Building a serverless company on AWS

Padraig O'Brien @Podgeypoos79
Luciano Mammino @loige

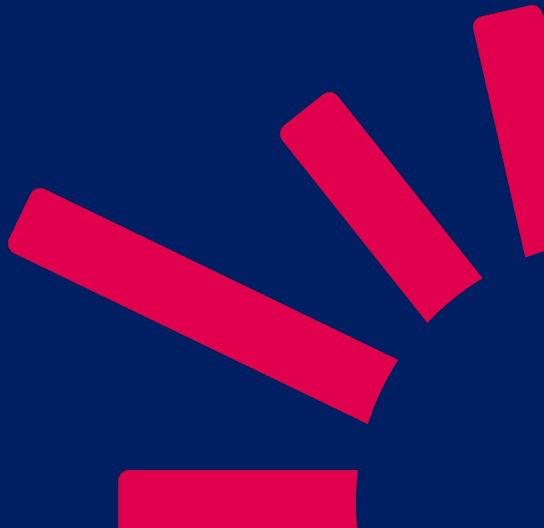
CODE EUROPE

Wroclaw, 13 Dec 2017

PLANET9

What we will cover

- Planet 9 Energy and the problem we are solving
- What is serverless?
- Our technology stack
- How our code is organised
- Path to production
- Gotchas and things we learned
- The Future

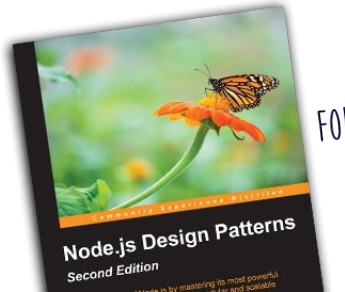


Who are we?

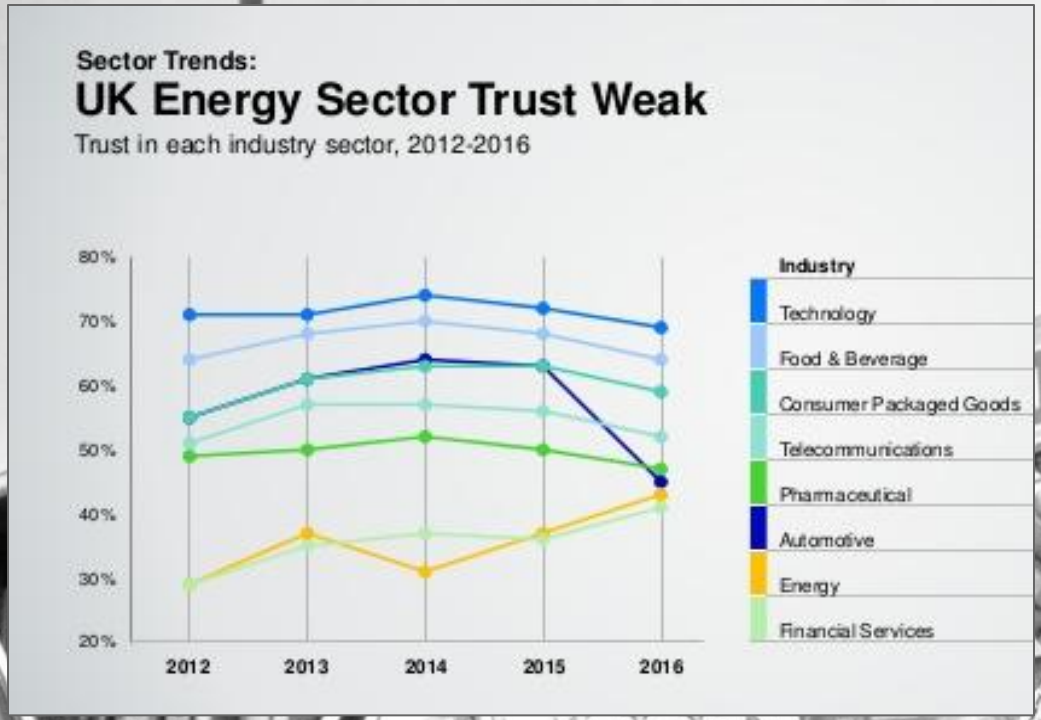
PLANET9

```
{
  “name”: “Padraig”,
  “job”: “engineer”,
  “twitter”: “@Podgeypoos79”,
  “extra”: [
    “NodeSchool organiser”,
    “LeanCoffee organiser”,
    “Serverlesslab.com founder”
  ]
}
```

{
“name”: “**Luciano**”,
“job”: “engineer”,
“twitter”: “@loige”,
“Website”: “loige.co”
“side-projects”: [
“Node.js Design Patterns”,
“Fullstack Bulletin”,
“Serverlesslab.com founder”
]
}



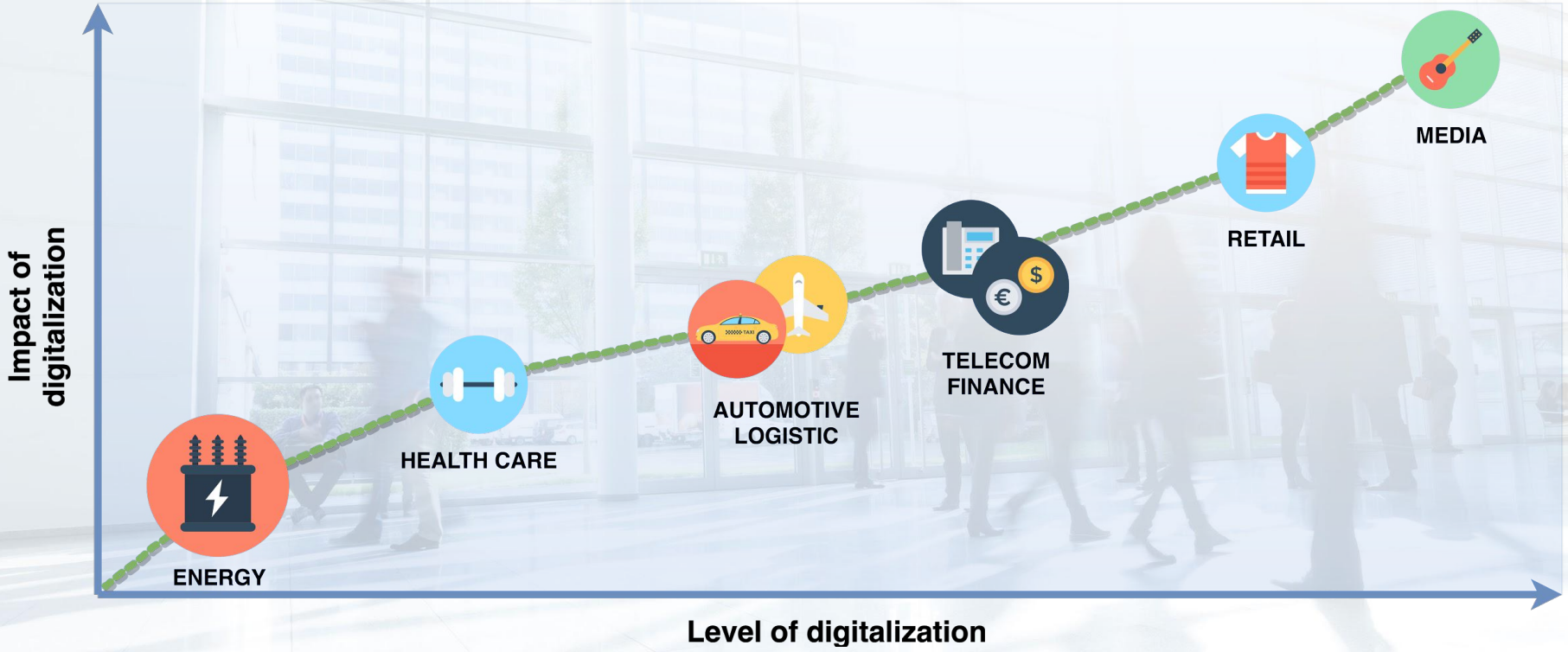
FOLLOW ME ON TWITTER
FOR A DISCOUNT!



