Serverless from the trenches

@Podgeypoos79 - @loige

loige.link/serverless-trenches

user group network
AWSUGN

Dublin, 03/10/2017

Photo by Stijn Swinnen on Unsplash

Padraig O'Brien

PLANET9

Head of Data and analytics

Serverless lab

Connect

padraigobrien.com - @podgeypoos79 - padraigobrien - Linkedin

Luciano Mammino

Fullstack Bulletin

fullstackbulletin.com

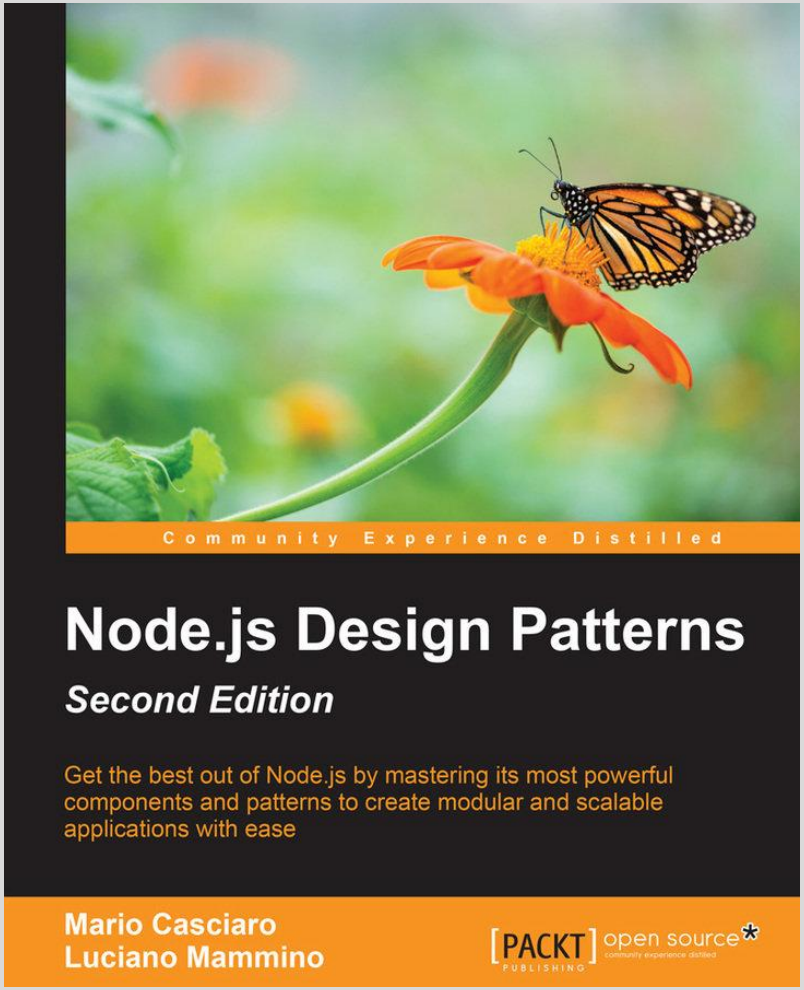
PLANET9

Principal Application Engineer

Serverless Lab

Connect

loige.co - @loige - lmammino - Linkedin



Mission

Campaign 1:

Stories of architecture

Campaign 2:

Stories of development

Campaign 3:

