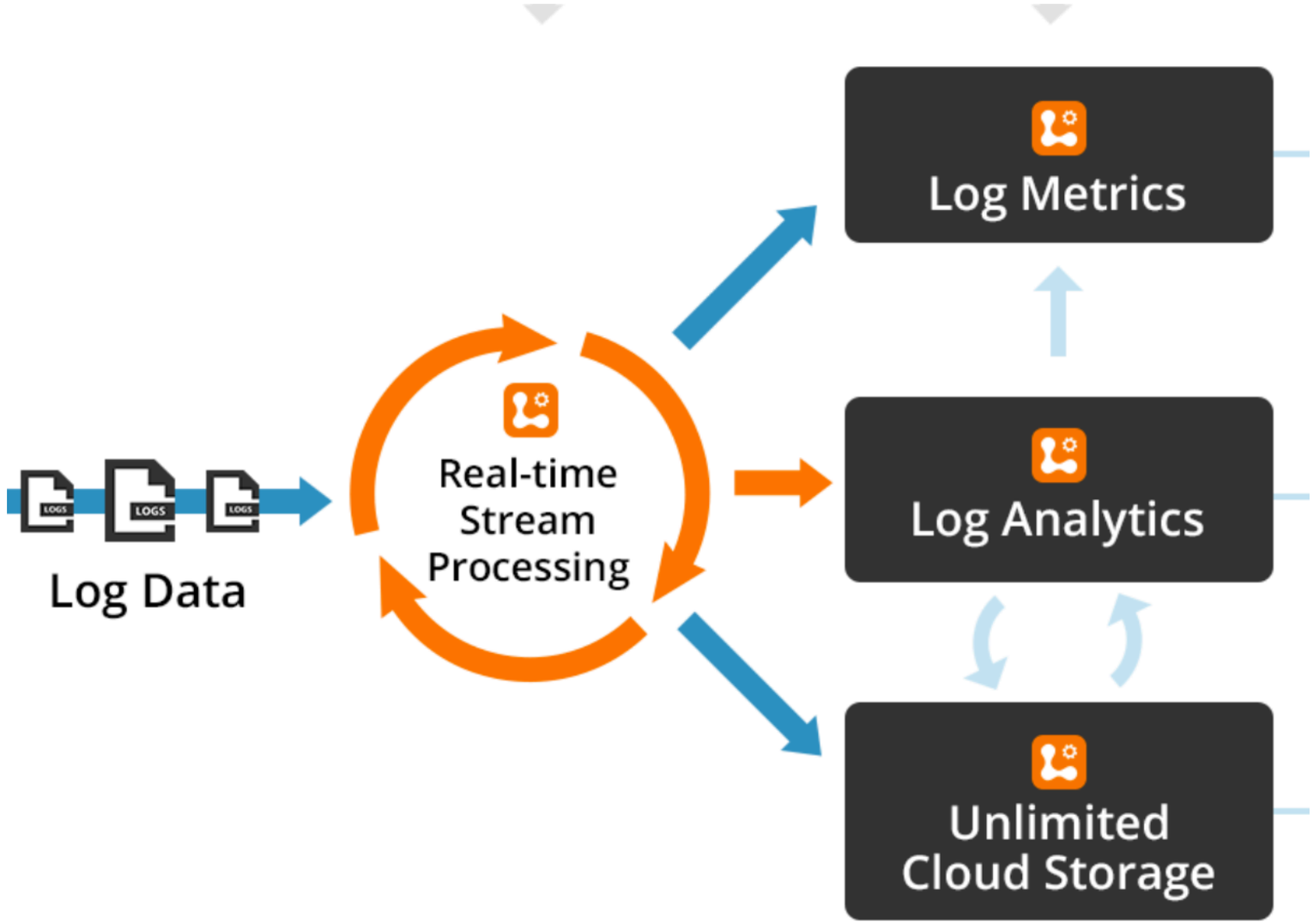


…my opinion…

my first task:

log stream processing


Log Metrics
Log Analytics
Unlimited
Cloud Storage
Real-time
Stream
Processing
LOGS
LOGS
LOGS
Log Data



near realtime

If you have been doing data pipelines, probably you would go for:

If you have been doing data pipelines, probably you would go for:

If you have been doing data pipelines, probably you would go for:

a lot to configure
a lot to manage

a lot to configure
a lot to manage

zookeeper..
yarn cluster..
driver instance..
JVM..
HDFS..

a lot to configure
a lot to manage

zookeeper..
yarn cluster..
driver instance..
JVM..
HDFS..

you have to make sure those parts are running smoothly

you have to make sure those parts are running smoothly

I met Jeff

I met Jeff

a coworker

ops engineer

He loves Serverless

And he has good reasons for it…

he does not want to do
any server management

So Jeff suggested to use serverless to solve our stream processing problem

I raised my eyebrow in doubt:

“Serverless”??

a quite controversial &
confusing term..

“Serverless”

1. Fully managed services, managed on your behalf

Databases (DynamoDB)..
Storage (s3)..
Queues (kafka as a service, sqs…)
heroku…

“Serverless”

2. Function as a Service (FaaS)

AWS lambda
Azure functions
Kubeless
Google Cloud functions..
Nuclio
..openwhisk….
…many more…….

Function as a service

Upload your code and it will run…

Upload your code and it will run…

you don’t care on top of what is running..

Upload your code and it will run…

you don’t care on top of what is running..

you only focus on your business logic

(i) zero administration

- Focus on a single function

zero administration

- Focus on a single function

- Managed by provider

zero administration

- Focus on a single function

- Managed by provider

- You gain peace of mind

zero administration

- Focus on a single function

- Managed by provider

- You gain peace of mind

- cost: more integration with your vendor(vendor lock-in)

(ii) You are billed by number of invocations

so how does this looks like in python?

```
def handler(event, context):
    # do something
    pass
```

```
def handler(event, context):
    # do something
    url = event['url']
    scrape(url)
```

```
def handler(event, context):
    # do something
    database = # magic
    username = event['username']
    database.find(username)
```

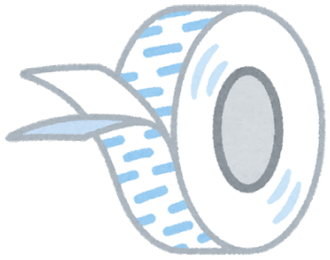
so how do I deploy my function?

in order run your project on FaaS:

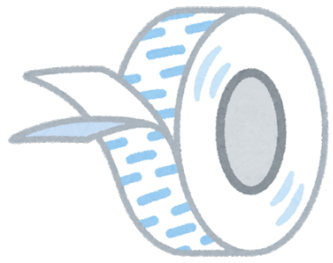
0. define your function
1. package your function
2. upload your package
3. call your function

tools address those steps

(tools) 0. define function (infrastructure / glue)

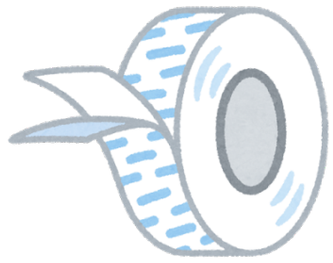


(tools) 0. define function (infrastructure / glue)



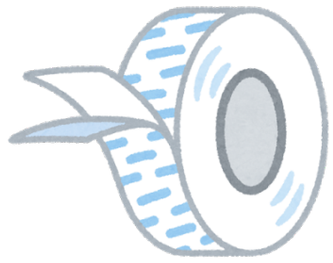
- runtime (python, golang, js..)