Electricity suppliers:

Do you trust them?

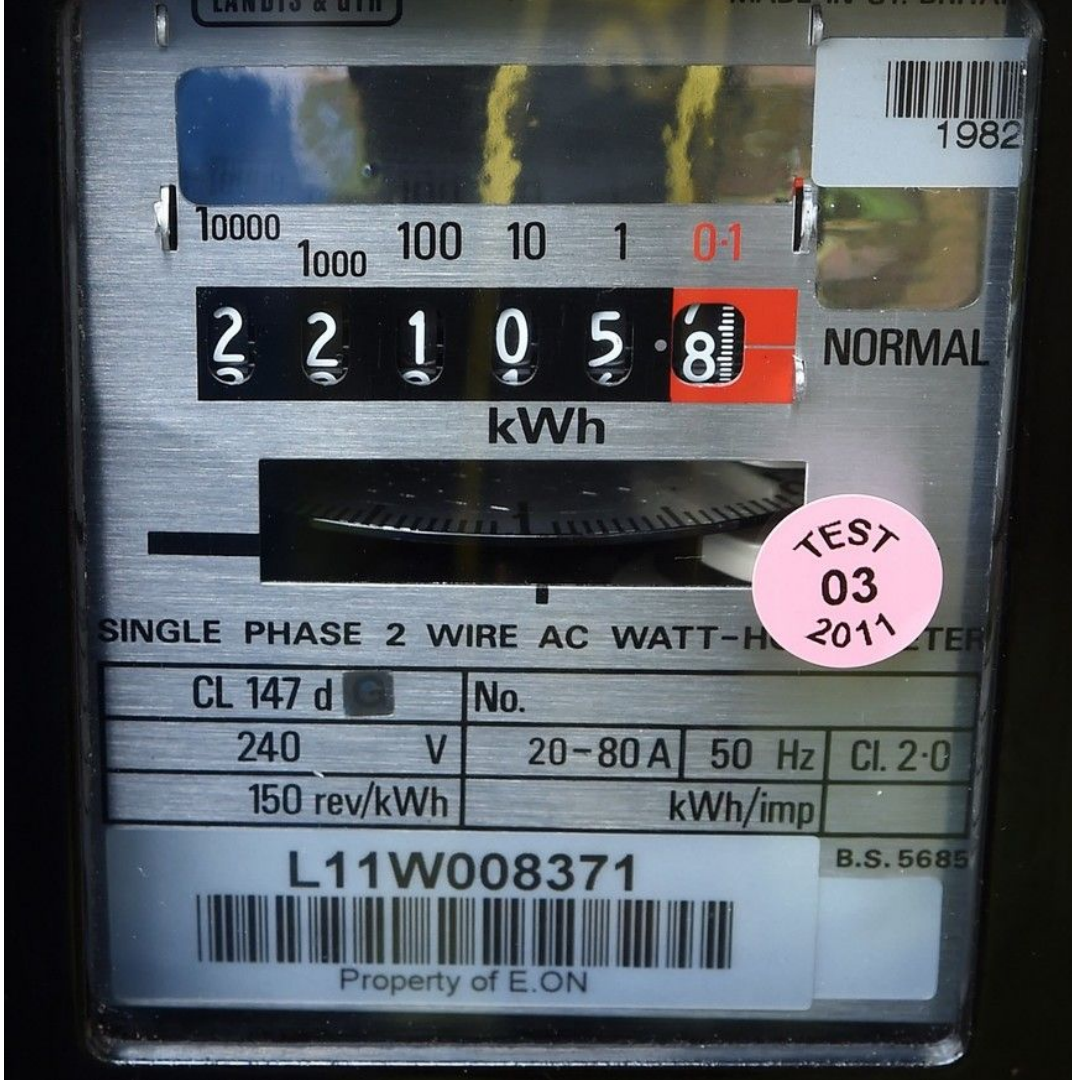
Technology adoption by industry



Source: BCG, Boston Consulting Group, 2016

The numbers

- 17520 half hours in a year.
- 30 line items per half hour.
- 6 revisions of that data.
- **~ 3 million data points (year × meter point)**



See your bill down to the half hour

Automated Energy Trading

Planet9Energy

- ESB funded startup (25 people)
- UK energy supplier.
- Focus on I & C customers.