Stories of things we learned

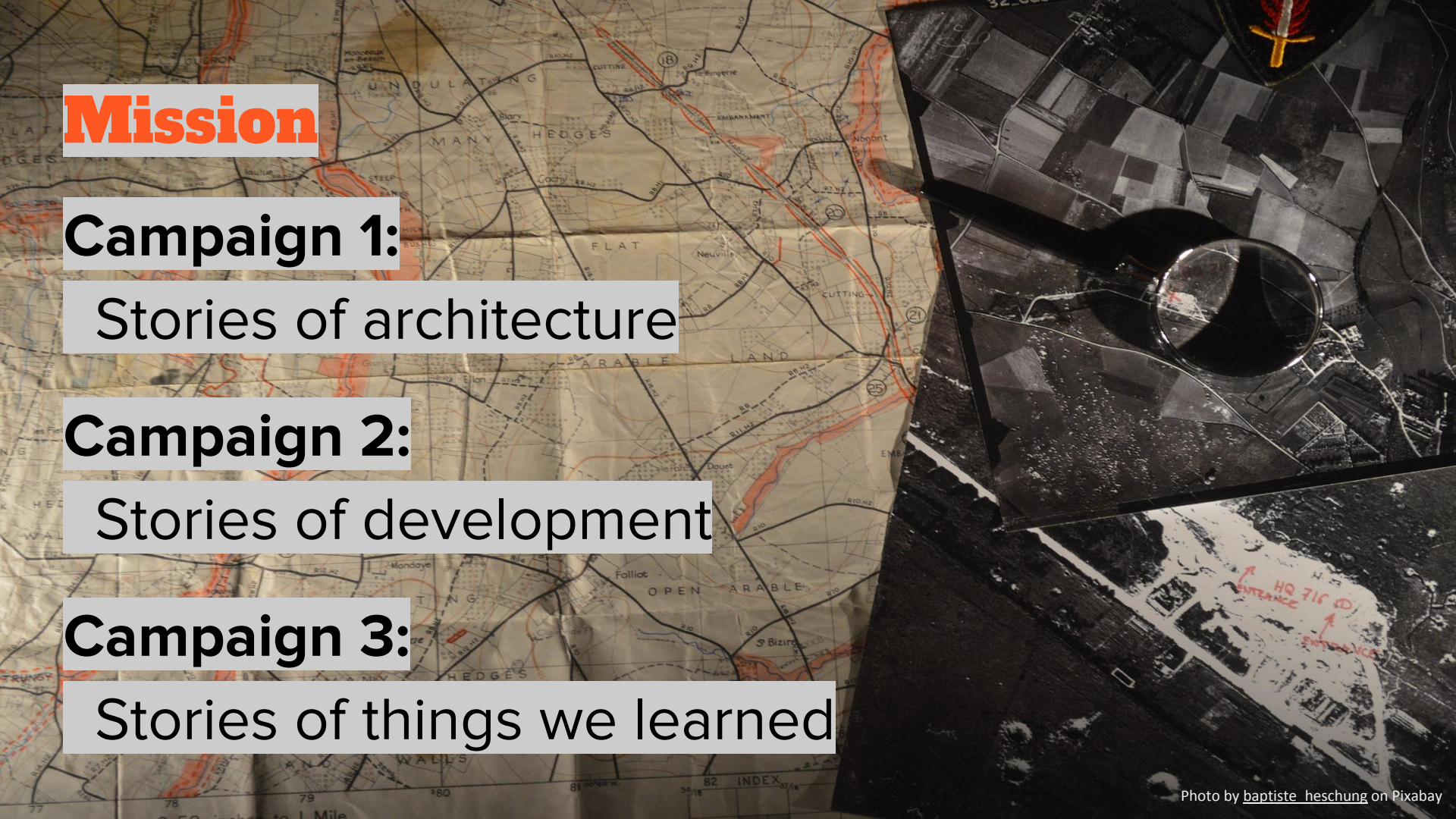


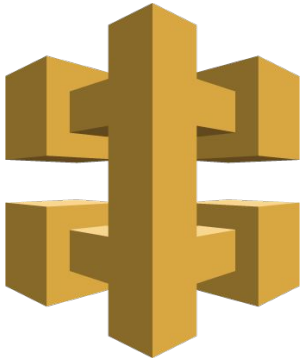
