
# What is Serverless

...

## and how to live with it

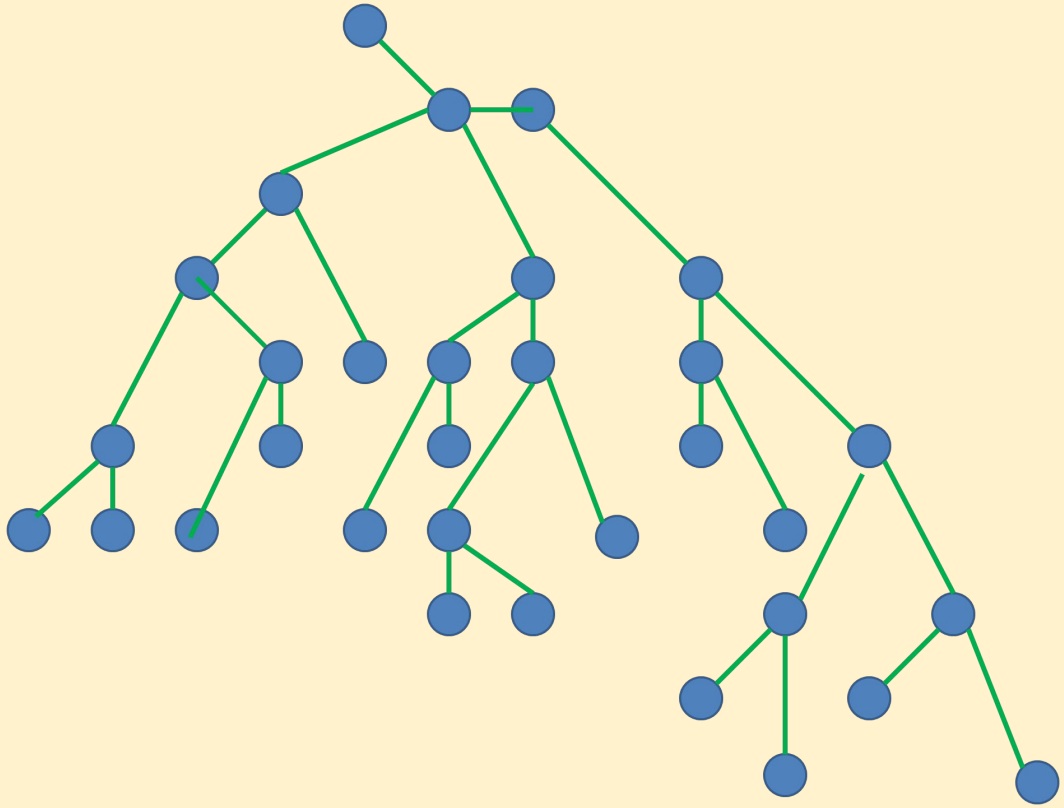
Nikolay Markov, 2017

# Shameless Plug

- My name is Nikolay Markov
- Senior Data Engineer at Aligned Research Group
- Used Python for 6+ years
- PyData Moscow Organizer (http://meetup.com/PyData-Moscow/)
- Python, C++, Scala and FP are good, everything with “java” in its title is bad, haven’t decided about Go yet

# Pipelines (+ ETL's)

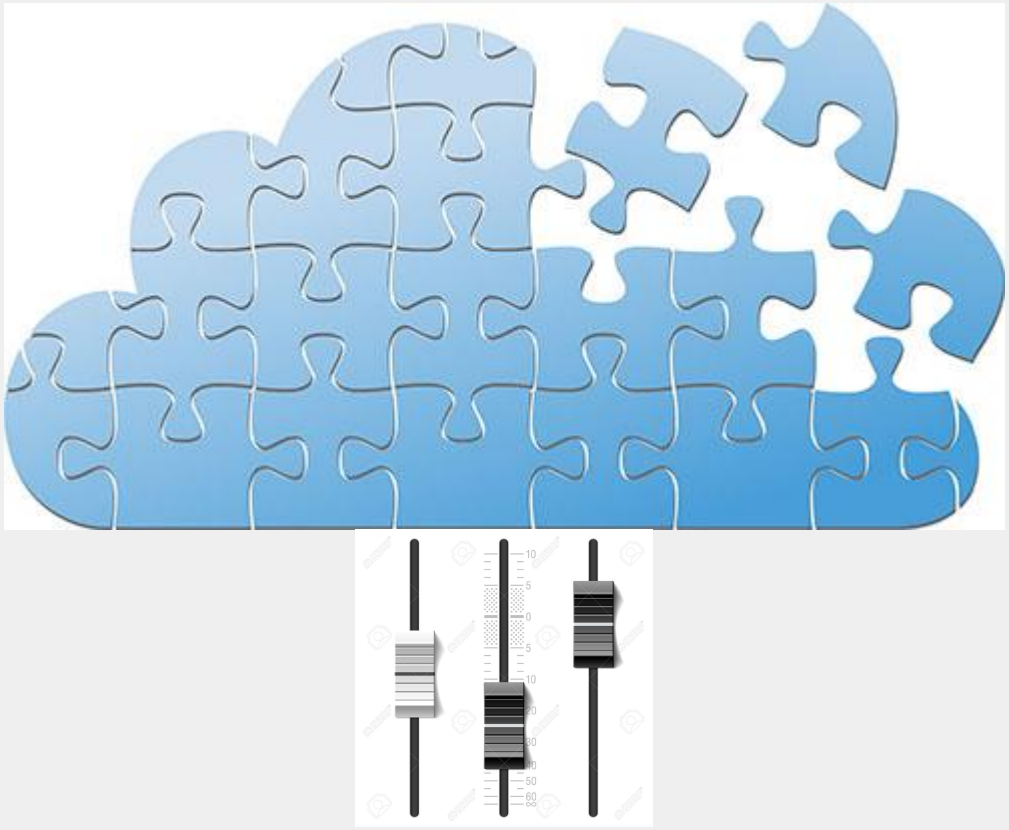
- Airflow/Luigi/Jenkins
- Bash
- RabbitMQ/Apache Kafka
- SQL
- MongoDB/HBase
- ELK
- …
- PROFIT



# Enough marketing words!

Let’s talk about Clouds, Big Data and Microservices instead!