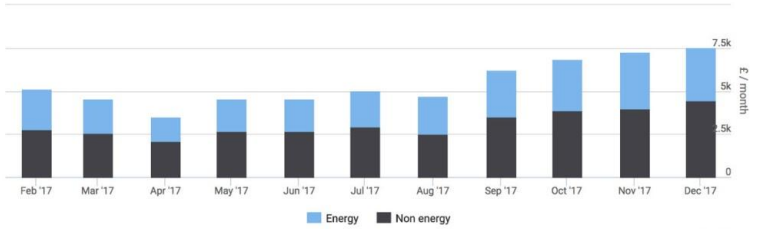
Start date for cost calculator FROM: 2017-02-01 TO: 2018-02-01

Forecast Unit Cost - Annual
p/kWh 8.59

Forecast Spend - Annual
£ 59,834

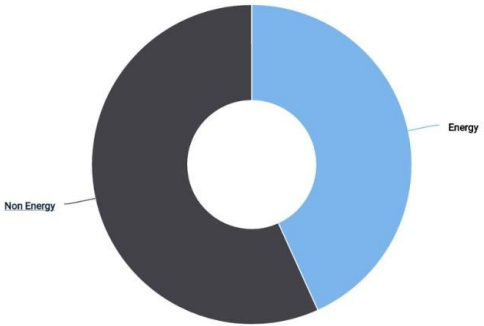
Forecast Consumption - Annual
kWh 696,667

Forecast Cost - Annual



Bill Detail

Group results by year / month
01.Feb.2017 / 01.Dec.2017
£59834.43



Bill Detail

Group results by year / month
01.Feb.2017 / 01.Dec.2017
£59834.43

Bill element	Amount	% of Total
Energy	£25862.45	43.22%
Network	£16342.40	27.31%
Renewables	£16942.06	28.31%
3rd Party	£558.19	0.93%
Imbalance	£129.33	0.22%

Cost
£

Consumption
kWh

Unit price
p/kWh

LOGOUT

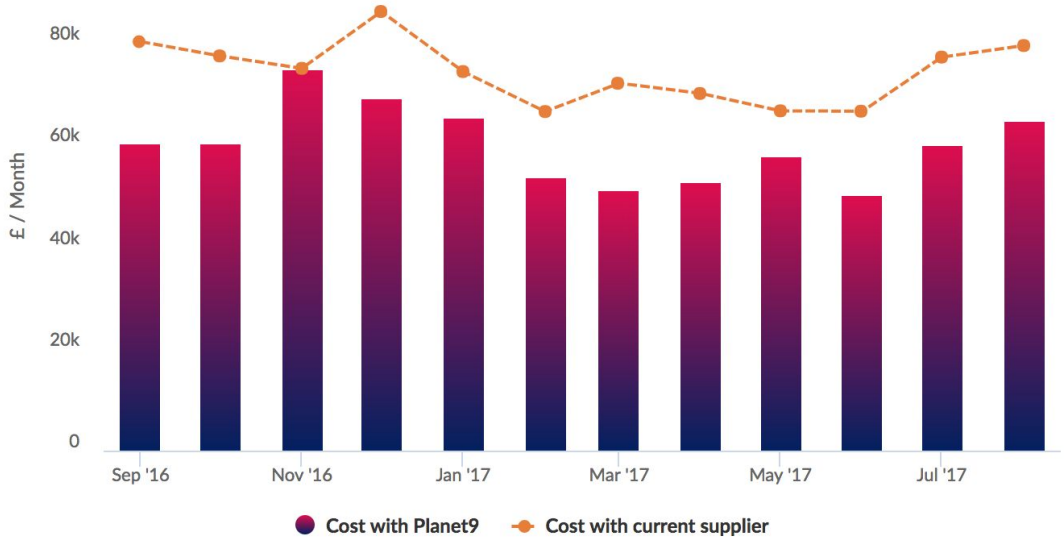
Current Supplier Rates

Day rate	Day hours begin	Night rate	Night hours begin	Max import capacity
12 p/kWh	06:00	8 p/kWh	00:00	2200 kVA