(tools) 0. define function (infrastructure / glue)



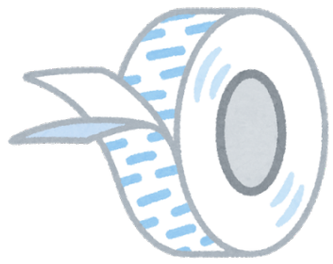
- runtime (python, golang, js..)
- memory (128mb, 500?..)

(tools) 0. define function (infrastructure / glue)



- runtime (python, golang, js..)
- memory (128mb, 500?..)
- access to resources

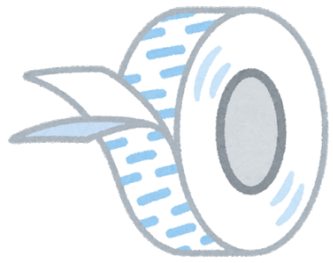
(tools) 0. define function (infrastructure / glue)



- runtime (python, golang, js..)
- memory (128mb, 500?..)
- access to resources

mouse & clicking

(tools) 0. define function (infrastructure / glue)

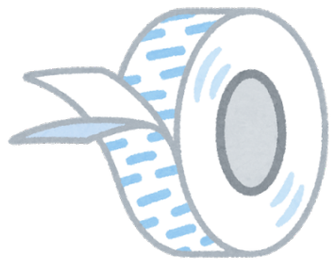


- runtime (python, golang, js..)
- memory (128mb, 500?..)
- access to resources

mouse & clicking

HashiCorp Terraform

(tools) 0. define function (infrastructure / glue)



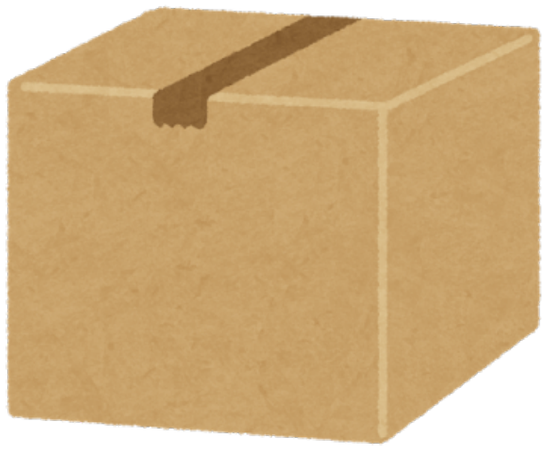
- runtime (python, golang, js..)
- memory (128mb, 500?..)
- access to resources

mouse & clicking

HashiCorp Terraform

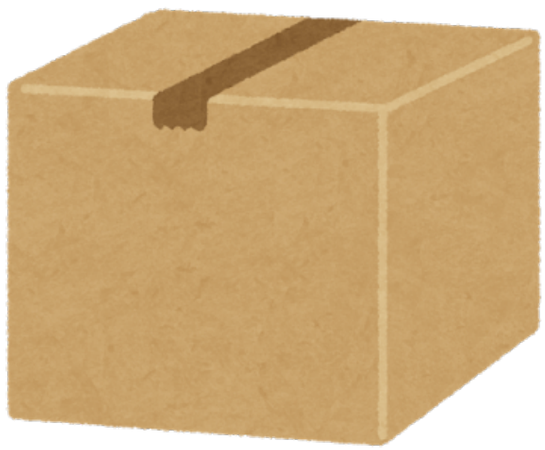
CloudFormation

1. package your function



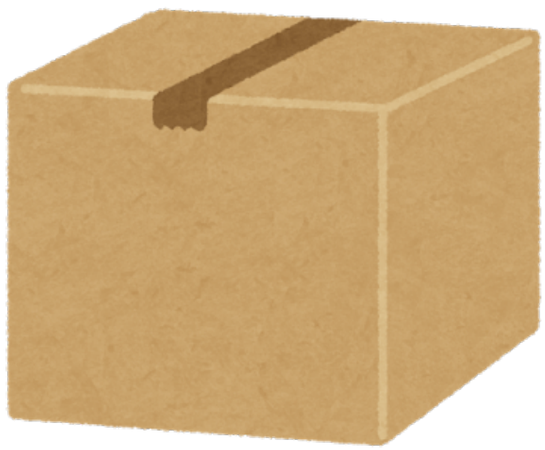
1. package your function

your function
code



1. package your function

your function code + dependencies



2. upload package

"deploying it"

3. call your function

3. call your function

“what triggers it?”

3. call your function

"what triggers it?"

function is called whenever:

3. call your function

“what triggers it?”

function is called whenever:

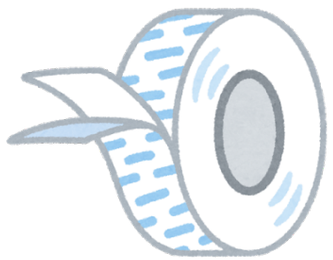
url gets hit..

an object is uploaded to s3..

a record is added to your database..

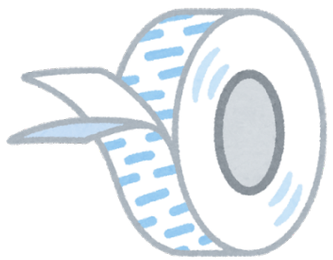
while queue is not empty..

3. call your function



Deeply entangled on your cloud provider services

3. call your function



Deeply entangled on your cloud provider services

(infrastructure /glue)

mouse & clicking

HashiCorp Terraform

CloudFormation

tooling address those steps

deploying:

infrastructure + code

back to my story..

back to my story..

first week..
started looking at repos..