# Let’s get ourselves some cloud



- Move the slider - get the resources
- Cut the cloud into pieces (VMs)
- Now let’s have DevOps guys to support them…
- You see where this is going, right?

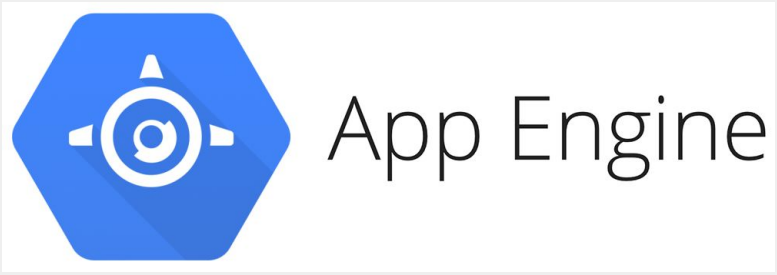
# So, what is Serverless then?

- An application that significantly or fully depend on 3rd party cloud-based applications/services to manage server-side logic and state (Backend as a Service).
- Parts of a business logic run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a 3rd party (Function as a Service).

https://martinfowler.com/articles/serverless.html

# Typical cases: API

- Someone or something is querying your service
- You do some background magic and return the result


# Typical cases: Storage

- Object storage
- Document storage
- Analytic storage
- BI/Data Warehouse


IBM Cloud
Object Storage

# Typical cases: Mobile/IoT

- Sending messages and notifications
- Collecting data from a network of devices
- Launch events directly on devices
- Build cross-platform apps and firmwares




# Typical cases: CI/CD and Security

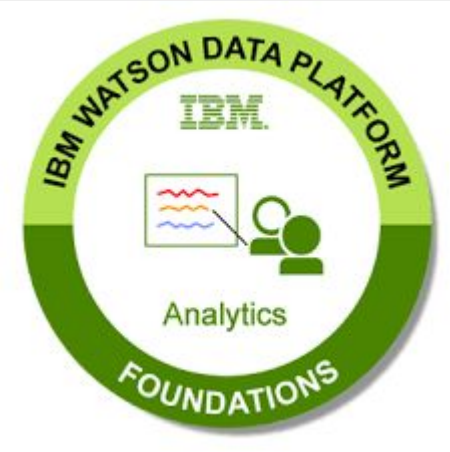
- Run tests
- Simulate user traffic
- Security analysis
- Build packages
- Roll out updates

tenable.io™

Loadrunner

CloudTest

# Typical cases: Distributed Computing





# *aaS pandemia





# FaaS to rule them all




CLOUD FUNCTIONS

Create small, single-purpose functions that respond to events in the cloud

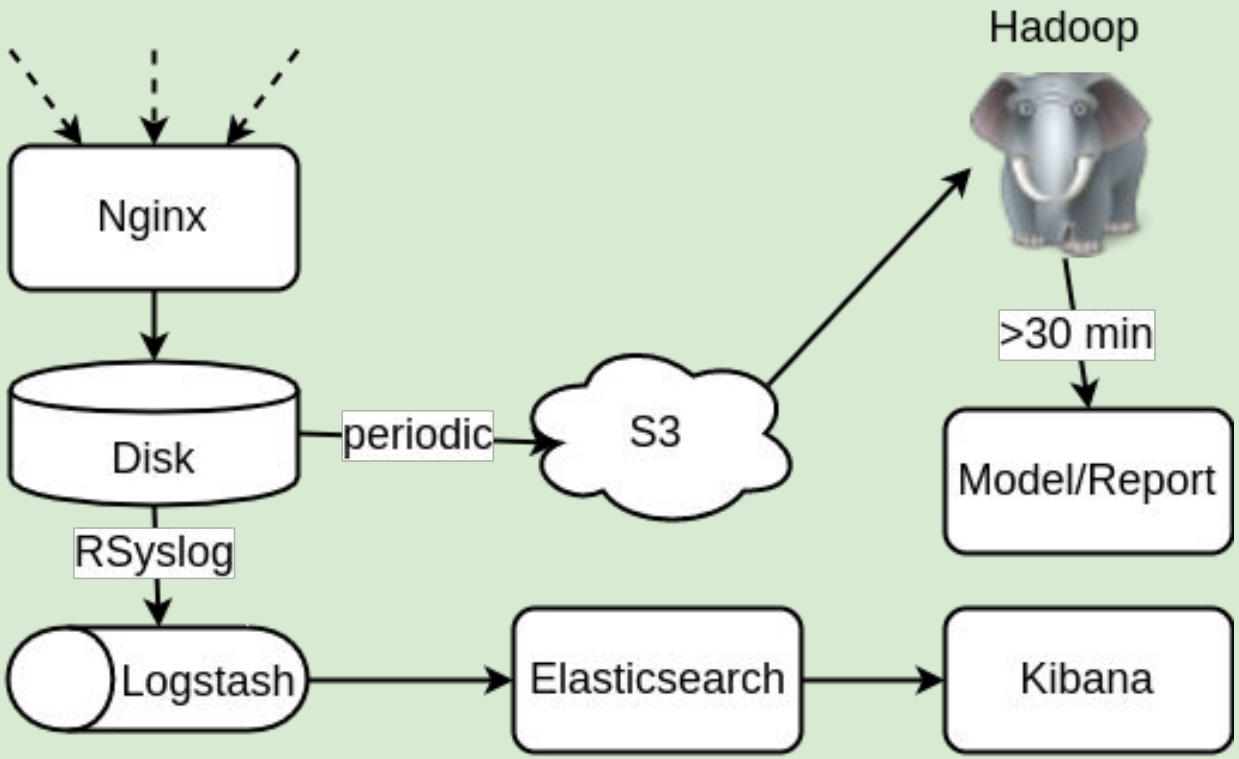
# Perks and advantages

- Decrease the load on DevOps
- Pay per usage time
- Just write your business logic