EDIT

What you would have paid with Planet9

Sep 16 - Aug 17



Cost

Sep 16 - Aug 17

Current supplier:	£892,516
Planet9:	**£721,678**

COST SAVING WITH PLANET9

Cost saving:	▼ £170,839
Percentage cost saving:	▼ 19.00%

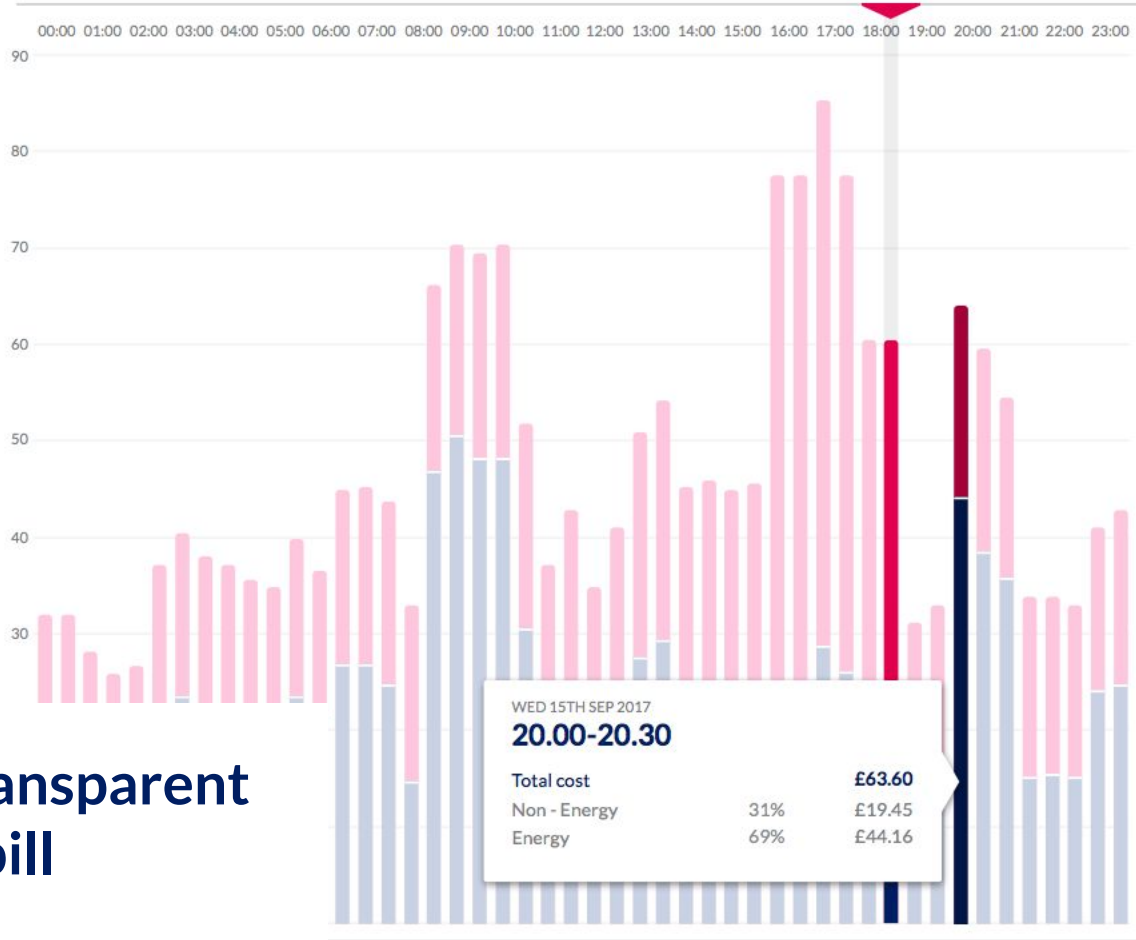
SITE
Green Phoenix
Resemer Road

MONTH VIEW

DAY VIEW

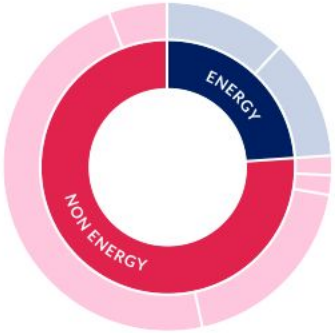
1/2 HOUR VIEW

MONDAY 25TH SEP 2017



MON 25TH, SEP 2017

18.30 - 19.00



TOTAL		**£59.93**
NON ENERGY	76%	£45.60
ENERGY	24%	£14.34

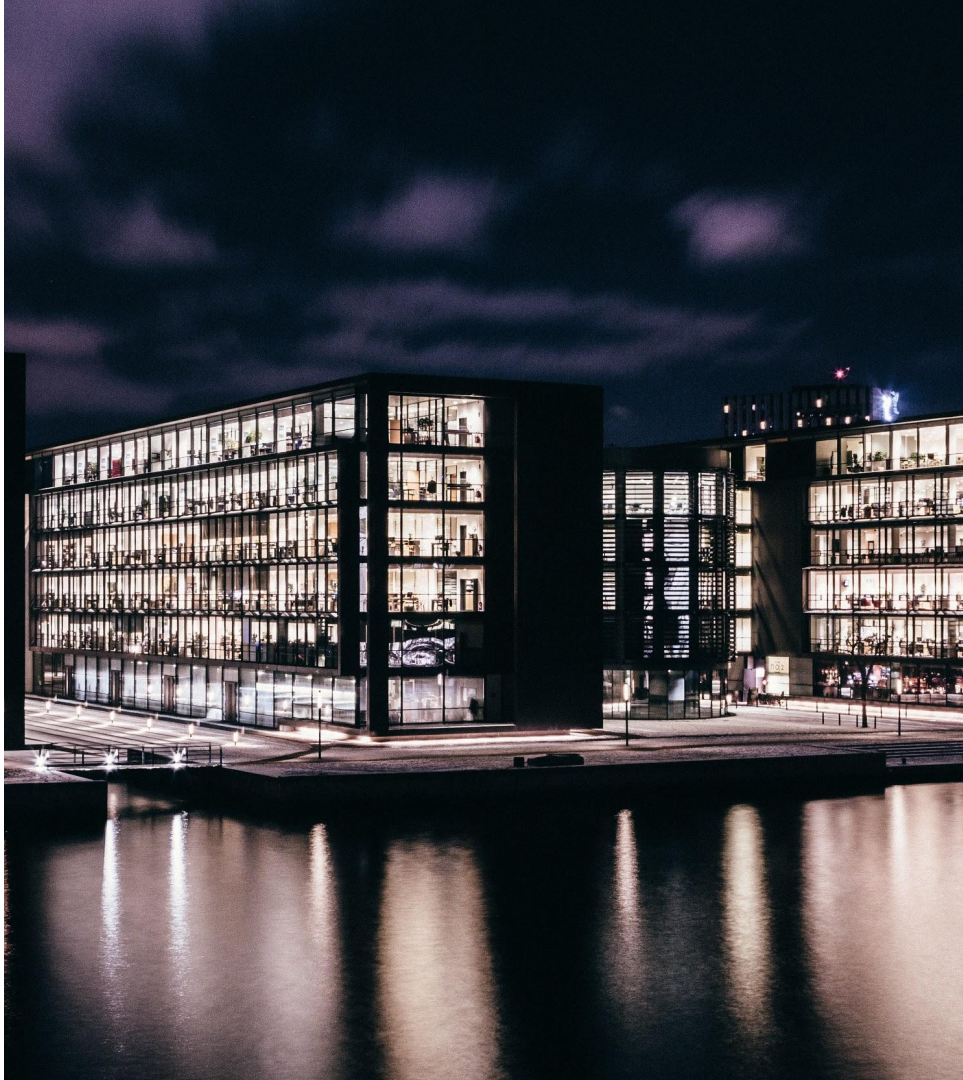
Fully transparent digital bill

What is serverless?

PLANET9

AS engineers

- We are lazy
- We want as little manual operational work
- We are full stack engineers with T/E profiles




amazon
web services

What is a Lambda?

- Function as a service (FAAS) in AWS
- Pay for invocation / processing time
- Virtually “infinite” auto-scaling
- Focus on business logic, not on servers

Lambdas as micro-services

- **Events** are first-class citizens
- Every lambda scales independently
- Agility (develop features quick and in an isolated fashion)

Classic micro-services concerns

- **Granularity** (how to separate features? BDD? Bounded Contexts?)
- **Orchestration** (dependencies between lambdas, service discovery…)

Anatomy of a Lambda in Node.js

```
1    exports.myLambdaFunction = (
2      event,
3      context,
4      callback
5    ) => {
6      // Get input from event and context
7      // Use callback() to return
8    }
```

Some use cases

- **REST** over HTTP (API Gateway)
- **SNS messages**, react to a generic message
- **Schedule/Cron**
- **DynamoDB**, react to data changes
- **S3**, react to files changes
- **IoT**

```
{
  "body": "{\"test\":\"body\"}",
  "resource": "/{proxy+}",
  "requestContext": {
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "accountId": "123456789012",
    "identity": {…
    },
    "stage": "prod"
  },
  "queryStringParameters": {
    "foo": "bar"
  },
  "headers": {…
  },
  "pathParameters": {
    "proxy": "path/to/resource"
  },
  "httpMethod": "POST",
  "stageVariables": {
    "baz": "qux"
  },
  "path": "/path/to/resource"
}
```


```
POST /path/to/resource?foo=bar

{
  “test”: “body”
}
```