One of them is an API

something.com/endpointA

something.com/endpointB

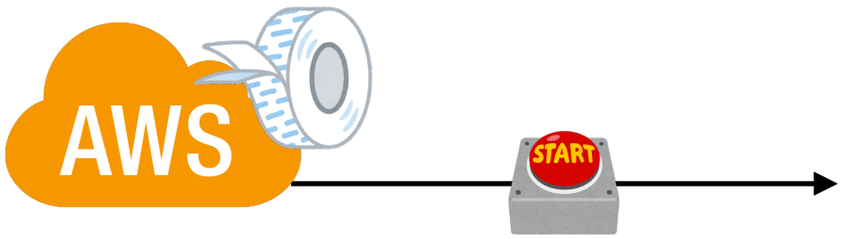
something.com/endpointC

my team was an early adopter of this technology

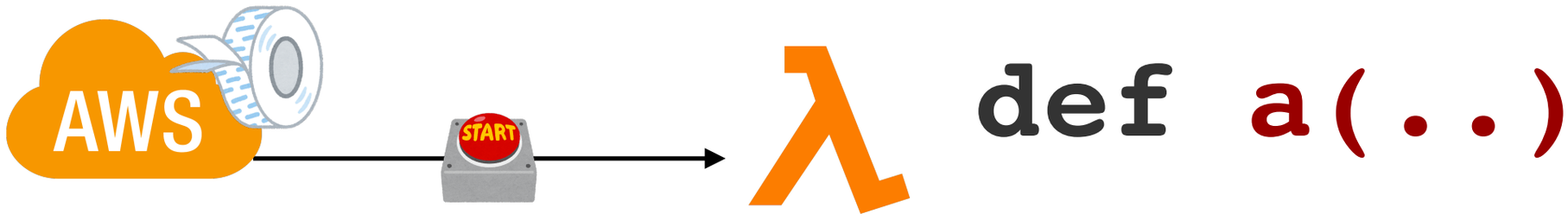
they stitched projects with the tools available back then

request to
urlA

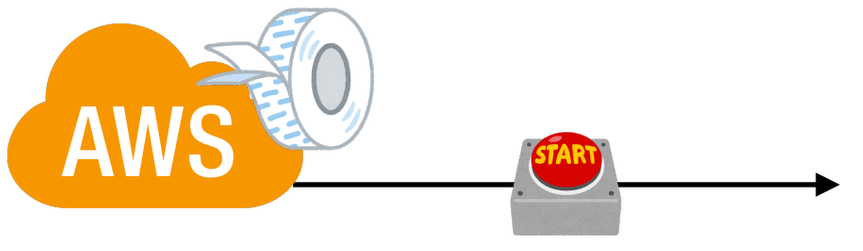
request to
urlA



request to
urlA

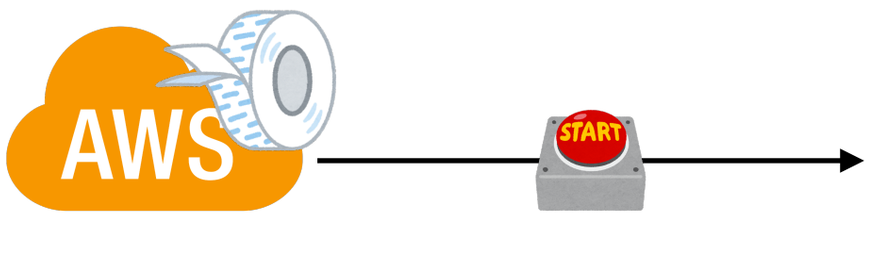


request to
urlA



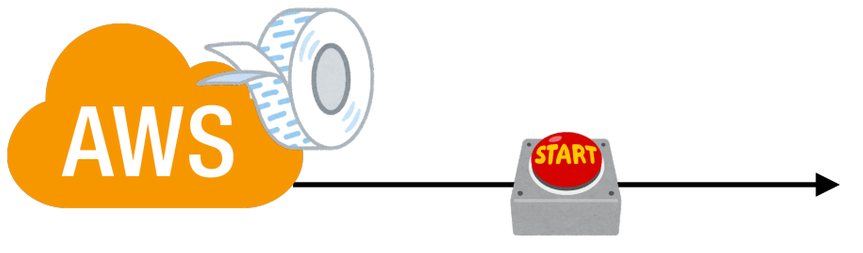
```
def a(..)
```

request to
urlB



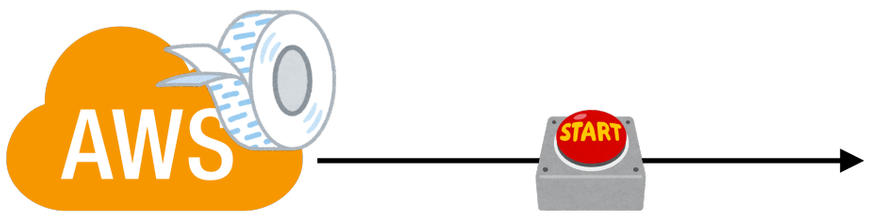
```
def b(..)
```

request to
urlA



```
def a(..)
```

request to
urlB



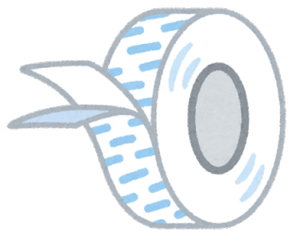
```
def b(..)
```

request to
urlC

```
def c(..)
```

deploy: makefiles

deploy: makefiles



glue:

HashiCorp
Terraform

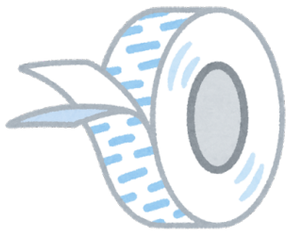
3 functions to deploy

3 functions to deploy

3 functions to package

3 functions to deploy

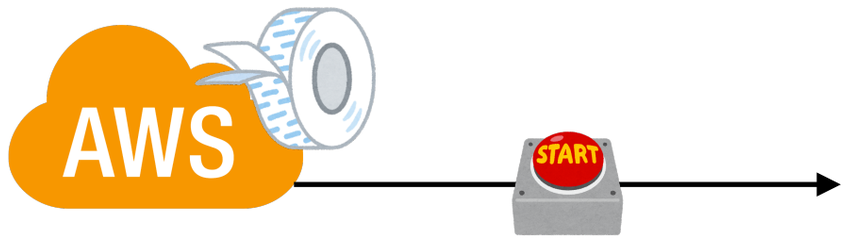
3 functions to package



HashiCorp Terraform

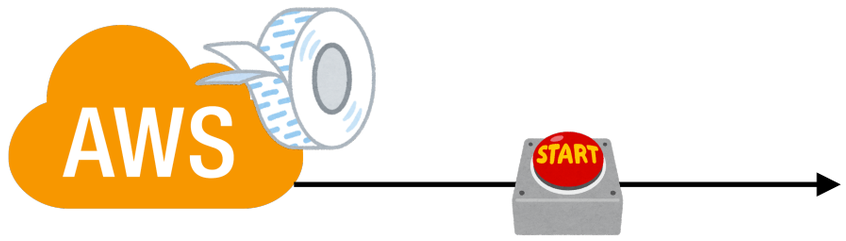
lots of glue :

- many pieces to move
- to worry about
- hard to test



HashiCorp Terraform

changing this kind of trigger(url)
on terraform is painful



HashiCorp Terraform

changing this kind of trigger(url)
on terraform is painful

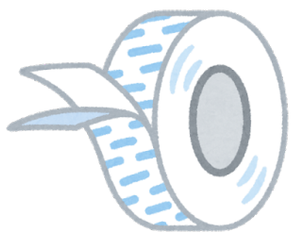
HashiCorp Terraform

running terraform is scary
you might destroy other infrastructure

if for whatever reason you want to move to non-serverless.. it is harder..