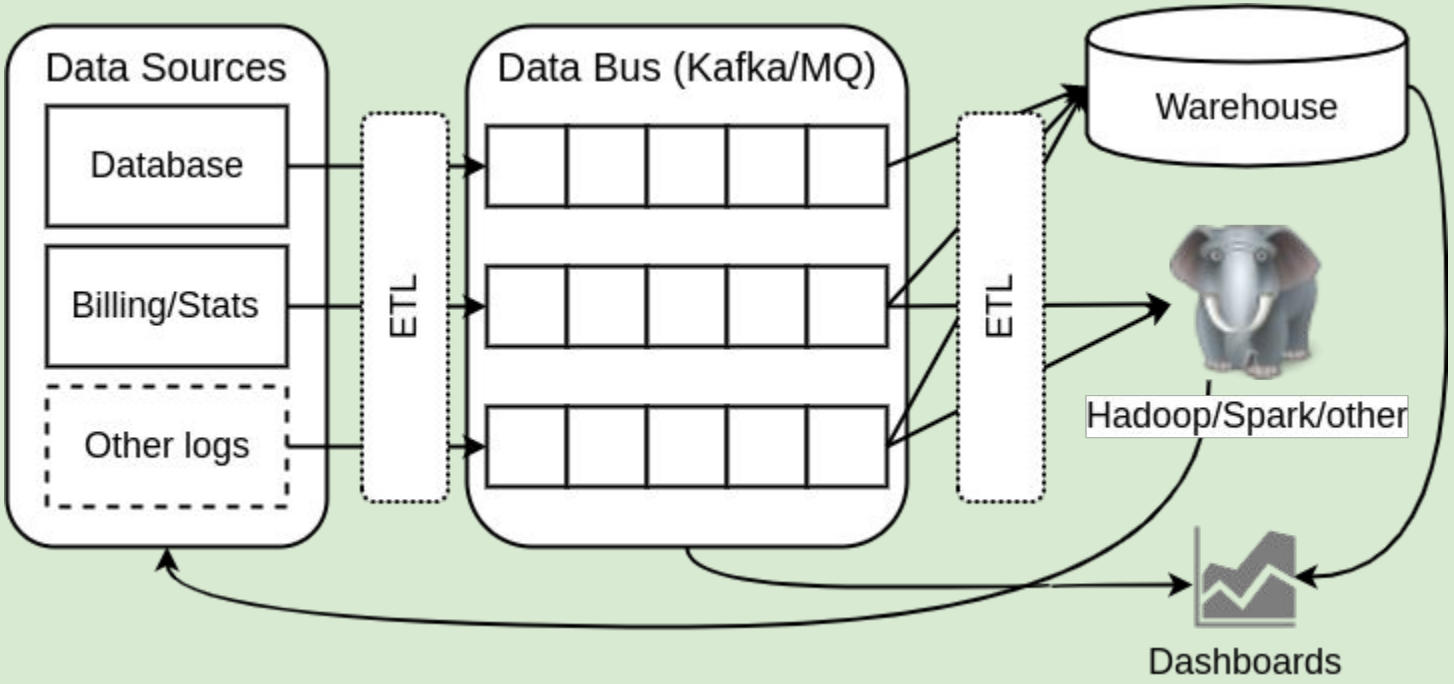
# Bad stuff

- Tied to a particular vendor
- May become expensive at some point
- Limited resources