Photo by baptiste_heschung on Pixabay

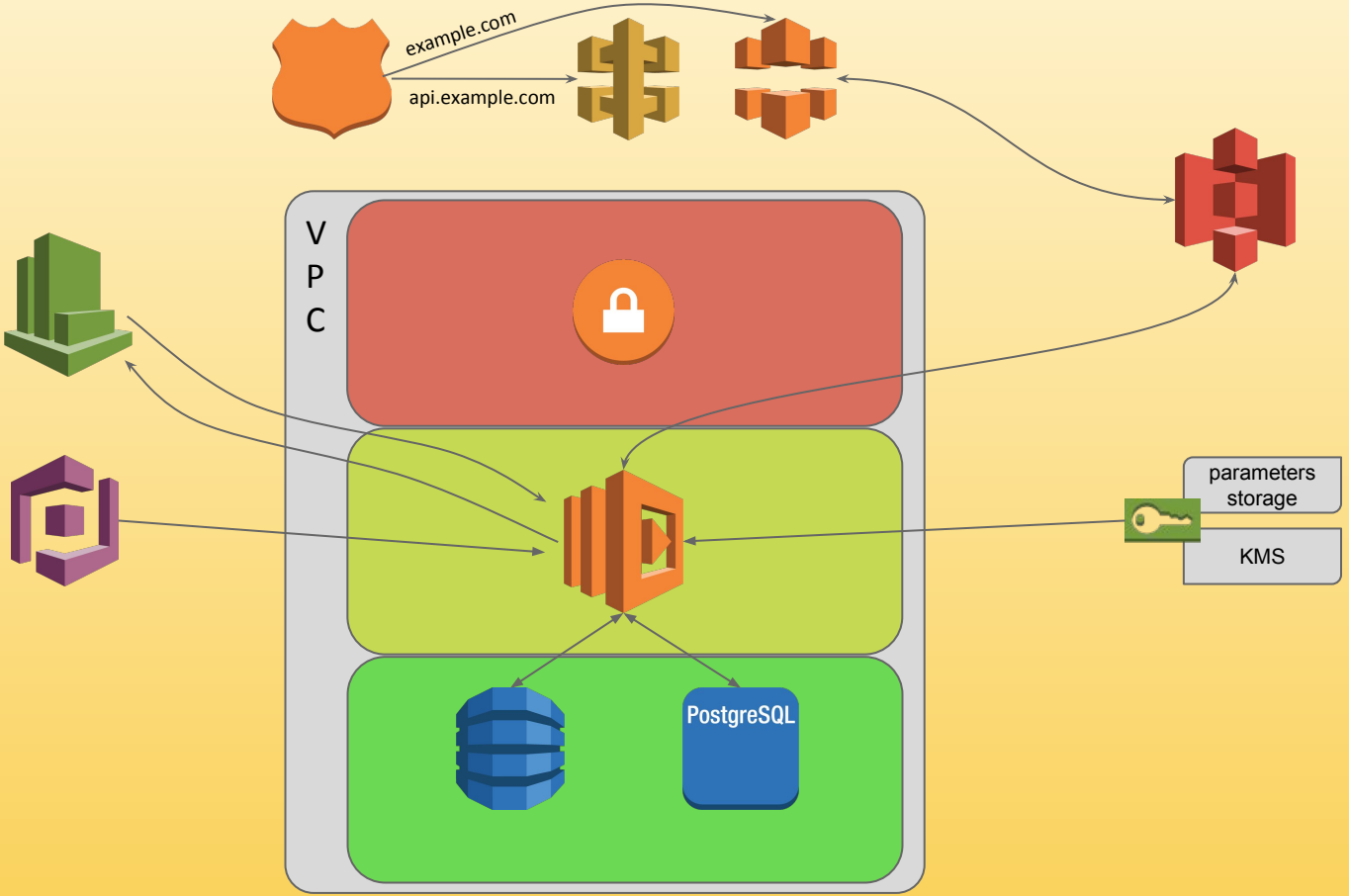
Campaign 1.