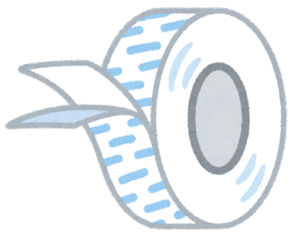
if for whatever reason you want to move to non-serverless.. it is harder..

why?



because all that glue that triggers the functions

you have to implement that glue in a different way

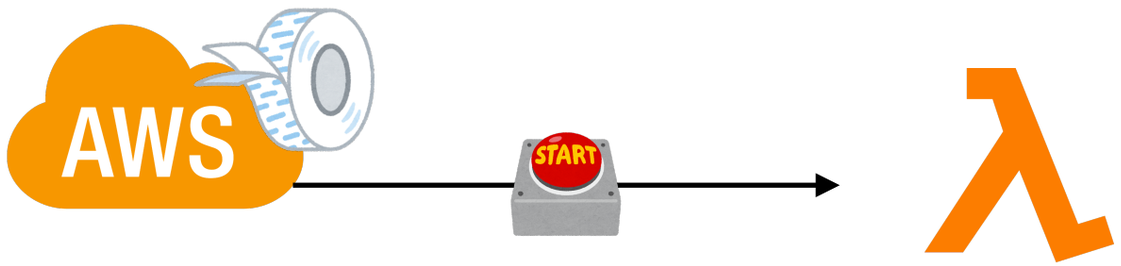


3 different lambdas..
also make people structure their project like 3 independent projects..

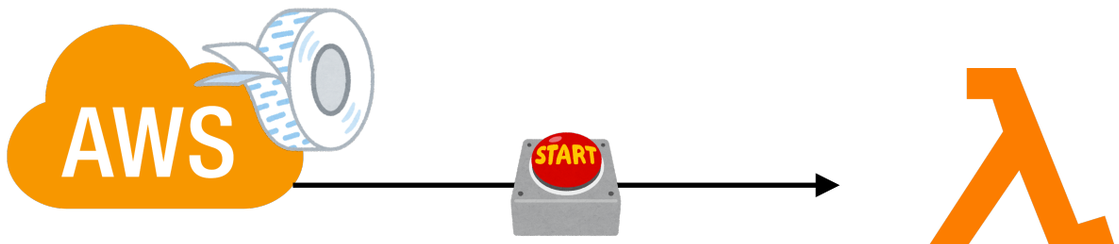
with tools available today

you can avoid this..

any request
to
something.com

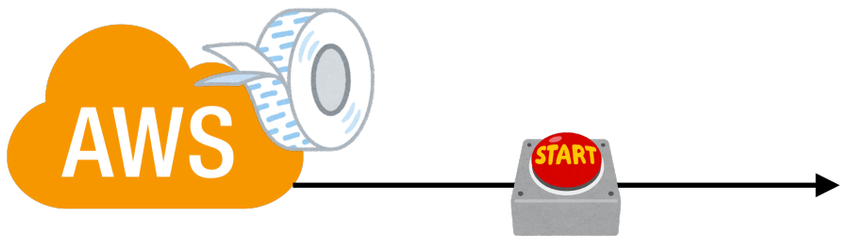


any request
to
something.com



wsgi wrapper
(like a flask app)

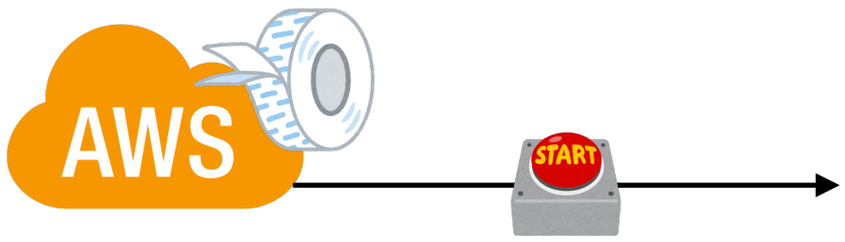
any request
to
something.com



wsgi wrapper
(like a flask app)

wsgi wrapper?

any request
to
something.com

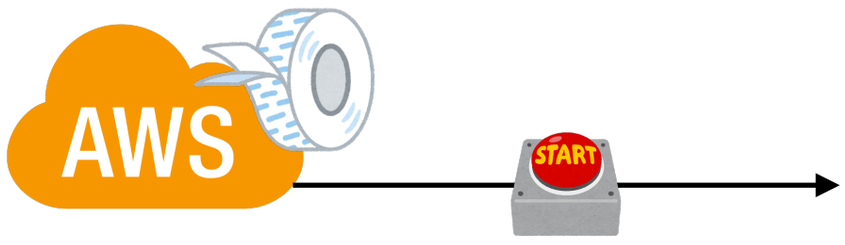


wsgi wrapper
(like a flask app)

wsgi wrapper?

translates requests arriving to **def handler(..)** into wsgi requests

any request
to
something.com



wsgi wrapper
(like a flask app)

wsgi wrapper?

translates requests arriving to **def handler(..)** into wsgi requests

it means we can use tools like **flask**, **django** ..

you can use all tools already available and all that come with them

testing is easier

you can use all tools already available and all that come with them

testing is easier

you can make your api with the same tools, and have serverless “for free”

you can use all tools already available and all that come with them

testing is easier

you can make your api with the same tools, and have serverless “for free”

minimal implementation of that wrapper (python3) is available at:

https://github.com/slank/awsgi

so if you have a dummy flask app

```
from flask import (
    Flask,
    jsonify,
)

app = Flask(__name__)
```

```
@app.route(‘/endpointB’)
def endpoint_b():
    # …
    return jsonify(status=200, message='OK')
```

to make it serverless you just need to add this function:

to make it server less you just need to add this function:

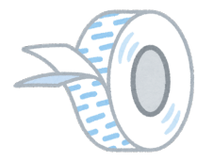
```
def handler(event, context):
    return awsgi.response(app, event, context)
```

to make it server less you just need to add this function:

```
def handler(event, context):
    return awsgi.response(app, event, context)
```

- can run it locally

- can run it locally
- easy to test routing

- can run it locally
- easy to test routing
- less glue 

API building serverless tools:

API building server less tools:

serverless

chalice

zappa

...

there are more..many..

Serverless

https://github.com/serverless/serverless
https://serverless.com/

serverless

serverless

Good:

- many plugins
- got funding
- particularly good when you are building apis
- it is not sluggish (for the given use case)
- provide you easy way to handle environments (stg, prd)
- great community

serverless

Bad:

no plan building:

serverless

Bad:

no plan building:

deploying:

infrastructure + code

serverless

Bad:

no plan building:

deploying:

infrastructure + code

it does not tell you what will change before deploying

serverless

Bad:

changing something in config file can leak infrastructure

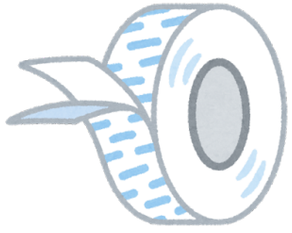
It is dangerous to leave infrastructure leaking behind

serverless

How to use it?

serverless

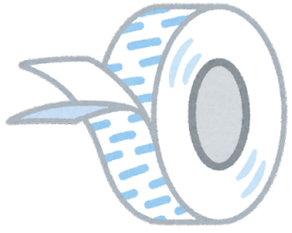
How to use it?