# More than 1 hour to get results? Perfect!



# More streamy-like should do it, right?

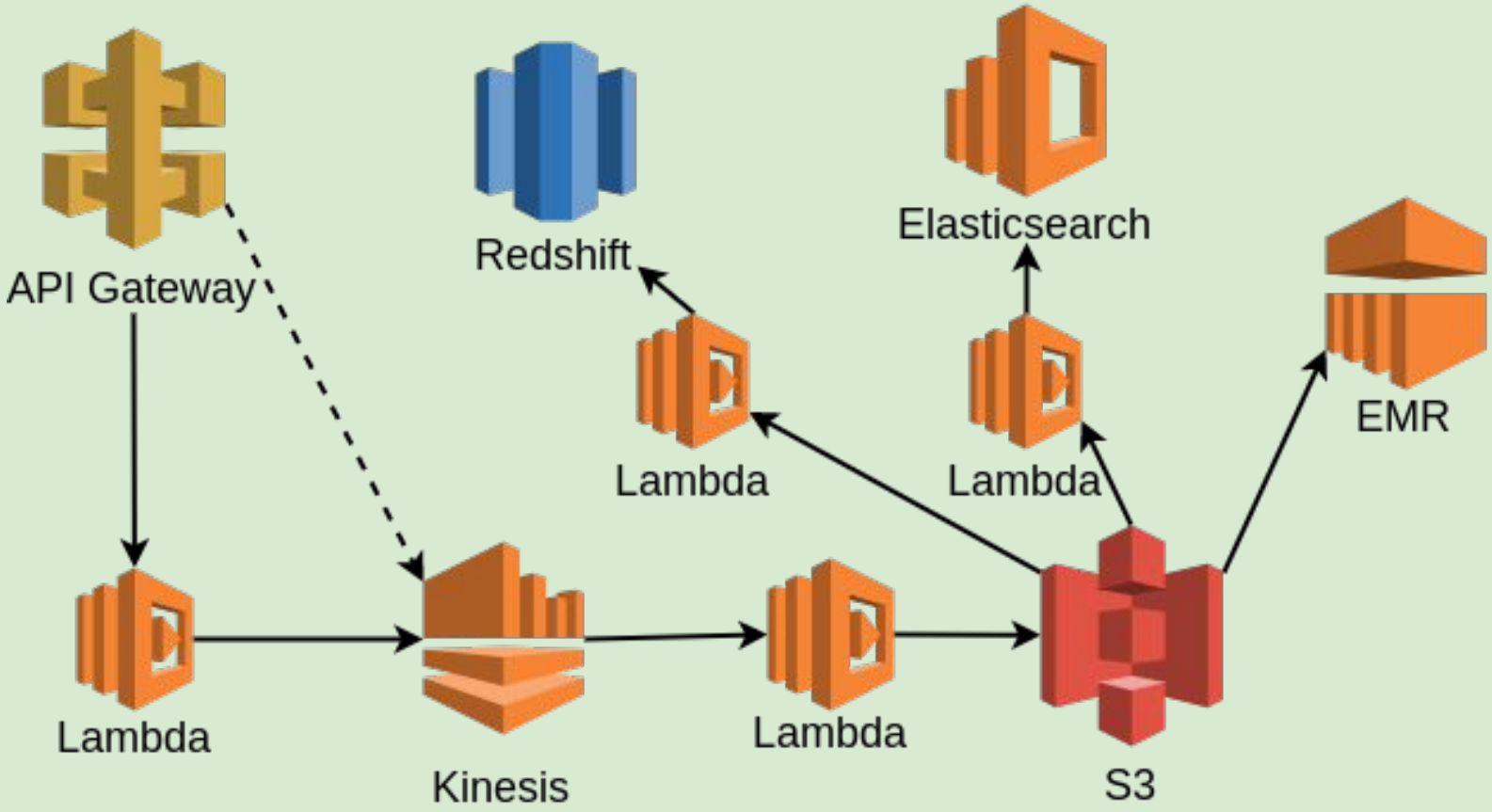


# Bash pipe

*~$ sleep 3 | echo “OK”*

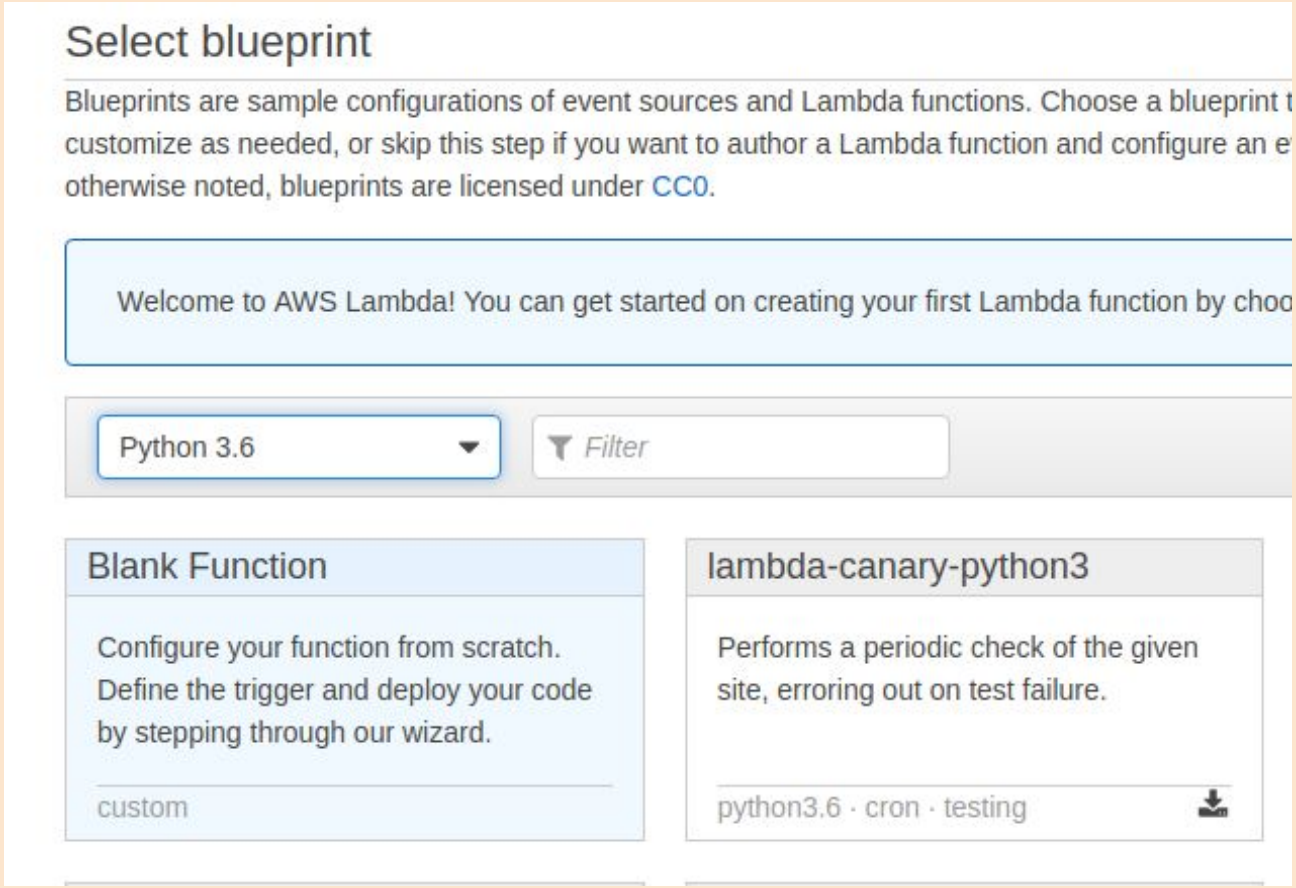
Link to my Bash pipeline talk slides (in Russian): http://bit.ly/2tfdUCG

