Serverless

The Future of the Cloud?!

by Bert Ertman

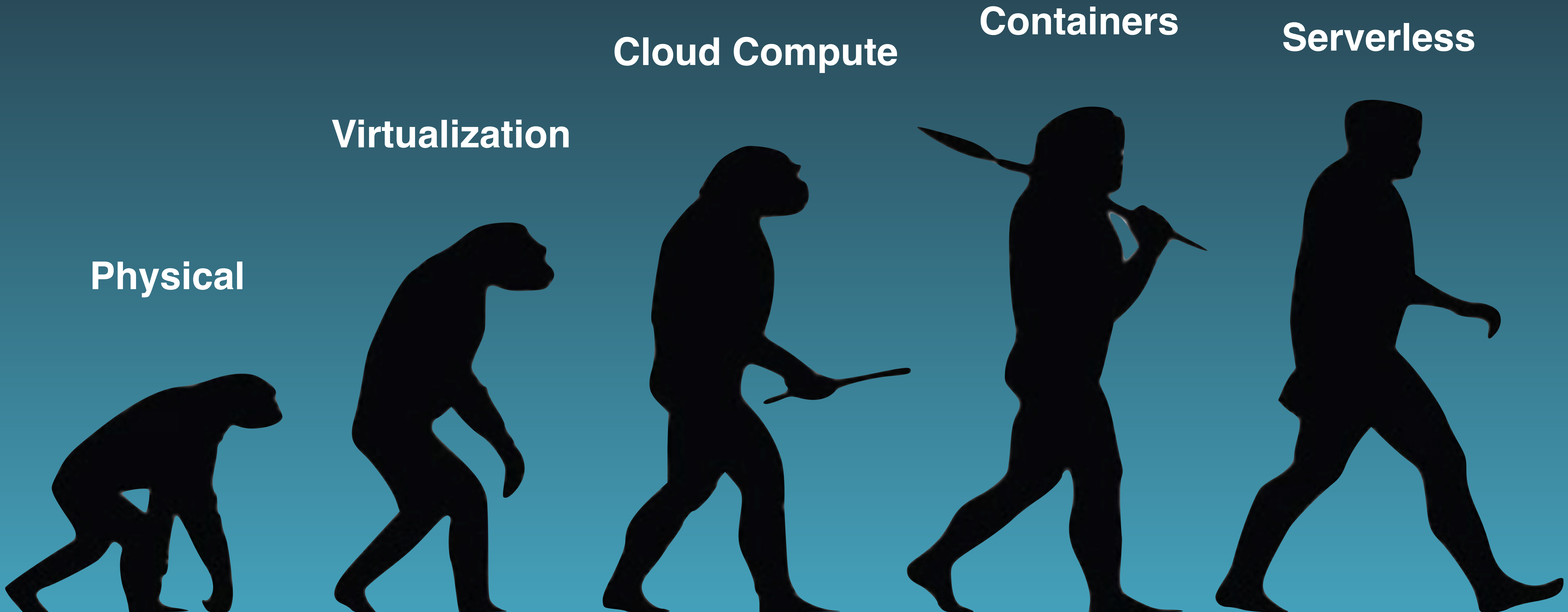
Jfokus DEEP DIVE

luminis
Conversing worlds

@BertErtman

- Fellow, Director of Technology Outreach at Luminis
- Background in all things Java since 1995
- Java Champion, JavaOne Rockstar Speaker, and a Duke's Choice Award Winner
- Involved in architecting and implementing dozens of large scale systems over the past 20 years or so
- Book author for O'Reilly, speaker at many conferences

The Evolution of Compute



WHY?

The case for Serverless

- So far, the cloud is just someone else's computer
- Servers should be treated as cattle, not pets
- PAYGO? or PAYGO&aAYDG?

“No server is easier to manage than no server”

- Werner Vogels
CTO, Amazon

Serverless - what’s in a name?

- Mostly describes what its not…
- Also known as Functions as a Service (FaaS)
- or Function PaaS (fPaaS) as defined by Gartner

- Some people refer to it as Back-end as a Service (BaaS)

Wait! Did someone just say “Back-end as a Service”?

Back-end as a Service??