1. define glue in .yml file

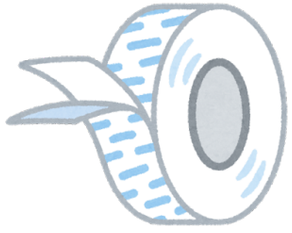
serverless

How to use it?



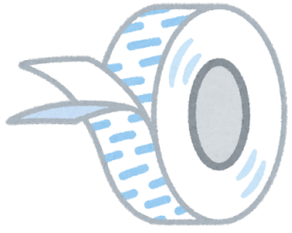
1. define glue in .yml file
2. do `sls deploy`

serverless



.yml file

serverless



.yml file

- I want a function with this memory..
- I want a function with this name…
- I want it to get called when X and Y happens

serverless

`sls deploy`

- Package + upload

serverless

serverless has a lot fo plugins that you can add to your .yml file.

don’t use it to manage infrastructure.

serverless

serverless “applications”

code + glue + infrastructure

i.e: serverless service to get a slack bot via FaaS

```
> serverless install --url <service-github-url>
> sls deploy
```

Chalice

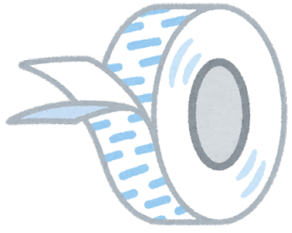
https://github.com/aws/chalice

```
> chalice new-project my_sample_project
> chalice deploy
```

- comes with “wsgi wrapper”
- purely focused on AWS
- aimed at API particular case

Zappa

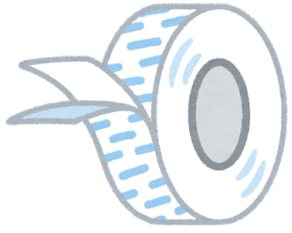
How to use it?



1. define glue in .json file

Zappa

How to use it?



1. define glue in .json file

2. some of the glue code is defined as python decorators

```
@task
```

```
    @task
    def make_pie():
        """ This takes a long time! """
        ingredients = get_ingredients()
        pie = bake(ingredients)
        deliver(pie)

    @task
    def make_soup():
        ingredients = get_ingredients()
        soup = bake(ingredients)
        deliver(soup)
```

Zappa

How to use it?

it has some cool decorators

it lacks plugins/addons

good

if you are building APIs

if you have lots of cron/async calls

Another use case..

you have a lot of data to process in batch

you have a lot of data to process in batch

text..
numbers..

count frequencies

clean wikipedia
etl
data preprocessing

compute to TB datasets across hundreds of functions

map & reduce..right?

map & reduce..right?

get me a spark cluster

use pyspark ..done

this is a great use case for FaaS

this is a great use case for FaaS

pywren

https://github.com/pywren/pywren

pywren

benchmark of

- 80 GB/sec read
- 60 GB/sec write

pywren

No knowledge of AWS required

pywren

No knowledge of AWS required

No large (expensive) cluster up

pywren

No knowledge of AWS required

No large (expensive) cluster up

Using vanilla python

pywren

> pywren-setup

pywren

> pywren-setup

essentially :

1. takes a python function (creates a FaaS)

pywren

> pywren-setup

essentially :

1. takes a python function (creates a FaaS)
2. takes data and uploads it to s3

pywren

> pywren-setup

essentially :

1. takes a python function (creates a FaaS)
2. takes data and uploads it to s3
3. runs your python function in parallel on the data uploaded to s3

pywren

```
def add_one(x):
    return x + 1
```

pywren

```
def add_one(x):
    return x + 1
```

```
[0, 1…9]
```

pywren

```
def clean_text(text):
    # clean text
    return cleaned_text
```

pywren

```
def clean_text(text):
    # clean text
    return cleaned_text
```

wikidump (100G)

pywren

```
def add_one(x):
    return x + 1
```

pywren

```
def add_one(x):
    return x + 1
```

creates lambda

pywren

```
def add_one(x):
    return x + 1
```

creates lambda

```
[0, 1…9]
```

pywren

```
def add_one(x):
    return x + 1
```

creates lambda

```
[0, 1…9]
```

uploads data to

part1

pywren

```
def add_one(x):
    return x + 1
```

creates lambda

```
[0, 1…9]
```

uploads data to

part1

part2

pywren

```
def add_one(x):
    return x + 1
```

creates lambda

```
[0, 1…9]
```

uploads data to

part1 

part2 

part3 

pywren

```
def add_one(x):
    return x + 1
```

```
[0, 1…9]
```

pywren

```
def add_one(x):
    return x + 1
```

```
add_one(0)
```

```
[0, 1…9]
```

pywren

```
def add_one(x):
    return x + 1
```

[0, 1…9]

add_one(0)

add_one(1)

pywren

```
def add_one(x):
    return x + 1
```

[0, 1…9]

add_one(0)

add_one(1)

add_one(2)

. . . . .

pywren

```
def add_one(x):
    return x + 1
```

[0, 1…9]

add_one(0)

add_one(1)

add_one(2)

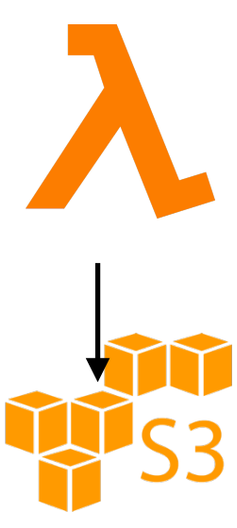
. . . . .

pywren

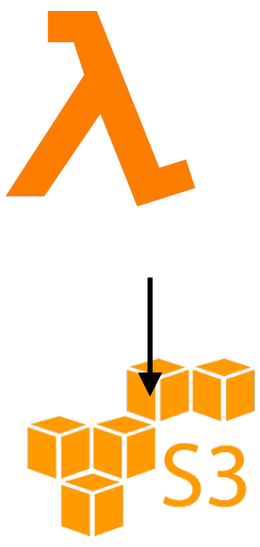
```
def add_one(x):
    return x + 1
```

[0, 1…9]

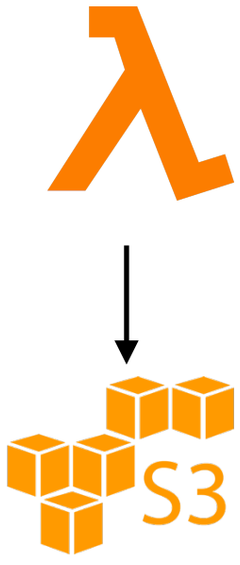
add_one(0)



add_one(1)



add_one(2)



