In search of a new paradigm

Who am I ?

Gildas Le Nadan

gildas@endemic-systems.com

@endemics

What do I do ?

I work in IT

I'm an Independant Consultant
I'm a System Engineer
I do Technical Architecture
and project management
and also I do CI/CD
ah, and Automation
and sometime other stuff too

You do what ?

I play with LEGO®

Enterprise-class LEGO®

Enterprise-class LEGO®

A timeline…

(a « career progression »)

In the beginning, there were physical servers

(a few of them only)

Installing an app would take months

- order new server
- wait for it to arrive...
- install the machine in the datacenter (rack, patch, etc) [manual]
- install and configure the OS [manual]
- install and configure the app [manual]

Servers were tailored for peak use

As a result they were idle most of the time
(GARTNER numbers around 80% idle)

☒ Pricey !

To decrease cost, servers were mutualised between applications (or clients)

- ☒ DLL-hell
- ☒ Complexity
- ☒ Security issues
- ☒ Configuration side effects

Servers were all slightly different and hardly reproduceable

- ☒ Patch-levels (security!)
- ☒ Procedure changes

Enter virtualisation

Provisioning a new (virtual) server would now take minutes !

… provided enough capacity was available on the existing physical infrastructure

☑ Provisioning of Apps now takes weeks instead of months

☑ Better resource utilisation of the physical infrastructure

☑ virtualisation layer is an app, there is now an API to provision a new server!

But then we had too many servers to maintain

- ☒ Installation and configuration was still manual in most cases
- ☒ Even with the use of golden image, servers would drift
- ☒ There is only so much you can do with dsh/ssh in a loop... scaling was difficult

Then there was Configuration Management

(and there was much rejoicing)

- ☑ Provisioning and operations were now hands-off after CM recipes are written
- ☑ Systems could be reconfigured on the fly
- ☑ Security patches could be deployed automatically

Scale to a huge number of servers !

(with a very small team of admins)

The On-Calls in the room

NFR über alles

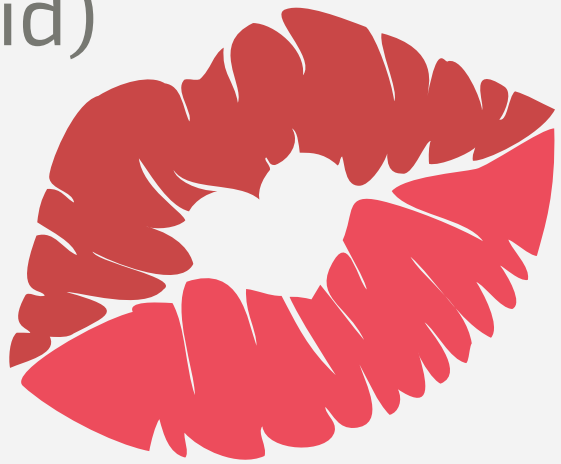
(Non Functional Requirements)

Scalability,
Operability,
Loose coupling

(and other -ilities)

KISS

(Keep It Simple Stupid)



MTTD/MTTR over MTBF

(HA is not always the best solution)

Bloody immature development frameworks !

(Why Ô why can't they use the old one ?)
(Or even just one, pick one)

Us vs. Them

(They should know better !)

The Agile Manifestation

NFR vs. TTM*

(*Time To Market)

It's all about Business Value

(but what about the value of my sleep ?)

Sustainable Pace

(scramble in the war room)

Definition of Done

(Dev is only a small part of the app life)

Finding the balance

DEVOPS !

Everything is a Workflow !

(and I want to deploy 100 times a day)

Everything is a Value Chain !

(and Lean is King !)

And, all of a sudden,
R.I.P.
Configuration Management

Everything is a Container !

(and it works on my machine)

- ☒ Complexity
  (network, orchestration)
- ☒ Security updates
- ☒ Fat !

What else is out there ?

Everything will be an Unikernel ?

(hopefully not !)

- ☒ Less portability
- ☒ Complexity
- ☒ Debuggability

A new paradigm ?