HTTP REQUEST - API Call

Enter the Serverless Framework

PLANET9

Anatomy of Serverless.yml

```yaml
frameworkVersion: ">=1.2.0"

service: d36-meterdata-service

provider:
  name: aws
  runtime: nodejs6.10
  stage: ${env:STAGE}
  region: eu-west-1
  memorySize: 1536
  timeout: 300
  cfLogs: true


custom:
  version: 2
  accountIds:
    dev: 123456
  customDomain:
    domainName: ${env:API_CUSTOM_DOMAIN}
    basePath: 'meterdata'
    createRoute53Record: false
```

Serverless.yml (1/2)
Environment configuration

Anatomy of Serverless.yml

```yaml
functions:
  getMeterReadings:
    handler: build/handler.getMeterReadings
    timeout: 30
    events:
      - http:
          path: readings
          method: GET
          private: true
```

Serverless.yml (2/2)
Defining functions and events

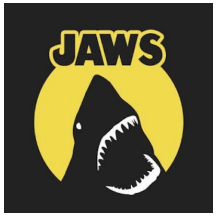
Tech stack

PLANET9

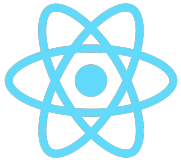
Initial stack











Iteration 1 Review

- **Dynamodb** - low ops overhead but only good for simple read patterns and no good backup solution.
- **Redshift** - epic at aggregation but limited to 50 or so connections.
- **JAWS - 1.x** was completely different so we had to re-write (almost) everything

Iteration 2

Iteration 2 Review

- **Cassandra** replaced redshift.
- **Postgres RDS** is a lot more flexible than dynamoDB
- **Ansible** is very good for provisioning VMs.
- **Rundeck** was used for runbook automation for deploying Lambdas.

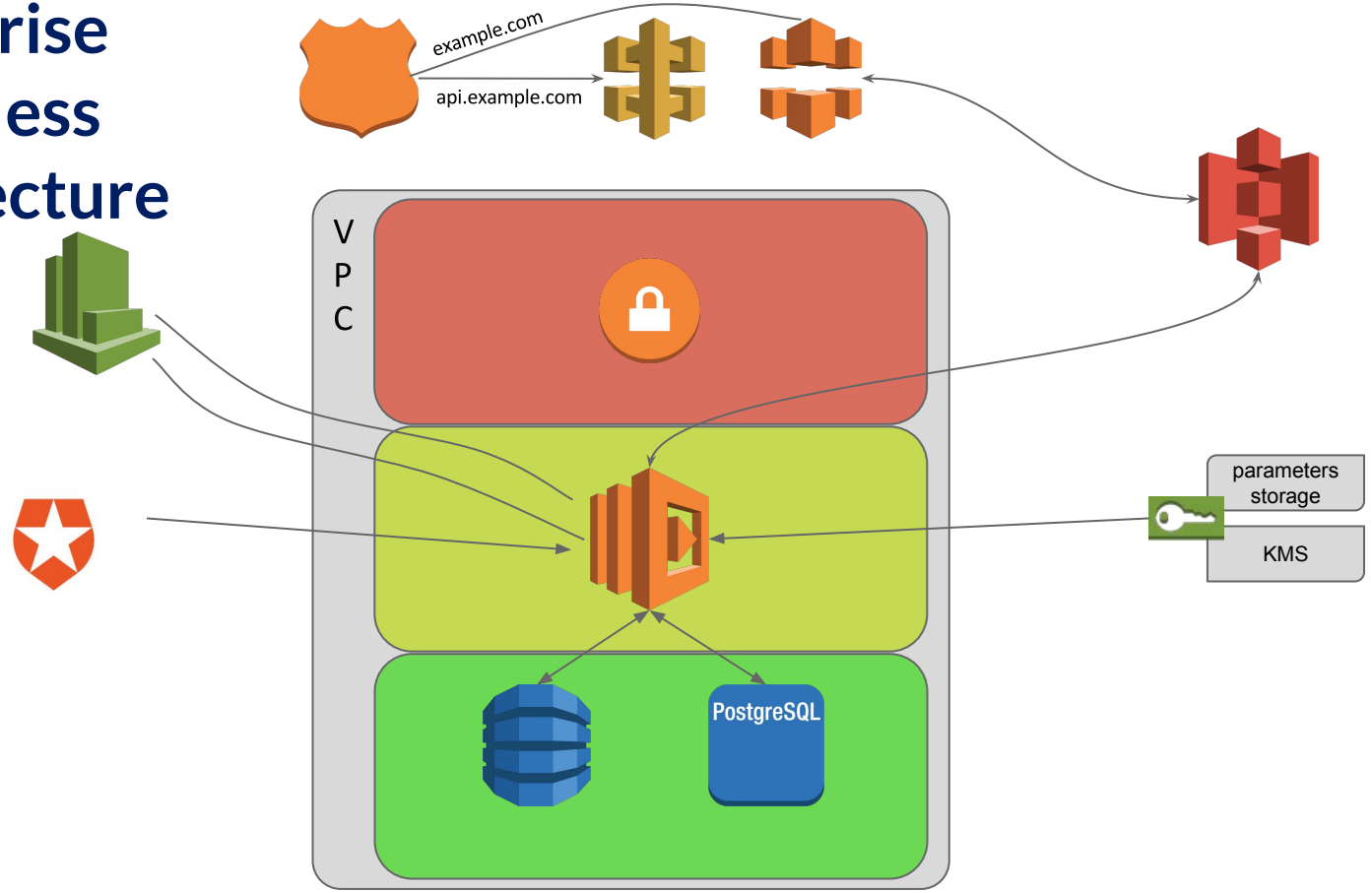
Current iteration

Parameter Store

Current iteration

- Defined **custom VPC**, Yay we are (more) secure.
- Dropped **Cassandra**.
- Dropped **Rundeck**, replaced it with parameter store and **Jenkins**.
- Started using **Terraform**.