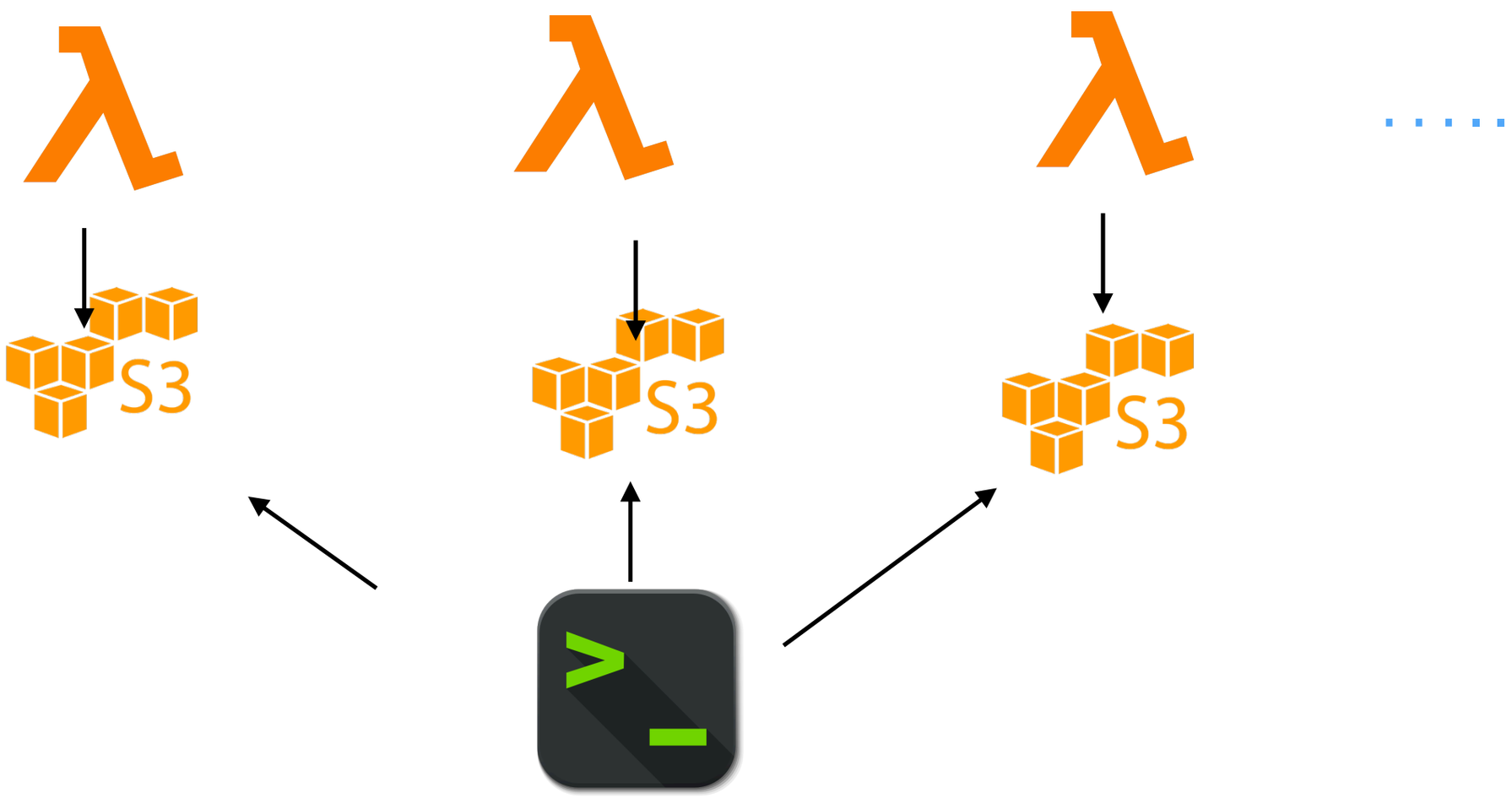
. . . . .

pywren

```
def add_one(x):
    return x + 1
```

[0, 1…9]

add_one(0) **add_one**(1) **add_one**(2)



pywren

how does it look like ?

```
import pywren

number_list = np.arange(10) # [0,1,2…9] data
```

pywren

how does it look like ?

```
import pywren

number_list = np.arange(10) # [0,1,2…9] data

# pywren magic
wrenexec = pywren.default_executor()
futures = wrenexec.map(addone, number_list)
```

pywren

how does it look like ?

```
import pywren

number_list = np.arange(10) # [0,1,2…9] data

# pywren magic
wrenexec = pywren.default_executor()
futures = wrenexec.map(addone, number_list)

# f.result() blocks until s3 file result is available
print [f.result() for f in futures]
```

pywren

how does it look like ?

```
import pywren

number_list = np.arange(10) # [0,1,2…9] data

# pywren magic
wrenexec = pywren.default_executor()
futures = wrenexec.map(addone, number_list)

# f.result() blocks until s3 file result is available
print [f.result() for f in futures]
```

> python sample.py

pywren

good for:

- ETL tasks..
- Scraping..
- Data crunching in general..

spark-on-lambda

https://github.com/qubole/spark-on-lambda

similar to pywren

spark-on-lambda

https://github.com/qubole/spark-on-lambda

similar to pywren

looks experimental though

spark-on-lambda

https://github.com/qubole/spark-on-lambda

similar to pywren

looks experimental though

same as spark..

just executors are lambda functions

spark-on-lambda

Scanning 1 TB of Data
1000 Lambda executors took 47s
cost turns out to be $1.18.

spark-on-lambda

Scanning 1 TB of Data
1000 Lambda executors took 47s
cost turns out to be $1.18.

---

on regular spark
50 r3.Xlarge instances..
2 or 3 mins just to setup + start

remarks..

no multiprocessing..
module

common mistakes

structure your project as any other python project

structure your project as

any other python project

don’t think of it as FaaS

if your project has many FaaS

if your project has many FaaS

it is a single project, with different entry points

immutability

FaaS is built on top of containers

containers for the same function sometimes gets reused..

```
db_connection = connect(something)
```

```
def handler(a,b):
 db_connection.query(something(a))
```

1st run

```
> db_connection = connect(something)
```

```
def handler(a,b):
 db_connection.query(something(a))
```

1st run

```
db_connection = connect(something)
```

```
> def handler(a,b):
  db_connection.query(something(a))
```