# To stream or not to stream?



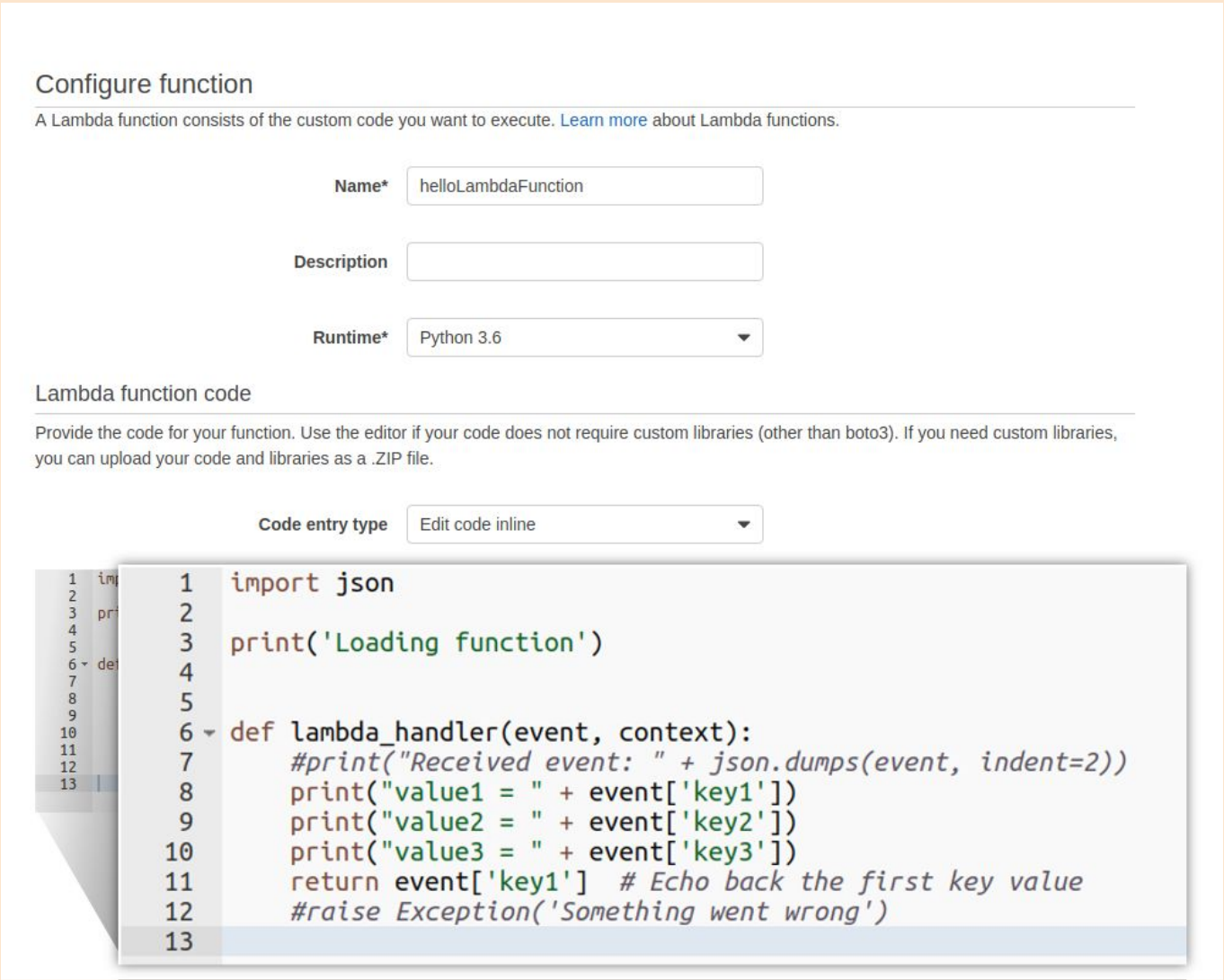
# Let's run some code!

1.



# Let's run some code!

2.



# Let's run some code!

3.

```
(aws) meow-nofer@pikachu ~ aws lambda invoke --invocation-type RequestResponse
 --function-name helloLambdaFunction --payload '{"key1": "hello", "key2": "world", "k
ey3": "!"}' outfile
{
    "StatusCode": 200
}
(aws) meow-nofer@pikachu ~ cat outfile
"hello"%
(aws) meow-nofer@pikachu ~
```

Filter events | all 30s 5m 1h 6h 1d 1w custom

| Time (UTC +00:00) | Message |
|---|---|
| 2017-07-09 | |
| | *No older events found at the moment.* Retry. |
| 13:09:06 | Loading function |
| 13:09:06 | START RequestId: c92d6180-64a7-11e7-af42-4110c810b6f4 Version: $LATEST |
| 13:09:06 | 'key1': KeyError Traceback (most recent call last): File "/var/task/lambda_function.py", line 8, in lambda_handler print("value1 = " + event['key1']) KeyError: 'key |
| 13:09:06 | END RequestId: c92d6180-64a7-11e7-af42-4110c810b6f4 |
| 13:09:06 | REPORT RequestId: c92d6180-64a7-11e7-af42-4110c810b6f4 Duration: 0.67 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 21 MB |
| 13:10:09 | START RequestId: c92d6180-64a7-11e7-af42-4110c810b6f4 Version: $LATEST |
| 13:10:09 | 'key1': KeyError Traceback (most recent call last): File "/var/task/lambda_function.py", line 8, in lambda_handler print("value1 = " + event['key1']) KeyError: 'key |
| 13:10:09 | END RequestId: c92d6180-64a7-11e7-af42-4110c810b6f4 |
| 13:10:09 | REPORT RequestId: c92d6180-64a7-11e7-af42-4110c810b6f4 Duration: 5.49 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 25 MB |
| 13:10:31 | START RequestId: fc3d484f-64a7-11e7-b29b-4f99709ec728 Version: $LATEST |
| 13:10:31 | value1 = hello |
| 13:10:31 | value2 = world |
| 13:10:31 | value3 = ! |
| 13:10:31 | END RequestId: fc3d484f-64a7-11e7-b29b-4f99709ec728 |
| 13:10:31 | REPORT RequestId: fc3d484f-64a7-11e7-b29b-4f99709ec728 Duration: 1.29 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 25 MB |