Cloud Functions

(Functions as a Service)

~~Serverless~~

(worst name ever, after devops)

Cloud functions,
As a fancy* block diagram

(*Almost as fancy as an ESB diagram)

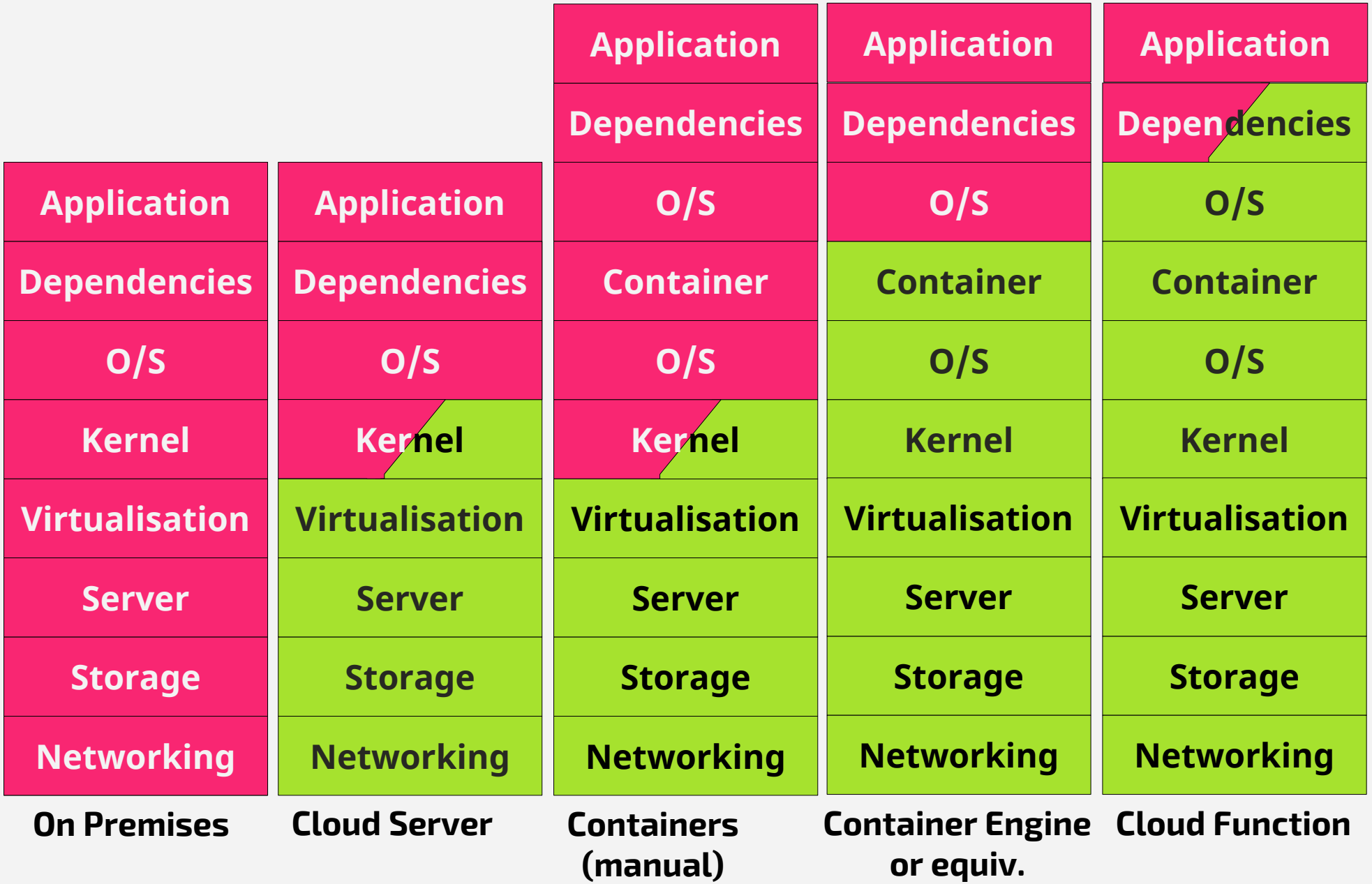
A Business dream?

- ☑ Concentrate on what brings value
- ☑ Cheap ! Pay-as-you-go

A dream architecture ?