2nd run

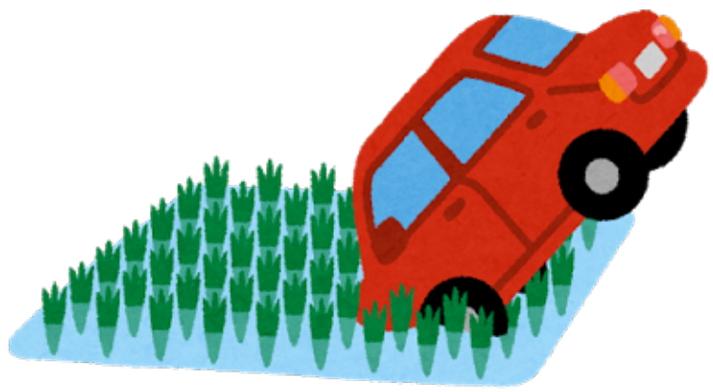
```
db_connection = connect(something)
```

```
> def handler(a,b):
  db_connection.query(something(a))
```

as in any project

as in any project

mutable globals are not desirable



```
list_of_users = [‘admin’]
def handler(a,b):
  list_of_users = list_of_users  + a[‘user’]
  do_something(list_of_users)
```

```
list_of_users = [‘admin’]
def handler(a,b):
  list_of_users = list_of_users  + a[‘user’]
  do_something(list_of_users)
```

```
list_of_users = [‘admin’]
def handler(a,b):
  list_of_users = list_of_users  + a[‘user’]
  do_something(list_of_users)
```

intended input to `do_something`

`[‘admin’, user1]`

actual input to `do_something`

`[‘admin’, user1]`

```
list_of_users = [‘admin’]
def handler(a,b):
  list_of_users = list_of_users  + a[‘user’]
  do_something(list_of_users)
```

intended input to `do_something`

`[‘admin’, user1]`

intended input to `do_something`

`[‘admin’, user2]`

actual input to `do_something`

`[‘admin’, user1]`

```
list_of_users = [‘admin’]
def handler(a,b):
  list_of_users = list_of_users  + a[‘user’]
  do_something(list_of_users)
```

intended input to `do_something`

`[‘admin’, user1]`

intended input to `do_something`

`[‘admin’, user2]`

actual input to `do_something`

`[‘admin’, user1]`

actual input to `do_something`

`[‘admin’, user1, user2]`

```
list_of_users = [‘admin’]
def handler(a,b):
  list_of_users = list_of_users  + a[‘user’]
  do_something(list_of_users)
```

intended input to `do_something`

`[‘admin’, user1]`

actual input to `do_something`

`[‘admin’, user1]`

intended input to `do_something`

`[‘admin’, user2]`

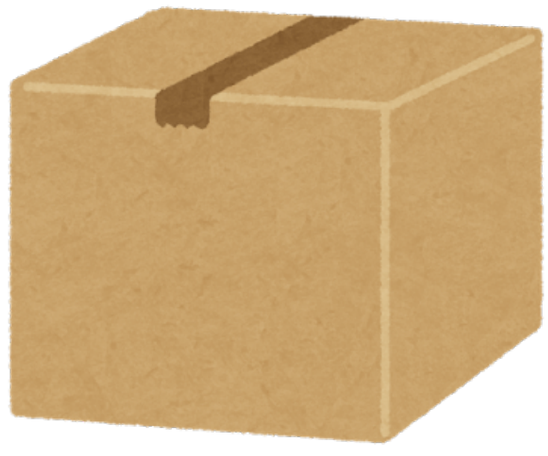
actual input to `do_something`

`[‘admin’, user1, user2]`

don’t mutate

..security..

Dependencies.. please update them



delete old functions

you don’t pay if you don’t use them
so you don’t get reminded in your bill

if you don’t use them, delete them

sounds easy..but..in a large organisation:
- you got no idea who is the owner

sounds easy..but..in a large organisation:

- you got no idea who is the owner
- is it safe to delete?

straight up your permissions

only give needed permissions

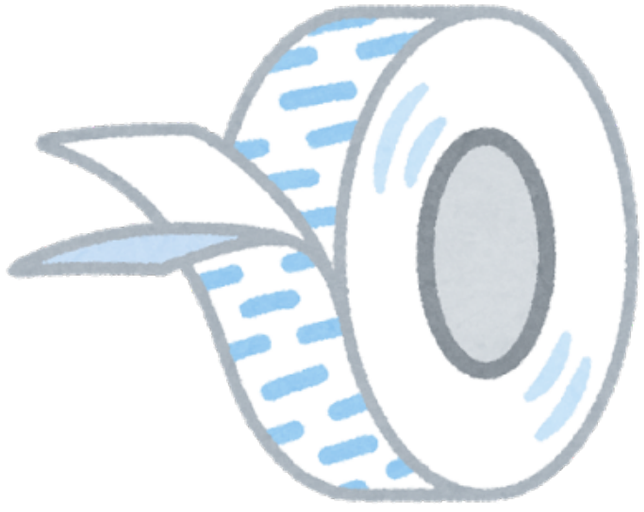
DDoS are wallet attacks

to sum up

lots of new tools are yet to come…

serverless provides a peace of mind.:
it will be running
it won't be down

but you agree on going full on using your cloud provider features



There is a lot of glue..
lots of events here and there….

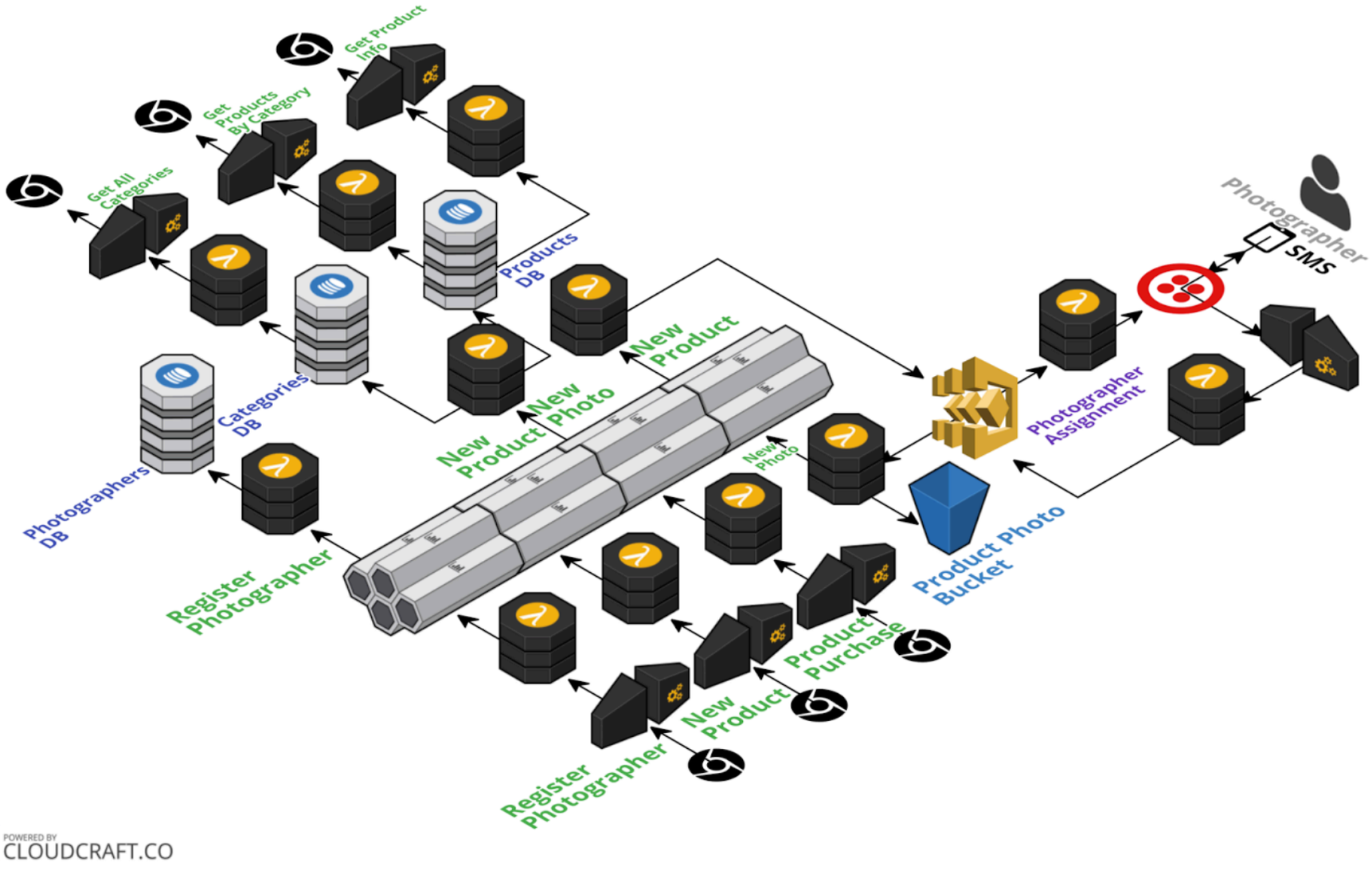
serverless is [in my opinion]