Stories of architecture

Photo by Marinas.com

There's more to it than API Gateway & Lambda

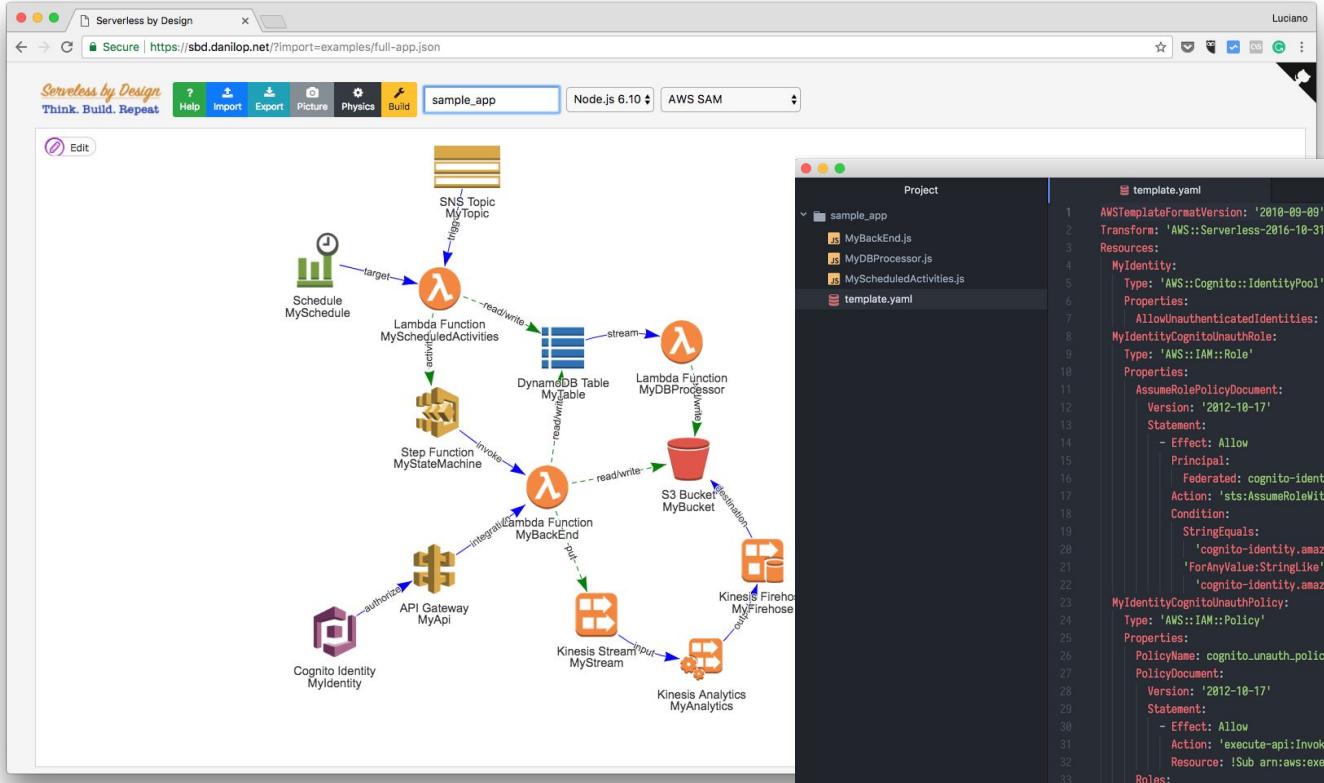


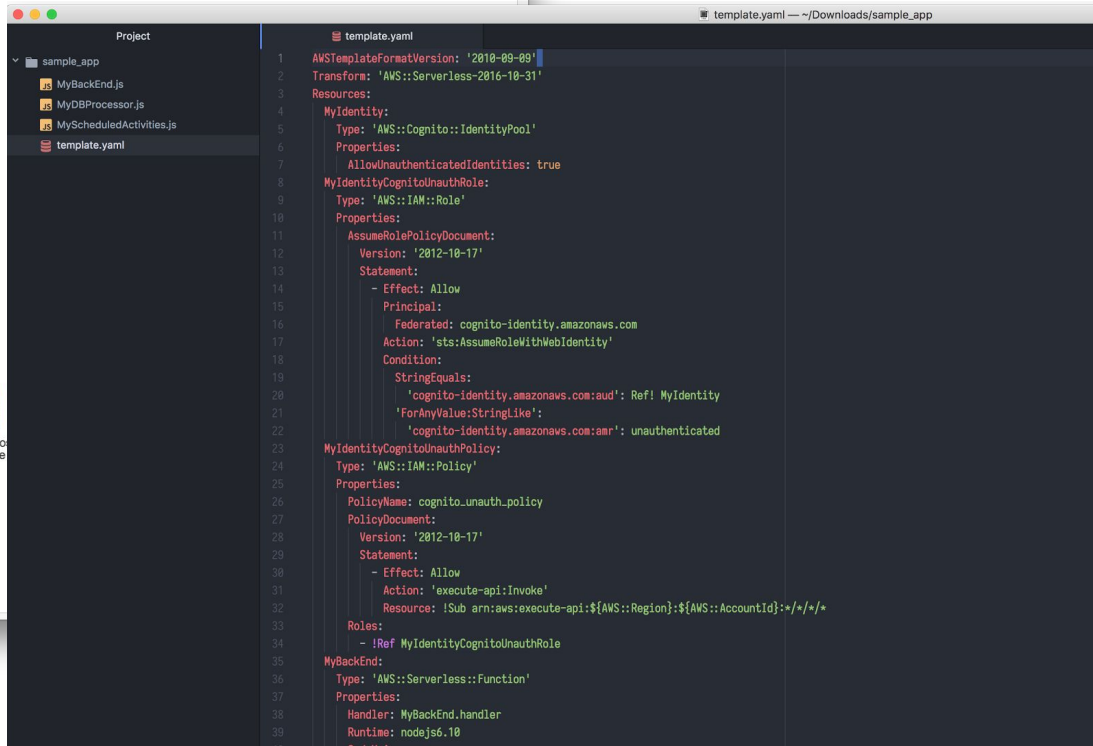

example.com
api.example.com
V
P
C
parameters
storage
KMS
PostgreSQL

No shortcuts!

- **Spend a lot of time reading and learning!**
  - Documentation & White papers
  - Security best practices
- **If you like books:**
  - Serverless architectures on AWS (P. Sbarski)
  - AWS Lambda in action (D. Poccia)
- **"Infrastructure as code" is your friend:**
  - Terraform or Cloudformation & SAM
  - Serverless framework
- **If that's not enough... You can always ask for help :)**
  - Version 1
  - Serverlesslab.com