- ☑ NFR for free (scalability/orchestration, loose coupling, share nothing, integrated logs, metrics, graphs, etc)
- ☑ Best-of-breed security

Responsability matrix



Managing dependencies security

Pick one of :

https://dependencyci.com/

https://gemnasium.com/

https://requires.io/

https://www.versioneye.com/

(None of these are endorsed by the speaker

pick the one you like/works for you,or find your own !)

What does this mean for Ops ?

(☹ or ☺ ?)

(NoOps v2 ? OpLess ?)

An overly optimistic hope

Ops,
with less « janitorial tasks »,
will be free to concentrate on business value,
and have a sustainable pace too.

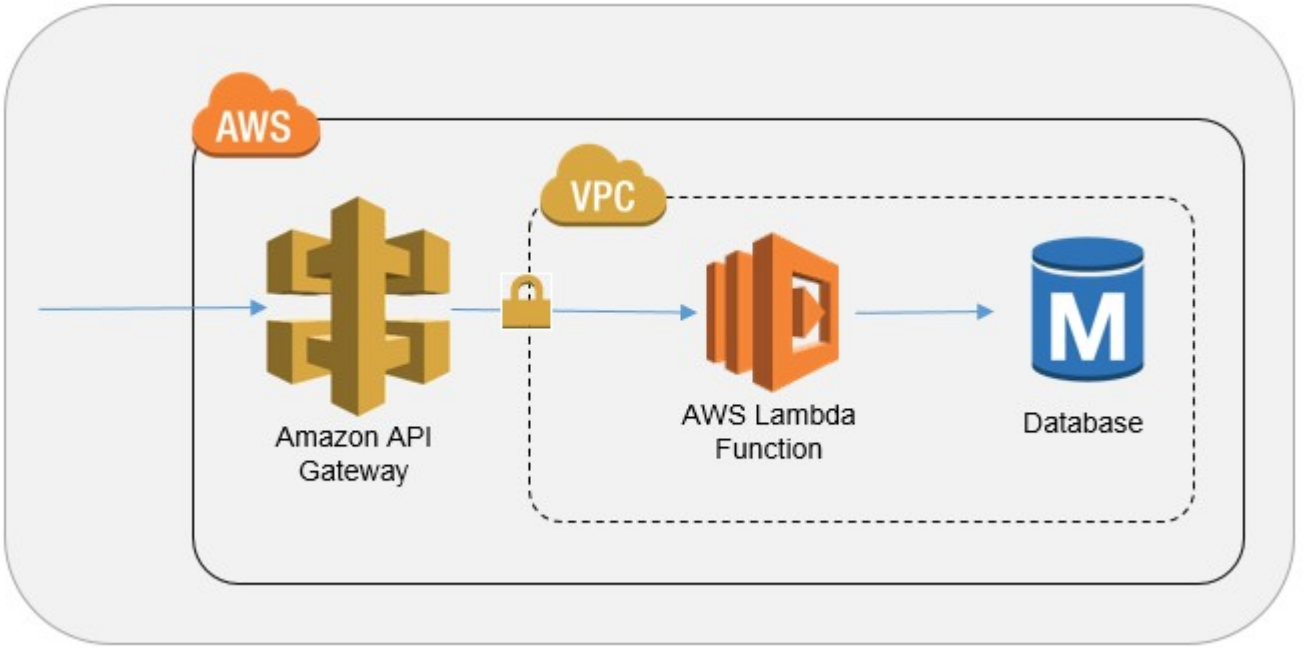
AWS Lambda

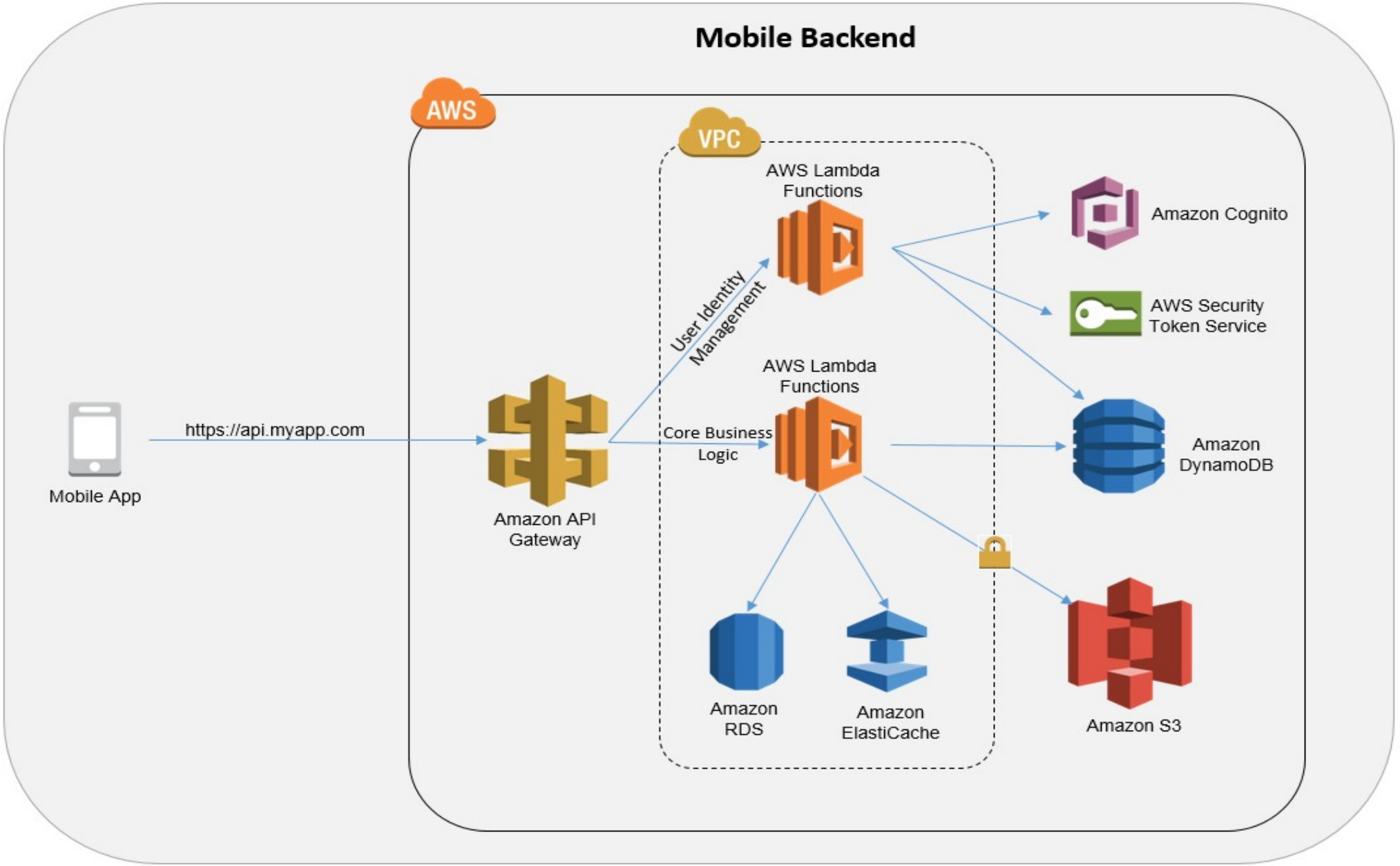
(as an example)