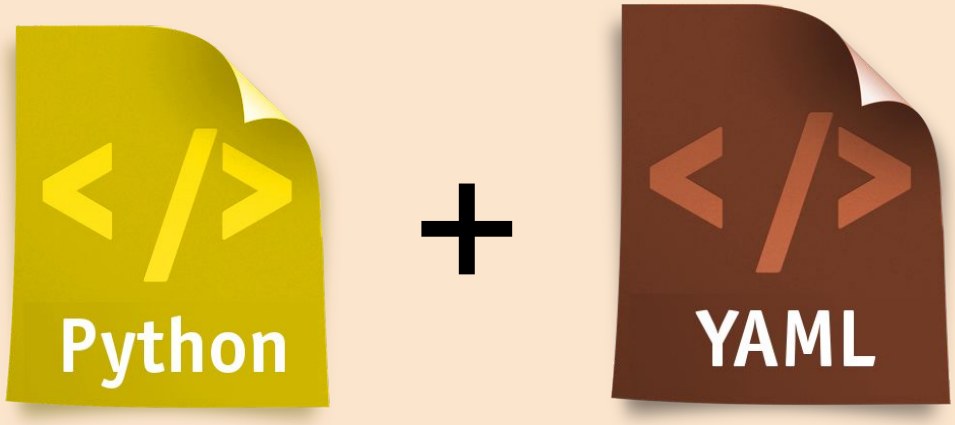
# Events and triggers

- Write code and pack it with dependencies
- Bind to certain events
- Configure security policies
- …
- Manually it’s kinda hard

# You need a framework!


Chalice


# Here's how it looks




Serverless:
*~$ sls create -t aws-python3*

```
.
├── handler.py
└── serverless.yml

0 directories, 2 files
```

Apex:
*~$ apex init*

```
.
├── functions
│   └── hello
│       └── index.js
└── project.json

2 directories, 2 files
```

*(+ .tf files for Hashicorp Terraform)*

# Here's how it looks

Serverless:

```
service: aws-python3 provider:
  name: aws
  runtime: python3.6
functions:
  hello:
    handler: handler.do_stuff
    events:
      - http:
          path: items/{item_id}
          method: get
```

Apex:

```
{
  "name": "mycoolproject",
  "description": "My cool
project that does stuff",
  "runtime": "python3.6",
  "memory": 128,
  "timeout": 5,
  "role":
"arn:aws:iam::SECRET:role/mycool
project_lambda_function",
  "environment": {}
}
```

All you need after that is "import boto3", write magic and "sls deploy" or "apex deploy"

# Pipeline Example: API to Kinesis to S3

1. Create API entry points and Kinesis stream
2. Create roles for our lambdas:
    a. With write policy for Kinesis and log access
    b. With read policy for Kinesis, log access and S3 bucket access
3. Write two lambda functions
4. Frustrate then everything fails
5. Relax
6. Think
7. Fix, redeploy - it works!
8. Aaand it’s already evening.

# Pipeline Example: API to Kinesis

```
import boto3
import json
import logging

kns = boto3.client('kinesis')
kns_stream = 'api_test_events'
kns_partition = 'api_test_partition'
logger = logging.getLogger()
```

```
def event_handler(event, context):
    try:
        kns.put_record(
            StreamName=kns_stream,
            Data=json.dumps(event),
            PartitionKey=kns_partition
        )
        return {
          "statusCode": 200,
          "headers": {"Content-Type": "application/json"},
          "body": "success"
        }
    except Exception as exc:
        err = (
            f"Failed to submit event to Kinesis "
            "(stream '{kns_stream}', partition
'{kns_partition}'): {exc}"
        )
        logger.error(err)
        return {
          "statusCode": 400,
          "headers": {"Content-Type": "application/json"},
          "body": err
        }
```

# Pipeline Example: Kinesis to S3

```
import base64
import datetime
import json

import boto3

s3 = boto3.client('s3')
```

```
def event_handler(event, context):
    events = []
    for rec in event['Records']:
        data = base64.b64decode(rec['kinesis']['data'])
        events.append(
            json.loads(
                json.loads(data.decode("utf-8"))["body"]
            )
        )

    now = datetime.datetime.utcnow()

    s3.put_object(
        Bucket="pycon-test-lambda-bucket",
        Key=(
            "{}/{}/{}/pycon_{}.json".format(
                now.year,
                now.month,
                now.day,
                now.strftime("%Y-%m-%d_%H:%M")
            )
        ),
        Body=json.dumps(events)
    )
```

# Pipeline Example: Serverless config: Functions

```
service: testKinesis2S3Workflow

provider:
  name: aws
  runtime: python3.6
  region: us-west-1

functions:
  api_to_kinesis:
    role: lambdaAPI2Kinesis
    handler: api_to_kinesis.event_handler
    events:
      - http:
          path: kns/submit
          method: post
  kinesis_to_s3:
    role: lambdaKinesis2S3
    handler: kinesis_to_s3.event_handler
    events:
      - stream:
          arn: arn:aws:kinesis:us-west-1:140461132978:stream/api_test_events
          batchSize: 3
          startingPosition: LATEST
          enabled: true
```