Serverless by design

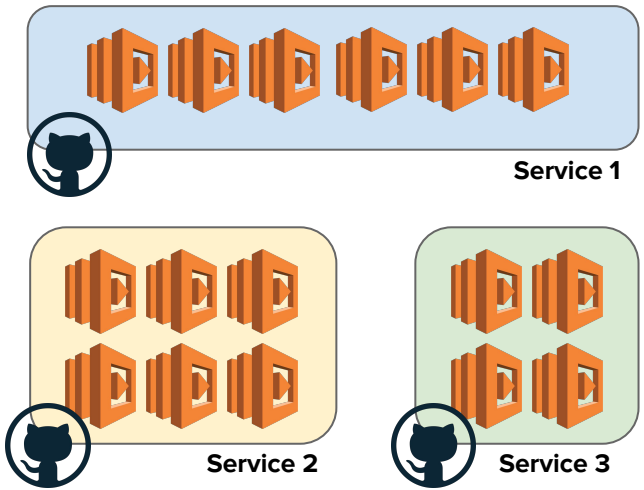




It's micro-services-ish

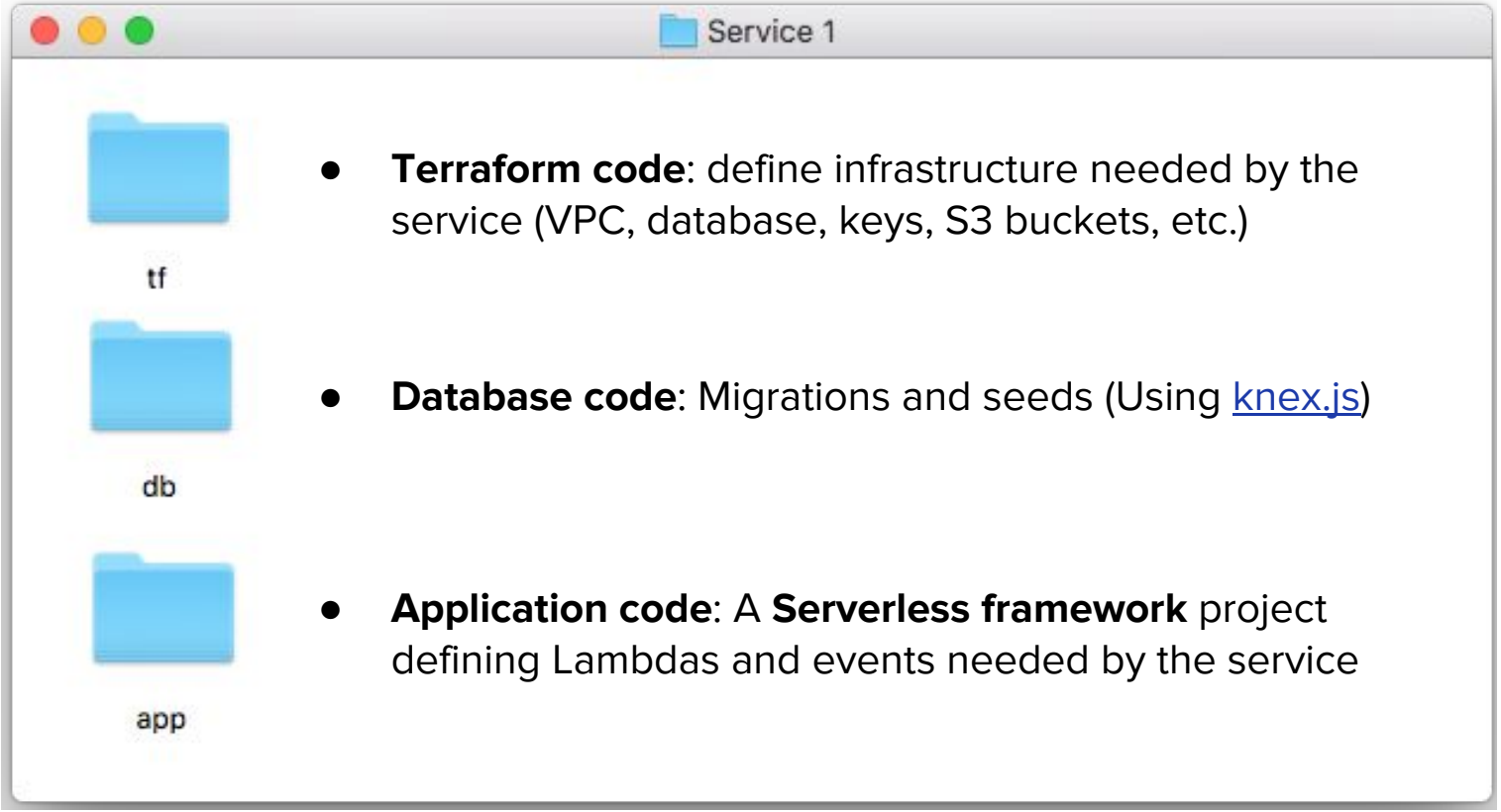
- A **function** (not a service) is the natural level of granularity!
- How to identify and structure services?
- How to connect services?
- How many repositories?
- How to deploy?
- Versioning?
- When and how to share code?