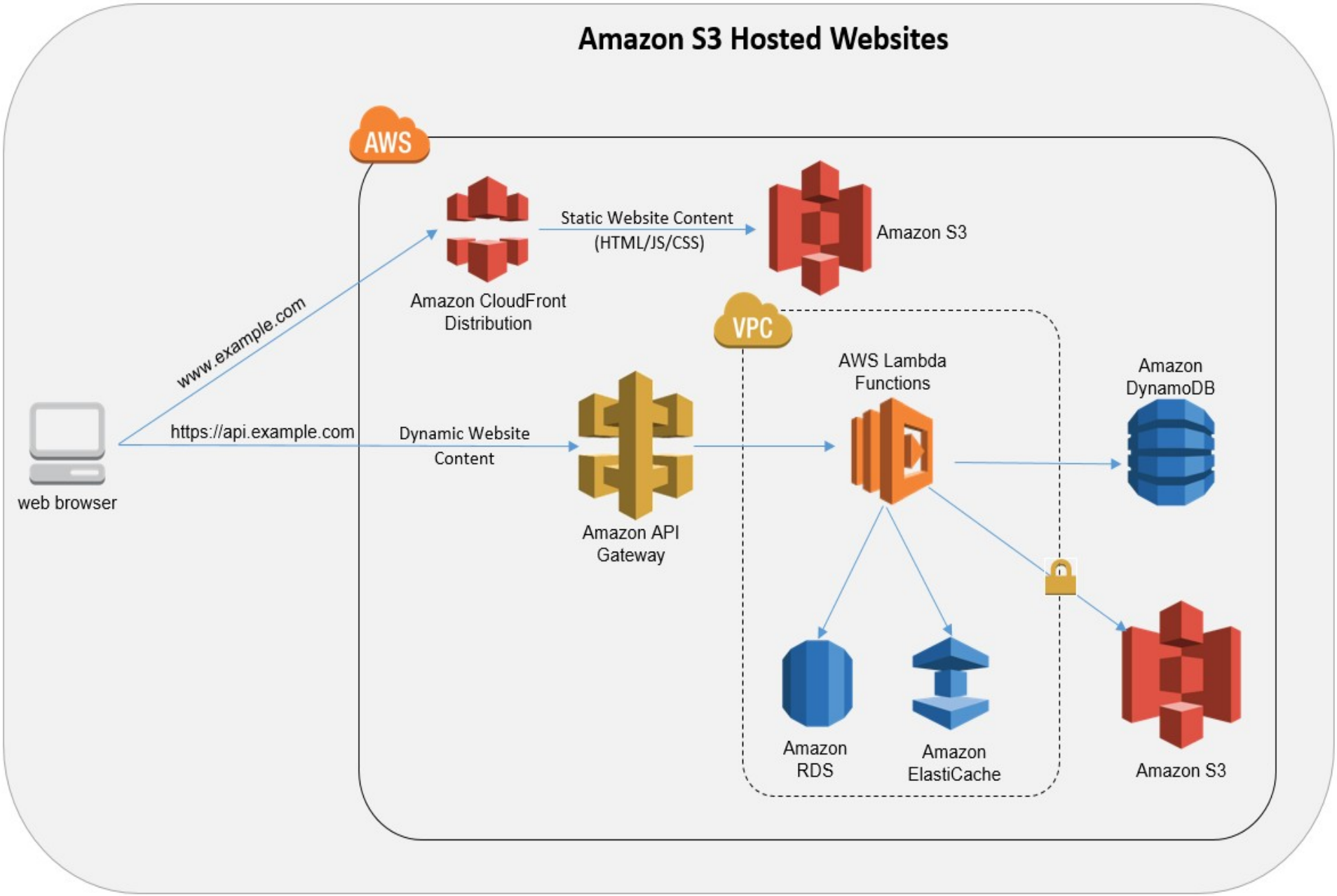
AWS Lambda architecture reference whitepaper examples