Typical enterprise serverless architecture



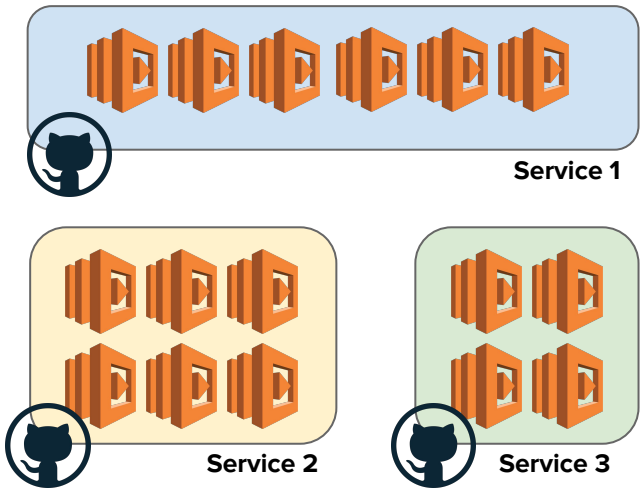
How our services are organised

PLANET9

Iteration 1

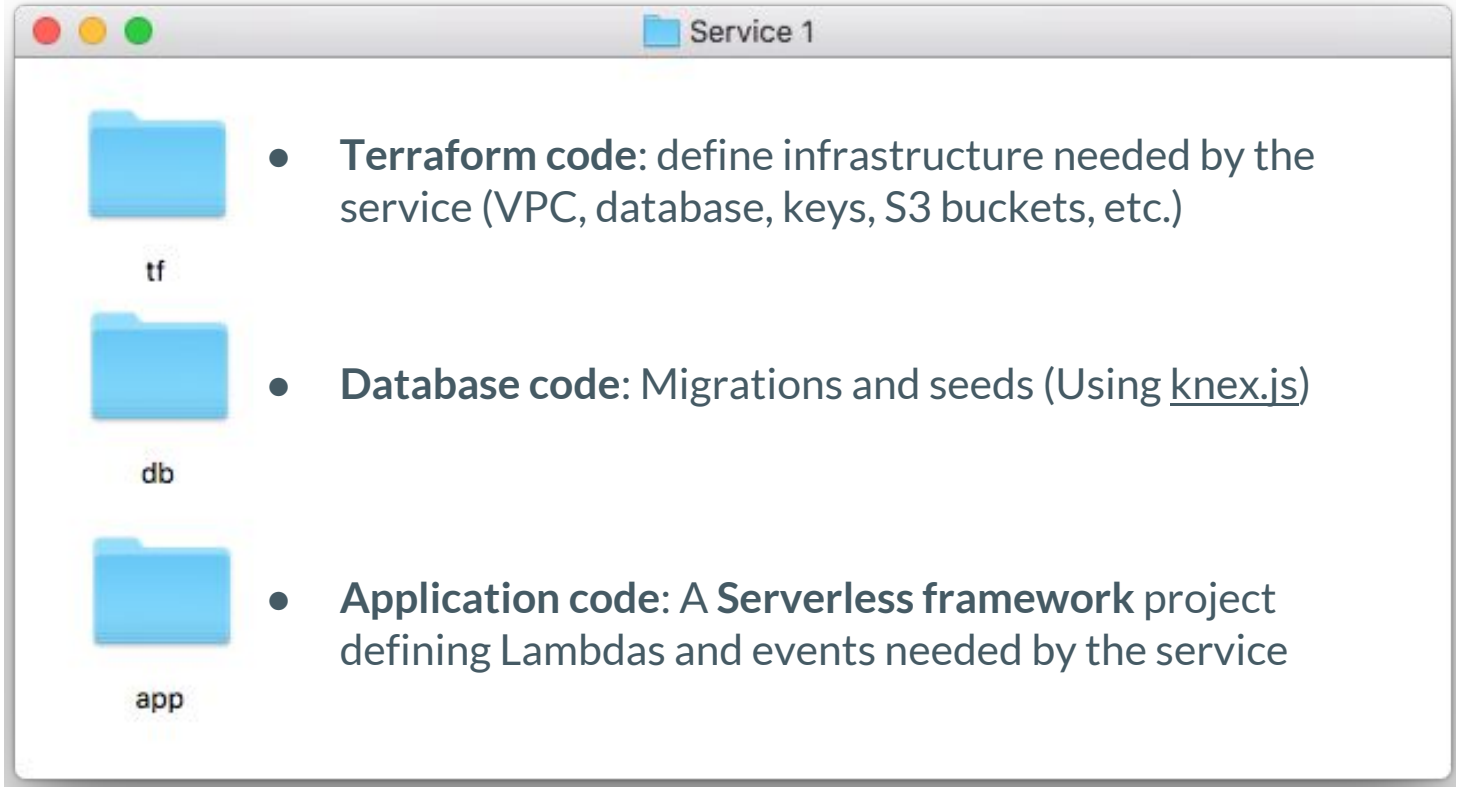
- A **function** (not a service) is the natural level of granularity!
- How to identify and structure services?
- How to connect services?
- How many repositories?
- How to deploy?
- Versioning?
- When and how to share code?

Iteration 2



- Proper service design using methodologies like **Domain Driven Design**
- Find the **bounded context** of each service
- Integration through message passing (events / APIs)
- Put everything related to a service into one repo

Current code layout



The path to production

PLANET9

Develop locally

- Develop locally on our laptops.
- **PostgreSQL** on docker.
- Plugins from Serverless to “mimic” API Gateway etc.
- Git commit all the things to branch.
- Pull request.
- Integrate to master.
- **Jenkins** takes care of everything else (more or less).

We we integrate to master

Our CI (Jenkins):

- Run tests
- Build the project
- Updates the infrastructure (**Terraform**)
- Updates the database (**Knex**)
- Deploy lambdas (**Serverless** framework)
- We have a **stop-gate** with manual approval before it goes to **production**

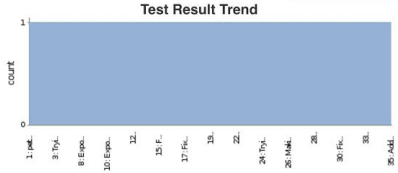
Back to Dashboard
Status
Changes
Build Now
Delete Pipeline
Configure
Open Blue Ocean
Full Stage View
SonarQube
Embeddable Build Status
Pipeline Syntax
Git Polling Log