# Pipeline Example: Serverless config: Permissions

```
resources:
  Resources:
    lambdaAPI2Kinesis:
      Type: AWS::IAM::Role
      Properties:
        RoleName: lambdaAPI2Kinesis
        Path: "/"
        AssumeRolePolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Principal:
                Service:
                  - lambda.amazonaws.com
              Action: sts:AssumeRole
        ManagedPolicyArns:
          - arn:aws:iam::aws:policy/AmazonKinesisFullAccess
          - arn:aws:iam::aws:policy/CloudWatchFullAccess
```

```
    lambdaKinesis2S3:
      Type: AWS::IAM::Role
      Properties:
        RoleName: lambdaKinesis2S3Role
        Path: "/"
        AssumeRolePolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Principal:
                Service:
                  - lambda.amazonaws.com
              Action: sts:AssumeRole
        ManagedPolicyArns:
          - arn:aws:iam::aws:policy/AmazonKinesisReadOnlyAccess
          - arn:aws:iam::aws:policy/CloudWatchFullAccess
        Policies:
          - PolicyName: PyconTestBucketAccess
            PolicyDocument:
              Version: '2012-10-17'
              Statement:
                - Effect: Allow
                  Action:
                    - s3:PutObject
                  Resource: arn:aws:s3:::pycon-test-lambda-bucket/*
```