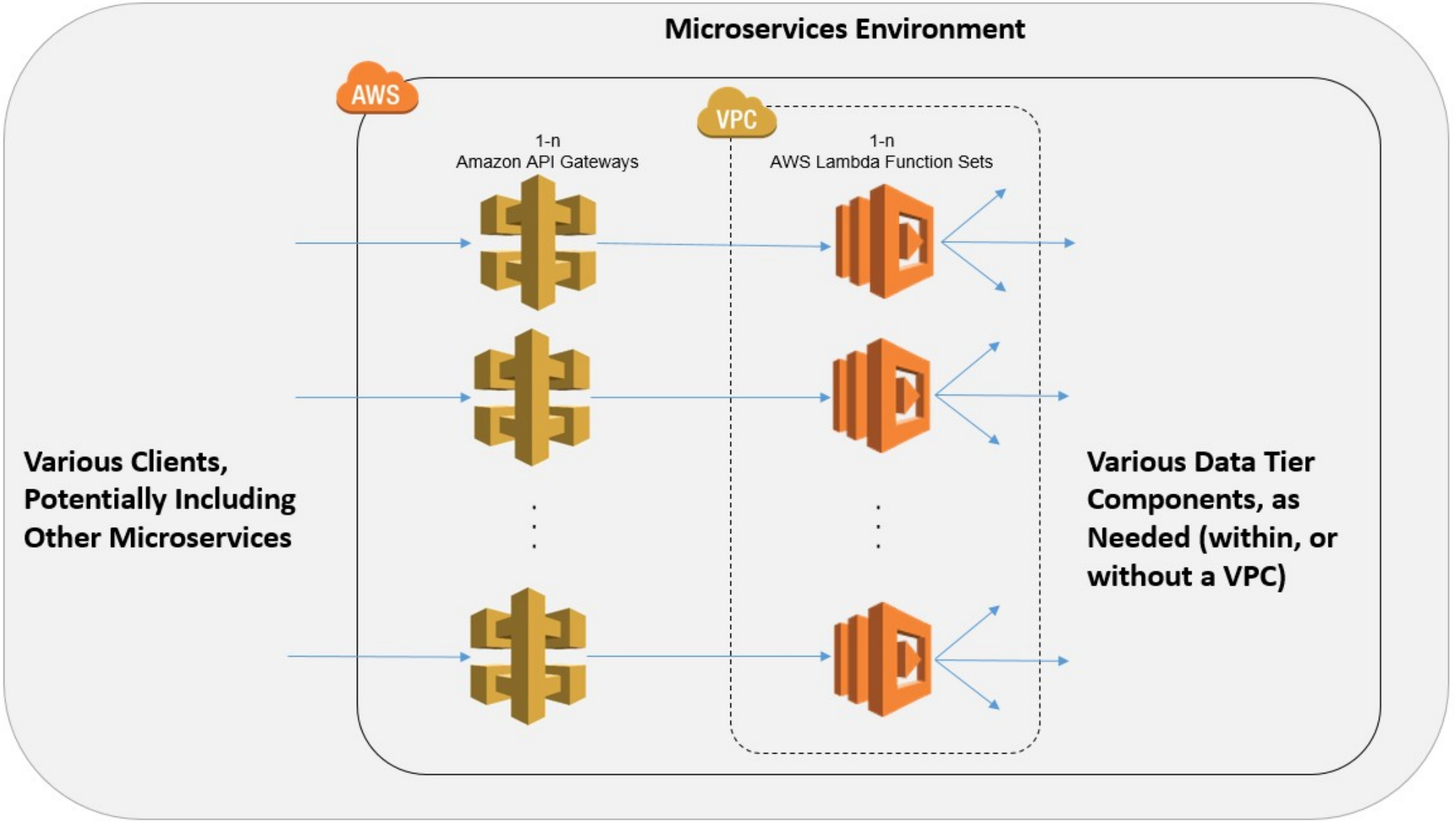
(https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf)