Pipeline duos-invoice-service

add description
Disable Project

Recent Changes

Stage View



(just show failures) enlarge

	Checkout	Install	Unit Test	Acceptance	Deploy QA	Smoke Test	Integration	Performance	Publish	Deploy Staging / Pre - Prod	Smoke Test	Deploy Prod	Smoke Test
Average stage times: (Average full run time: ~17min 39s)	8s	1min 1s	10s	636ms	59s	623ms	624ms	637ms	5s	1min 0s	5s	1min 3s	6s
36 : Fixed yaml error — Dec 07 13:15 — 1 commits	8s	1min 0s	10s	638ms	59s	616ms	628ms	621ms	8s	57s	620ms	59s (paused for 23min 29s)	617ms
35 : Added object tagging permiss — Dec 07 12:10 — 1 commits	9s	1min 10s	11s	628ms	25s failed								
34 : Adding tagging permissions — Dec 07 11:45 — 1 commits	8s	59s	10s	625ms	1min 11s	619ms	632ms	617ms	4s	1min 5s	623ms	1min 4s (paused for 4min 36s)	622ms
33 : Changed policy — Dec 07 11:21 — No Changes	8s	59s	10s	641ms	1min 5s	621ms	623ms	633ms	5s	1min 4s	620ms	1min 3s (paused for 9s)	627ms
#32 — Dec 07 11:18 — 1 commits													
31 : Updating buckets access poli — Dec 07 10:46 — 1 commits	8s	59s	10s	632ms	1min 8s	617ms	616ms	624ms	5s	1min 4s	631ms	1min 3s (paused for 25min 7s)	615ms
30 : Fixed response format — Dec 04 16:17 — 1 commits	9s	1min 3s	10s	681ms	55s	627ms	626ms	752ms	4s	55s	639ms	1min 0s (paused for 9s)	622ms

Build History

trend

find

- 36 : Fixed yaml error — 07-Dec-2017 13:15
- 35 : Added object tagging permiss — 07-Dec-2017 12:10
- 34 : Adding tagging permissions — 07-Dec-2017 11:45
- 33 : Changed policy — 07-Dec-2017 11:21
- #32 — 07-Dec-2017 11:18
- 31 : Updating buckets access poli — 07-Dec-2017 10:46
- 30 : Fixed response format — 04-Dec-2017 16:17
- 29 : Using server side encryption — 04-Dec-2017 16:04
- 28 : Recreating the bucket for th — 04-Dec-2017 15:45
- 27 : Permission fixes — 04-Dec-2017 15:00
- 26 : Making sure to deploy from t — 04-Dec-2017 14:44
- 25 : Updated handler definition — 04-Dec-2017 14:30
- 24 : Trying to create the buckets — 04-Dec-2017 12:18
- 23 : Trying to create the buckets — 04-Dec-2017 11:40

Things we learned

PLANET9

Lots of code is repeated in every lambda

```
(event, context, callback) => {
  // decrypt environment variables with KMS
  // deserialize the content of the event
  // validate input, authentication, authorization
  // REAL BUSINESS LOGIC (process input, generate output)
  // validate output
  // serialize response
  // handle errors
}
```

BOILERPLATE CODE

BOILERPLATE CODE

middy.js.org

The stylish Node.js middleware engine for AWS Lambda

```
const middy = require('middy')
const { middleware1, middleware2, middleware3 } = require('middy/middlewares')

const originalHandler = (event, context, callback) => {
  /* your pure business logic */
}

const handler = middy(originalHandler)

handler
  .use(middleware1())
  .use(middleware2())
  .use(middleware3())

module.exports = { handler }
```

- **Business logic code is isolated:**

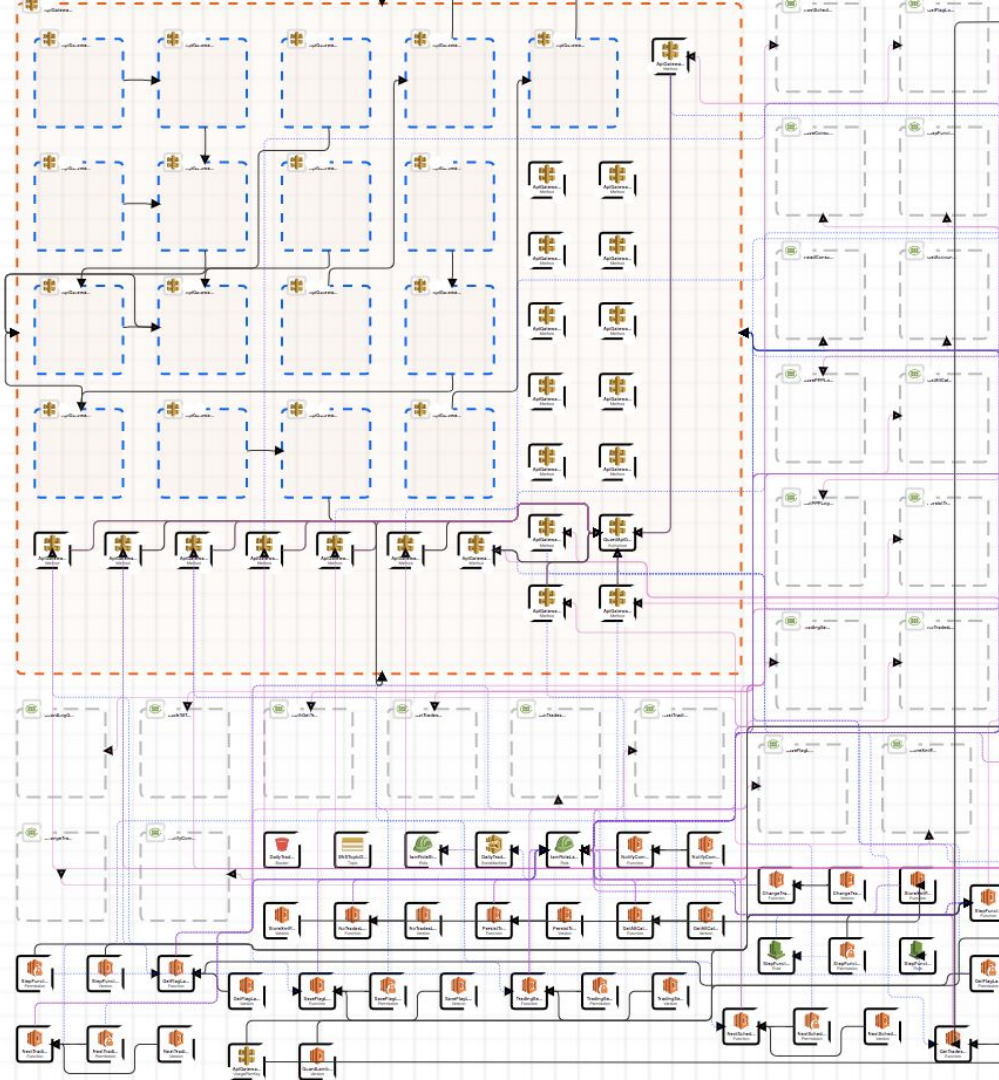
  Easier to understand and test

- **Boilerplate code is written as middlewares:**
  - Reusable
  - Testable
  - Easier to keep it up to date