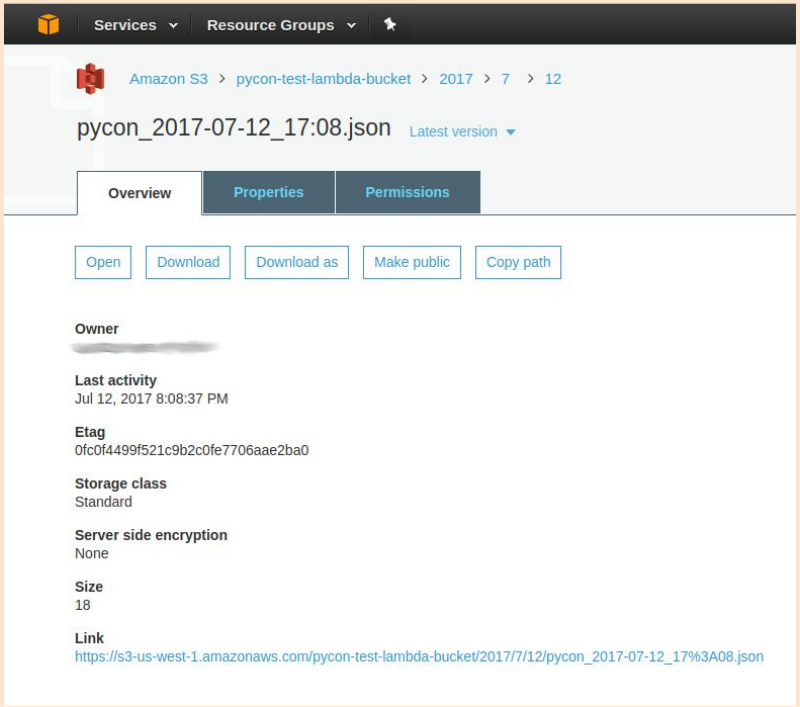
# Pipeline Example: PROFIT

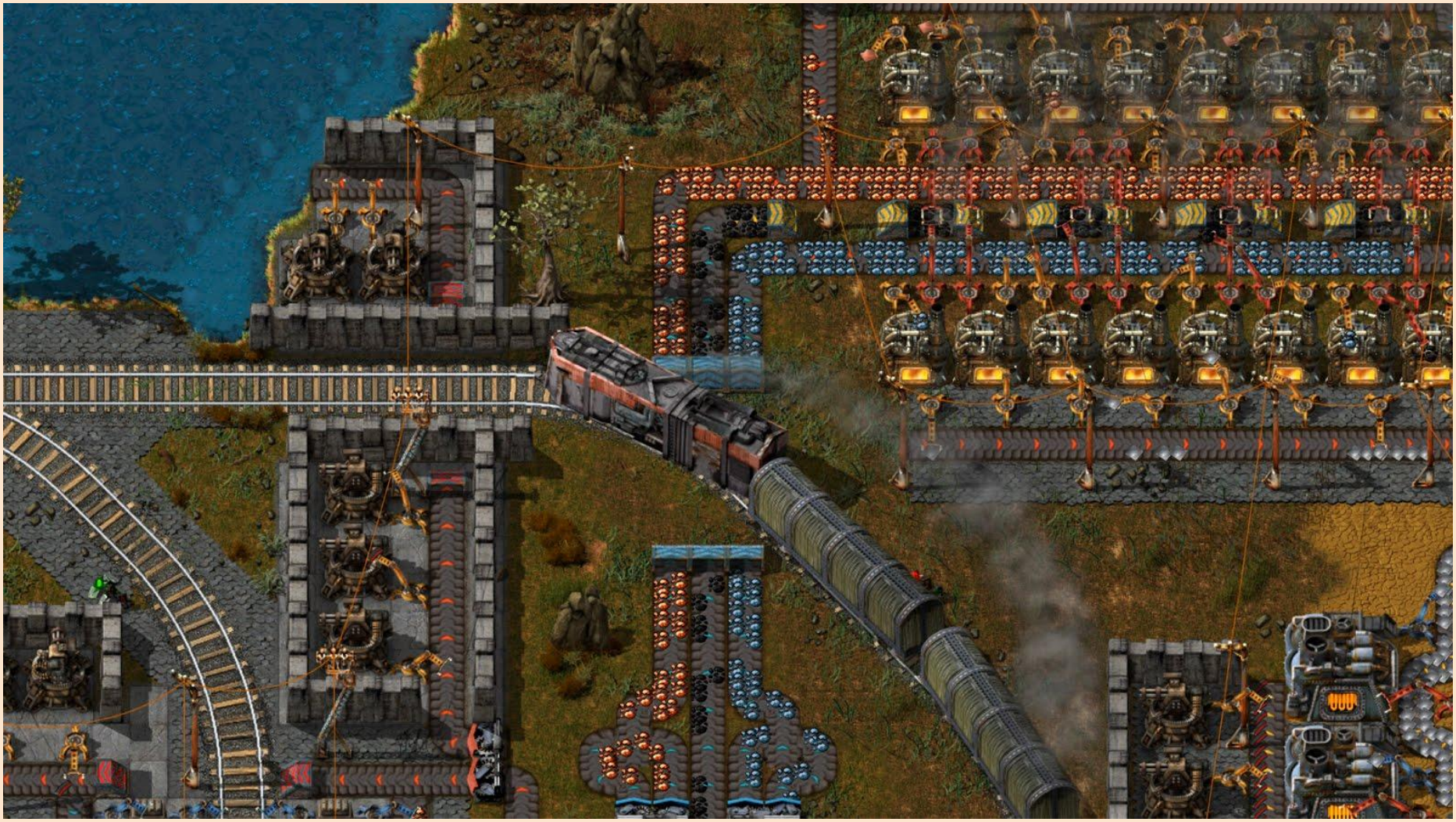
```
CloudFormation - CREATE_IN_PROGRESS - AWS::Lambda::Version - KinesisUnderscoretoUnderscores3LambdaVersionV2lWNJ6i1yDK2
BkU8VACj5dcWG7qqmxChsRSZVlo
CloudFormation - CREATE_IN_PROGRESS - AWS::Lambda::EventSourceMapping - KinesisUnderscoretoUnderscores3EventSourceMapp
ingKinesisApitestevents
CloudFormation - CREATE_COMPLETE - AWS::Lambda::Version - KinesisUnderscoretoUnderscores3LambdaVersionV2lWNJ6i1yDK2BkU
8VACj5dcWG7qqmxChsRSZVlo
CloudFormation - CREATE_COMPLETE - AWS::Lambda::EventSourceMapping - KinesisUnderscoretoUnderscores3EventSourceMapping
KinesisApitestevents
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Deployment - ApiGatewayDeployment1499879156874
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Deployment - ApiGatewayDeployment1499879156874
CloudFormation - CREATE_COMPLETE - AWS::ApiGateway::Deployment - ApiGatewayDeployment1499879156874
CloudFormation - CREATE_COMPLETE - AWS::Lambda::Permission - ApiUnderscoretoUnderscorekinesisLambdaPermissionApiGatewa
y
CloudFormation - UPDATE_COMPLETE_CLEANUP_IN_PROGRESS - AWS::CloudFormation::Stack - testKinesis2S3Workflow-dev
CloudFormation - UPDATE_COMPLETE - AWS::CloudFormation::Stack - testKinesis2S3Workflow-dev
Serverless: Stack update finished...
Service Information
service: testKinesis2S3Workflow
stage: dev
region: us-west-1
api keys:
  None
endpoints:
  POST - https://9r07kwazu7.execute-api.us-west-1.amazonaws.com/dev/kns/submit
functions:
  api_to_kinesis: testKinesis2S3Workflow-dev-api_to_kinesis
  kinesis_to_s3: testKinesis2S3Workflow-dev-kinesis_to_s3

Stack Outputs
KinesisUnderscoretoUnderscores3LambdaFunctionQualifiedArn: arn:aws:lambda:us-west-1:140461132978:function:testKinesis2
S3Workflow-dev-kinesis_to_s3:3
ApiUnderscoretoUnderscorekinesisLambdaFunctionQualifiedArn: arn:aws:lambda:us-west-1:140461132978:function:testKinesis
2S3Workflow-dev-api_to_kinesis:15
ServiceEndpoint: https://9r07kwazu7.execute-api.us-west-1.amazonaws.com/dev
ServerlessDeploymentBucketName: testkinesis2s3workflow-d-serverlessdeploymentbuck-97n6wljmsygf
```

# Pipeline Example: PROFIT

*~$ curl -d'{"foo": "bar"}' -H "Content-Type: application/json" https://9r07kwazu7.execute-api.us-west-1.amazonaws.com/dev/kns/submit*

*submit*





# It's similar with microservice frameworks

Zappa:

```
{
  "dev": {
    "app_function": "app.app",
    "aws_region": "us-west-1",
    "profile_name": "default",
    "s3_bucket": "zappa-20d98oewi"
  }
}
```

Chalice:

Basically just Flask

And your cloud-based Flask/Django/WSGI app runs as fast as "zappa deploy"

# PyWren

```
import pywren

def myfunc(args):
    # Do something!
    return result

pwex = pywren.default_executor()
futures = pwex.map(myfunc, args)
results = pwex.get_all_results(futures)
```

http://pywren.io/

# Some gotchas

- Mind your library-dependent requirements! (install serverless-python-requirements for Serverless)

Manually:

https://stackoverflow.com/questions/34749806/using-moviepy-scipy-and-numpy-in-amazon-lambda

Pre-built:

https://github.com/Miserlou/lambda-packages

- Nothing in Lambda console? Try CloudFormation!

# Some limits of AWS Lambda

- <= 512 Mb HD
- Request size <= 6Mb (if Event - 128K)
- <= 1000 concurrent executions per region
- <= 50 Mb compressed deployment package size
- <= 250 Mb uncompressed
- <= 75 Gb total packages uploaded per region
- <= 5 minutes run per request

https://docs.aws.amazon.com/lambda/latest/dg/limits.html

# AWS Lambda pricing

- First 1 million requests per month are free
- $0.20 per 1 million requests thereafter ($0.0000002 per request)
- The Lambda free tier includes 1M free requests per month and 400,000 GB-seconds of compute time per month.
- API Gateway: $3.50 per million API calls received, plus the cost of data transfer out, in gigabytes.

https://aws.amazon.com/lambda/pricing/

https://aws.amazon.com/api-gateway/pricing/

# How to test your serverless applications

Mock Boto:

https://github.com/spulec/moto

Run lambdas:

https://github.com/lambci/docker-lambda


https://twitter.com/enchantner

https://fb.me/enchantner