Functions are just building blocks!



- 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.

How do we organise a service



- **Terraform code**: define infrastructure needed by the service (VPC, database, keys, S3 buckets, etc.)
- **Database code**: Migrations and seeds (Using knex.js)
- **Application code**: A **Serverless framework** project defining Lambdas and events needed by the service

When we integrate to master...

Our CI (Jenkins):

- Run tests
- Build the project
- Updates the infrastructure (Terraform)
- Updates the database (Knex)
- Deploy lambdas (Serverless framework)

(... of course we have multiple environments)

The anatomy of a typical Lambda function

```
(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**:

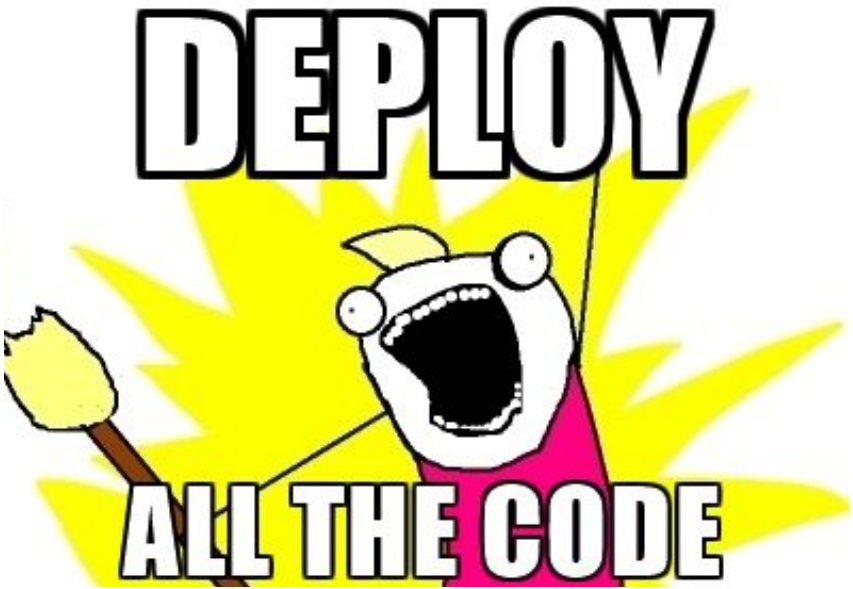
  Easier to understand and test

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

Campaign 2.

Stories of development

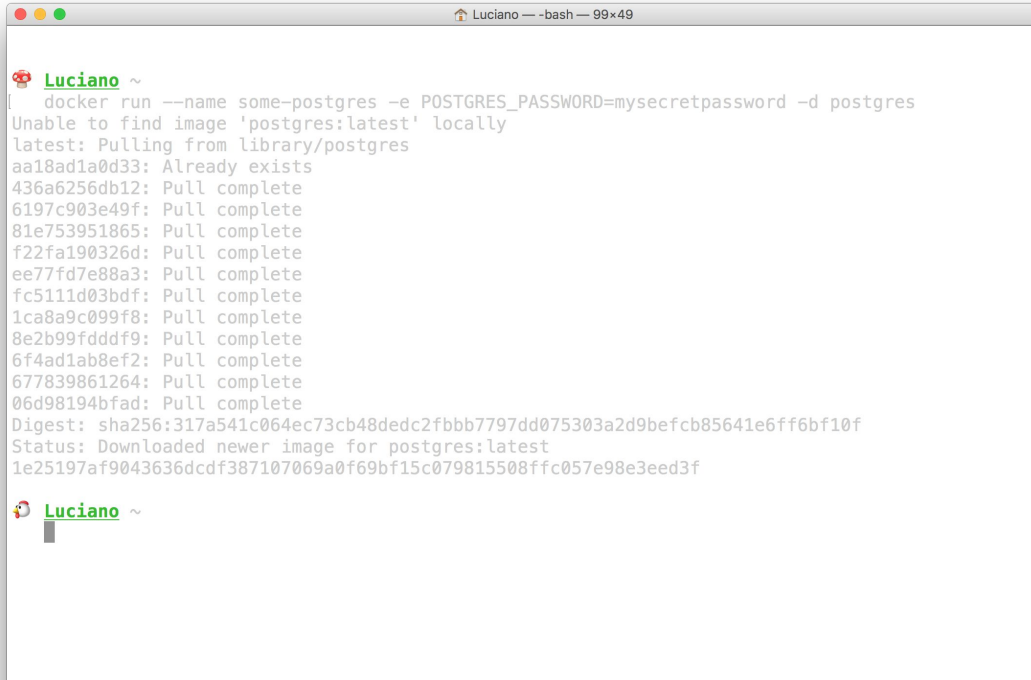
Organising development



- BDD
  - Discuss,
  - Distill
  - Acceptance criteria
- If you test, you Jest
- Commit to GitHub
  (short lived *feature branches*)
- PR, Code Review & Merge
- Jenkins does the deploy magic.

Local development

- VS Code, Sublime, VIM.
- Postgres locally using docker.