3-tier Lambda App



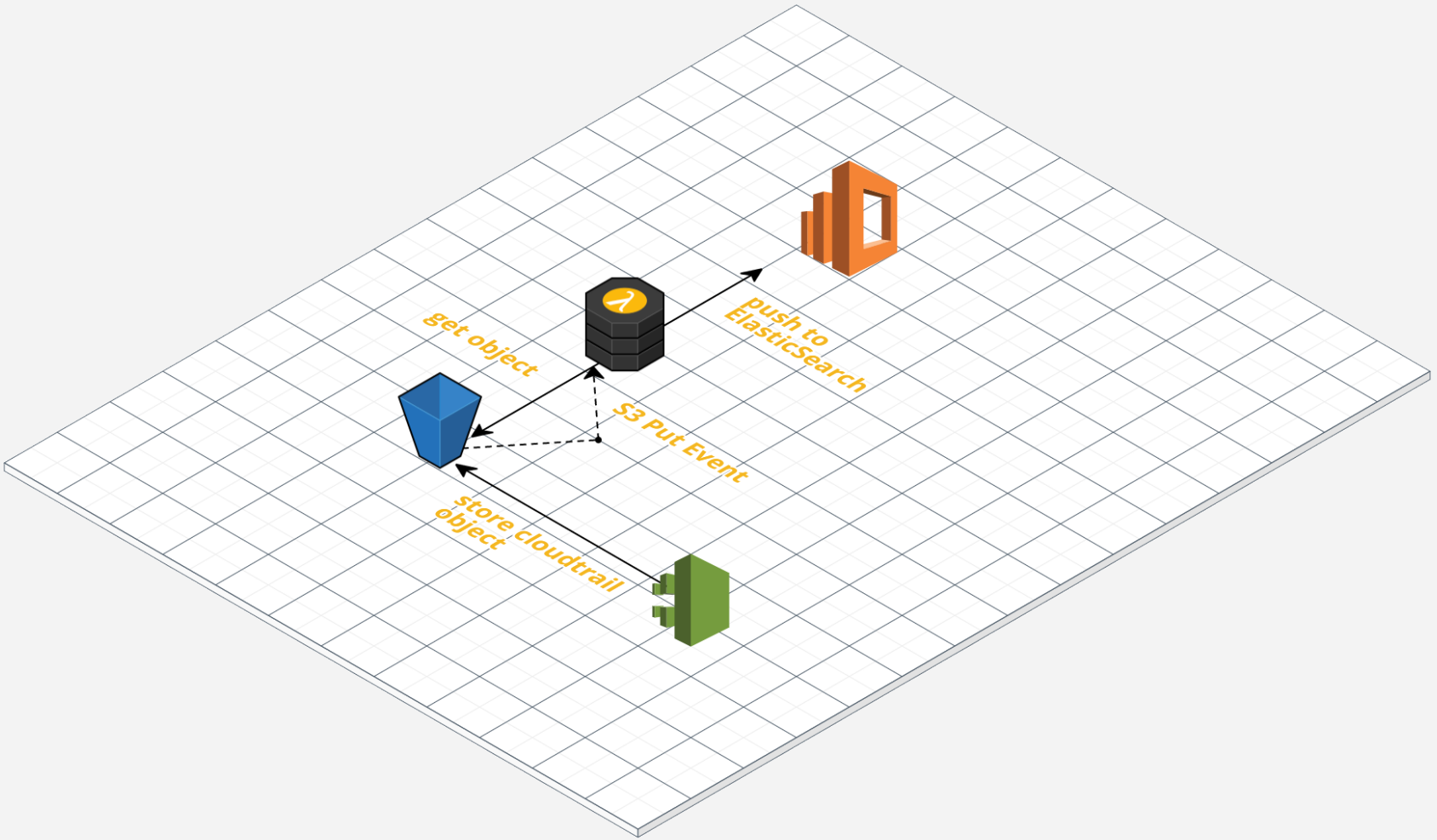

Mobile Backend
AWS
VPC
AWS Lambda Functions
Amazon Cognito
AWS Security Token Service
User Identity Management
AWS Lambda Functions
Core Business Logic
Amazon DynamoDB
https://api.myapp.com
Mobile App
Amazon API Gateway
Amazon RDS
Amazon ElastiCache
Amazon S3


Amazon S3 Hosted Websites
AWS
Amazon CloudFront Distribution
Static Website Content (HTML/JS/CSS)
Amazon S3
www.example.com
VPC
AWS Lambda Functions
Amazon DynamoDB
https://api.example.com
Dynamic Website Content
web browser
Amazon API Gateway
Amazon RDS
Amazon ElastiCache
Amazon S3


Microservices Environment
AWS
VPC
1-n
Amazon API Gateways
1-n
AWS Lambda Function Sets
Various Clients, Potentially Including Other Microservices
Various Data Tier Components, as Needed (within, or without a VPC)