Large services

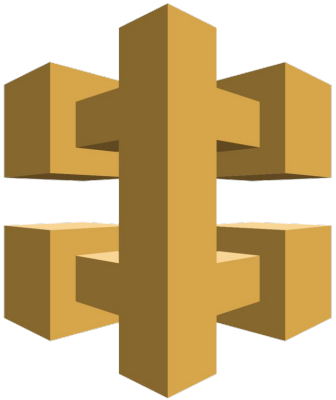
- Template format error: Number of resources, 214, is greater than the maximum allowed, 200

- serverless-plugin-split-stacks
  - migrates the RestApi resource to a nested stack



API Gateway & Lambda size limits

- 128 K payload for async event invocation
- 10 MB payload for response
- Don’t find these limits when using sls webpack serve



API Gateways events

https://myapi.me?name=me

```
{
  "requestContext": { … },
  "queryStringParameters": {
     "name": "me"
  },
  "headers": { … }
}
```

```
const handler = (event, context, callback) {
  console.log(event.queryStringParameters.name)
  // …
}
```

It will output "me"

API Gateways events

`https://myapi.me`

(no query string!)

```
{
  "requestContext": { … },

    (no queryStringParameters key!)

  "headers": { … }
}
```

```
const handler = (event, context, callback) {
  console.log(event.queryStringParameters.name)
  // …
}
```

undefined

TypeError: Cannot read property 'name' of undefined

API Gateways events

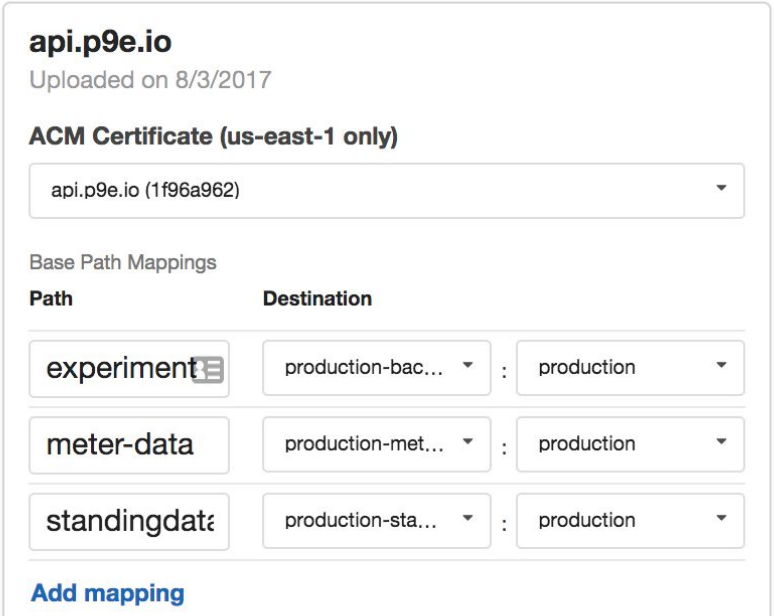
```
const handler = (event, context, callback) {
  if (event.queryStringParameters) {
    console.log(event.queryStringParameters.name)
  }
  // or
  console.log(event.queryStringParameters ?
    event.queryStringParameters.name : undefined
}
```

MOAR boilerplate!

Api Gateway proxy event normalizer middleware is coming to Middy!

API Gateways custom domain.

- Serverless does not provide custom domain name mapping
- Has to be done in cloudformation
- There is a plugin.
  [Serverless-plugin-custom-domain]
  [Serverless-domain-manager]



Disk usage matters

- 50 MB if deploying directly.
- 250 if going from S3.
- We use Node.js, Webpack and *tree shaking* help us (serverless webpack plugin)
- 75GB for entire region, covers all lambdas and versions of lambdas, you might need a *janitor lambda*...

Node.js Event loop

- We use postgres and connection pooling
- Event loop will never become empty
- Use Middy! :)


```
const middy = require('middy')
const {doNotWaitForEmptyEventLoop} = require('middy/middlewares')
const handler = middy((event, context, cb) => {
  // ...
}).use(doNotWaitForEmptyEventLoop())
```

S3 events: filename encoding


```
{
  "Records": [{
    "s3": {
       "object": {
        "key": "Podge%27s+Unicorn.png"
      }
    }
  }]
}
```

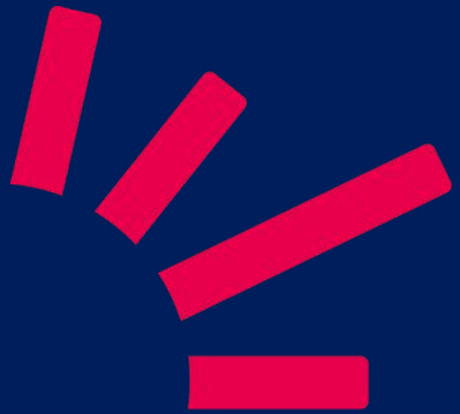
Space replaced with "+" & URL encoded

```
const middy = require('middy')
const { s3KeyNormalizer } = require('middy/middlewares')
middy((event, context, cb) => {
  console.log(event.Records[0].s3.object.key) // Podge's Unicorn
}).use(s3KeyNormalizer())
```


The Future

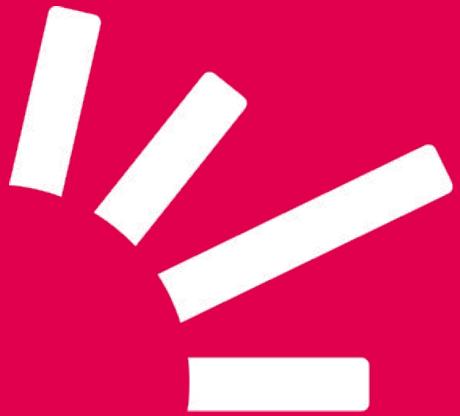
Serverless Aurora

https://aws.amazon.com/rds/aurora/serverless/



Blue Green Deploys

http://docs.aws.amazon.com/lambda/latest/dg/automating-updates-to-serverless-apps.html



A retrospective

2 years after...

- Learning how to do serverless right took a while
  (as learning any other new tech)
- We never received a call at 2AM!
- Our tech bill is extremely small
  (apart for RDS!)
- We definitely don't regret the choice :)

We are hiring!

@loige @Podgeypoos79

Thank you