cheaper not simpler

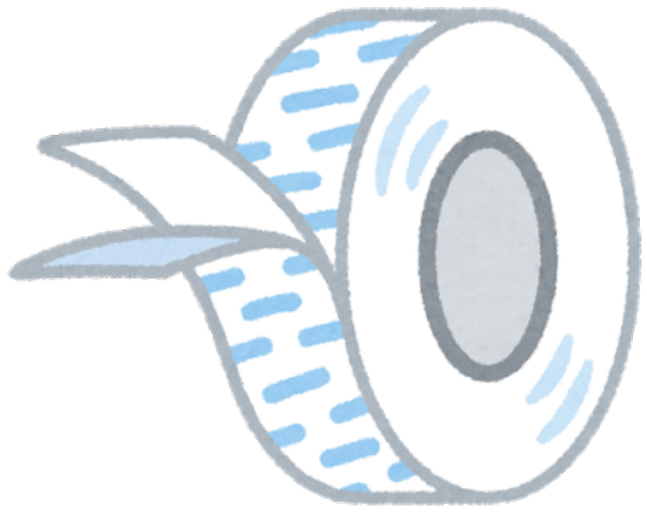
data crunching : yes

handling small events: yes

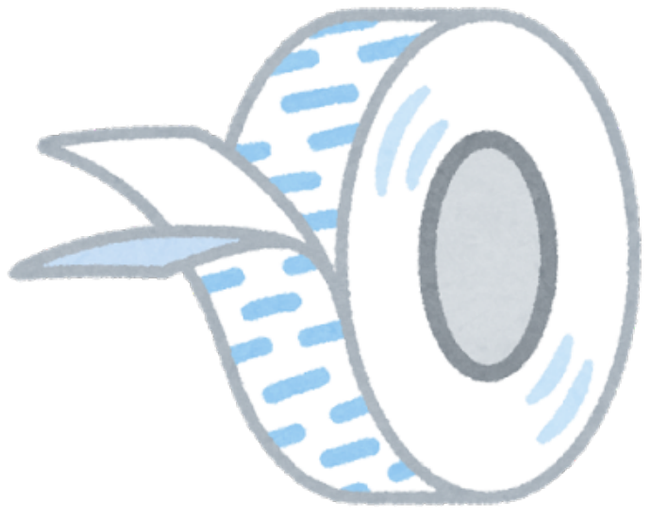
total complexity of your system might grow

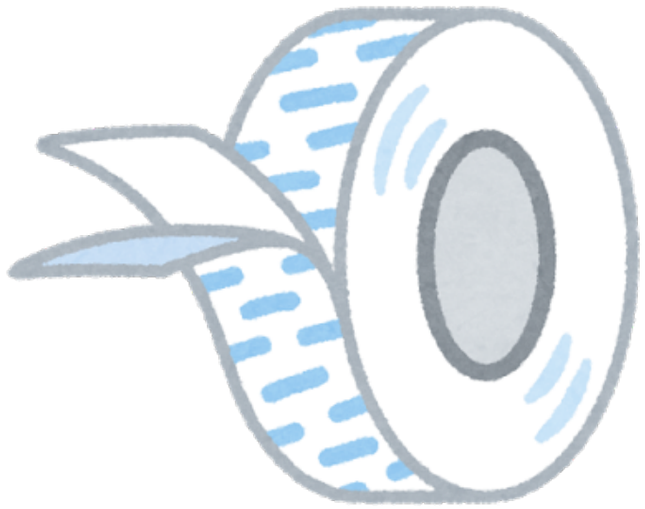

Get Product Info
Get Products By Category
Get All Categories
Products DB
Categories DB
Photographers DB
New Product
New Photo
New Product
New Photo
Register Photographer
Register Photographer
New Product
Product Purchase
Photographer Assignment
Photographer
SMS
Product Photo Bucket
POWERED BY
CLOUDCRAFT.CO

this is your extra complexity: glue, wiring



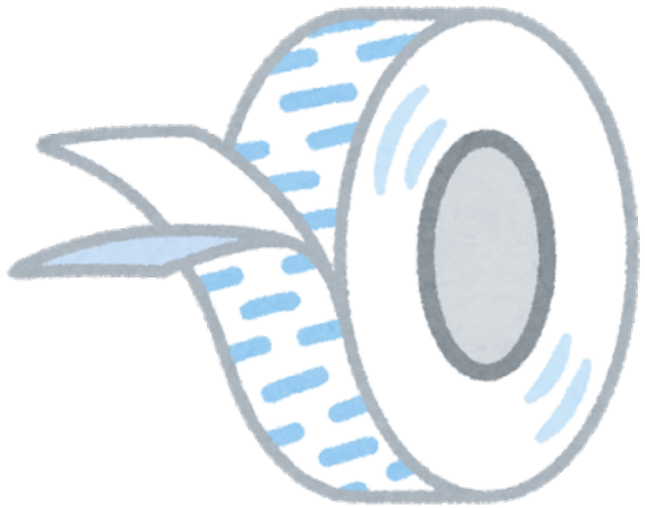
glue is hard to test





this is the hard part..

integration testing



does the event triggers the expected behaviour?

it is getting more popular..

many surveys seems to indicate FaaS adoption is as fast as that of containers

I guess new tools will help tame this complexity

I guess now you are wondering.

should I use serverless or not?

If you find yourself in any of these situations:

you have a developer complaining
about having to spin up infrastructure
before they can get something done

maybe you are a data scientist annoyed by how difficult is to run your experiment

then maybe is worth trying it

by the way, yes..

we built our stream processing system on top of FaaS

we built our stream processing system on top of FaaS

Gracias!

dav009

dav009