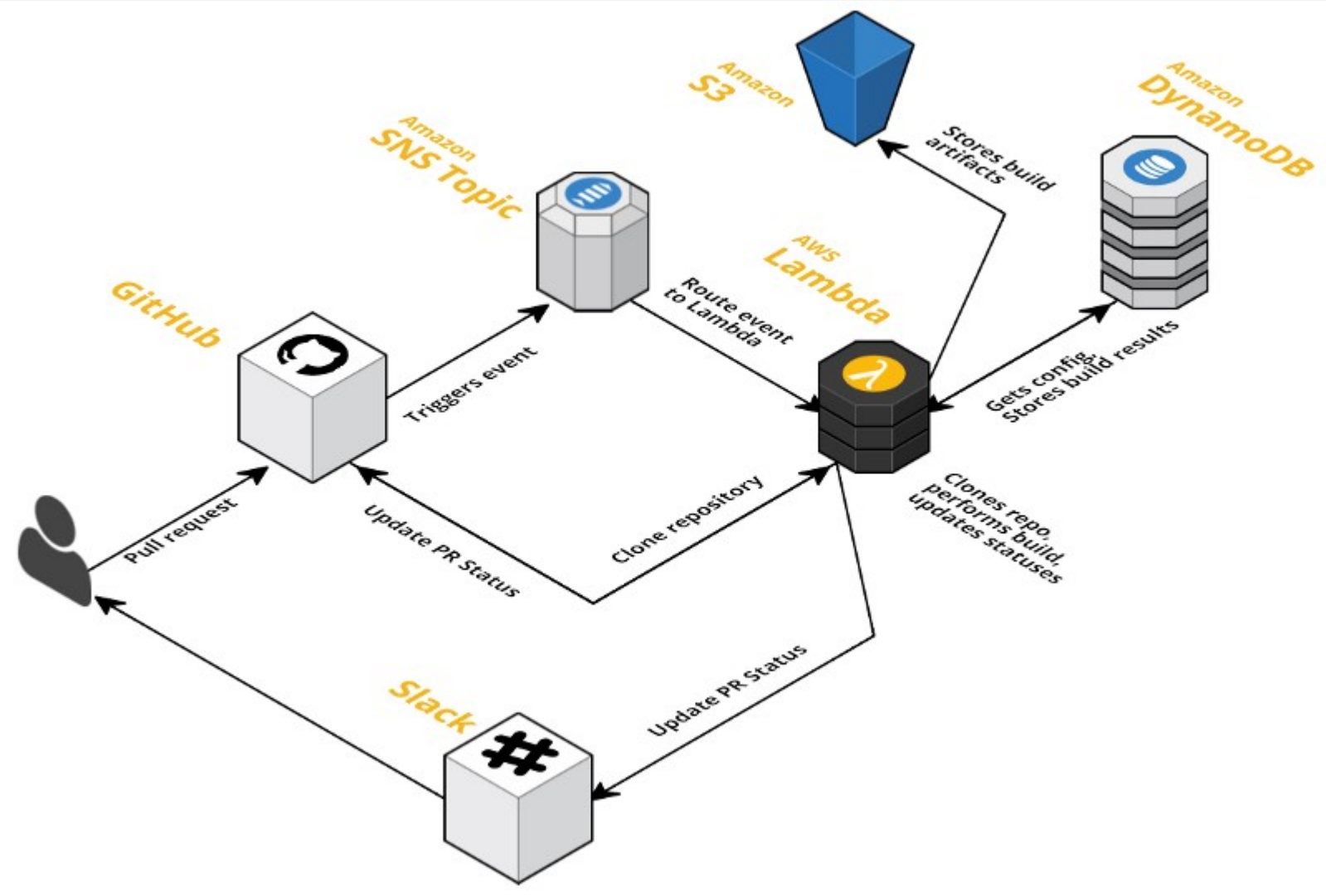
Some real life examples

Lame plug : Cloudtrail to ES



https://github.com/endemics/cloudtrail-es-lambda

LambCi



https://github.com/lambci/lambci

Developping in Lambda ?

- Running lambda node functions locally
  https://github.com/Tim-B/grunt-aws-lambda

- Running lambda python functions locally
  https://github.com/fugue/emulambda

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

Managing/deploying Lambda

- Manually

- CloudFormation

  https://aws.amazon.com/cloudformation/

- Terraform

  https://www.terraform.io/

Dealing with the API Gateway*

- Swagger

- CloudFormation, Terraform

(*The speaker wishes to inform you that at this point,
he is even more flying by the seat of his pants than usual)

The bad

- ☒ Unpredictible function states
- ☒ Tooling
- ☒ Debuggability
- ☒ End-to-end traceability

The Ugly

☒ Vendor lock-in

(See also
http://martinfowler.com/articles/serverless.html#drawbacks)

What if?

A Vendor-neutral framework for Cloud Functions ?

An ideal solution would :

- ☑ Allow you to write your API with swagger
  deploy directly & keep the documentation up-to-date!
- ☑ Shield you from implementation specificities while writing
  your Cloud Functions
- ☑ Deploy your code to multiple cloud vendors automatically
- ☑ Use an abstraction layer for the Persistence layer
- ☑ Provide Mocks for the external services to allow local dev

Questions ?

Thank You !