- Knex.js to create the DB and seed.
- No local api gateway from AWS.
- Serverless, serverless-webpack and serverless-offline to simulate api gateway



Serverless Application Model (SAM)

- Exploring SAM and SAM Local
- Uses similar markup to Serverless.
- SAM Local uses Docker.
- Step in the right direction for AWS in regard to local dev tooling

Campaign 3.

Stories of things we learned (the hard way...)

Large services

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

- Cloudformation has a hard limit
- Maximum of 200 resources

-

- 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

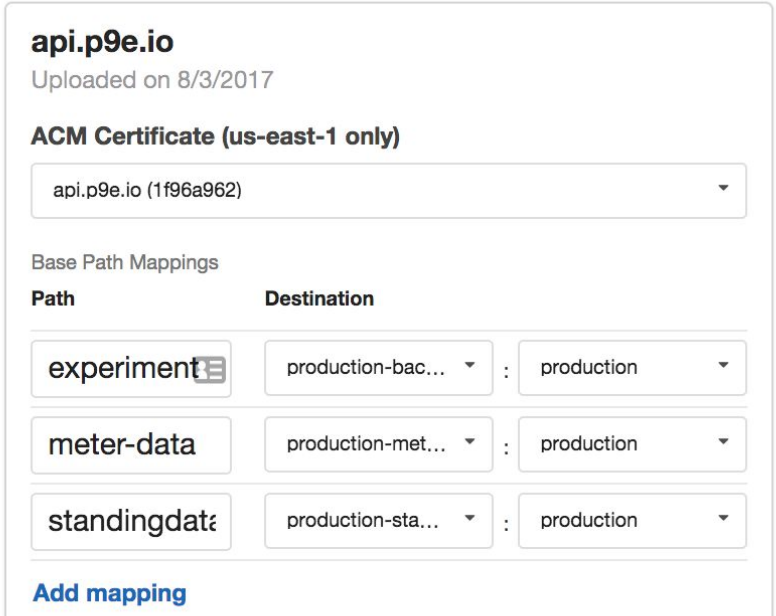
```
const handler = (event, context, callback) {
  if (event.queryStringParameters) {
    console.log(event.queryStringParameters.name)
  }
  // or
  console.log(event.queryStringParameters ?
    event.queryStringParameters.name : undefined
}
```

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]



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())
```

In Summary...

- Not a great body of work out there on best practices (yet...)
- You trip over the simplest of things in AWS
- Local tooling still lacking
- We still believe **this to be the future** and it is definitely improving!

Thank you!

@Podgeypoos79

@loige