Serverless Manifesto

- Functions are the unit of deployment and scaling
- No Machines, VMs, or Containers are visible in the programming model
- Permanent storage lives elsewhere
- Scales per request. Users cannot over- or under-provision capacity
- Never pay for idle (no cold servers/containers or their costs)
- Implicitly fault-tolerant because functions can run anywhere
- BYOC - Bring Your Own Code
- Metrics and Logging are a universal right

Main Benefits

- No servers to administer
- Pay for code execution only
- Automatic Scaling

In Other Words…

- We don’t (have to) care about Application Servers
- We don’t (have to) care about Docker

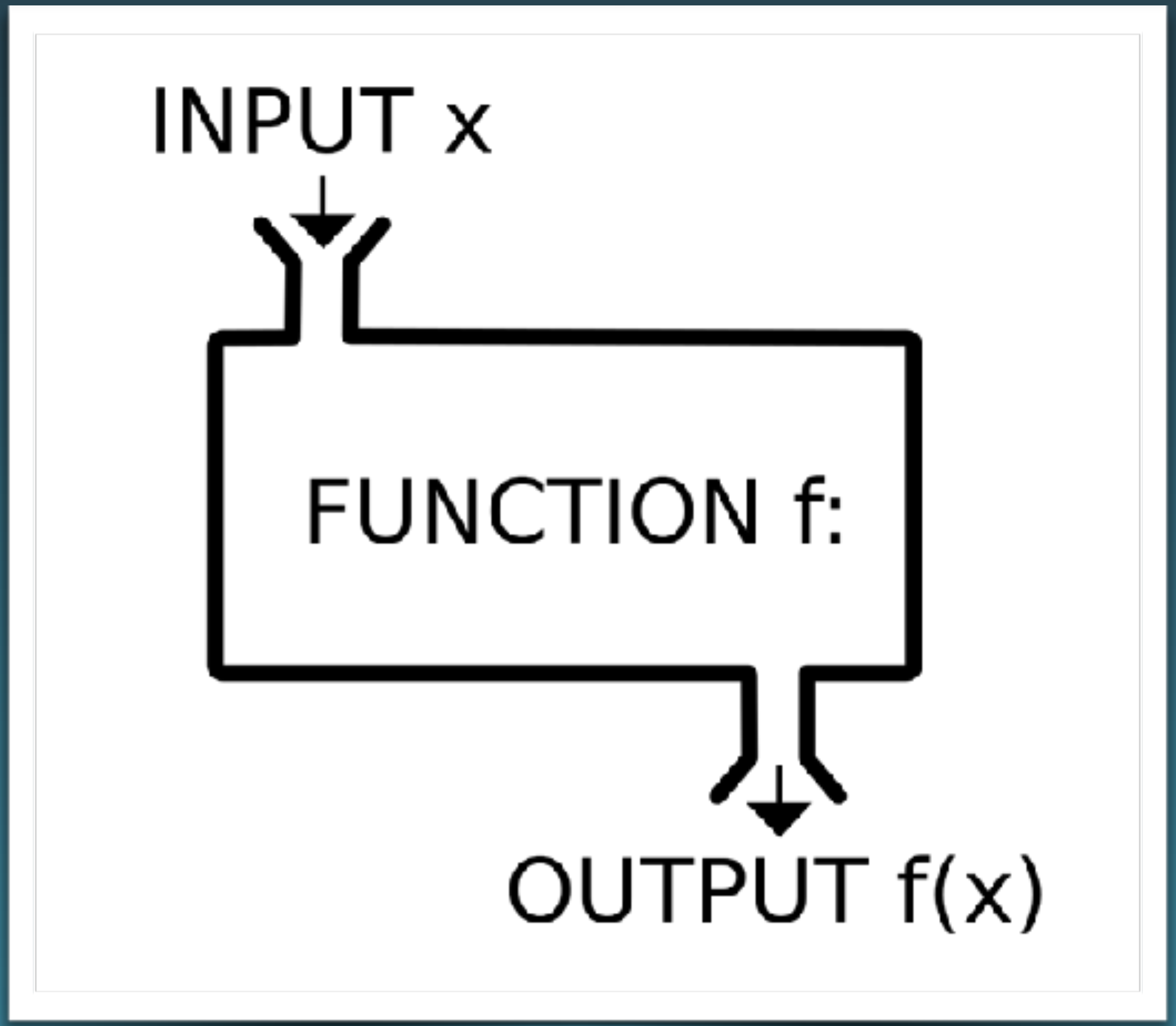
And Best of All

- Very short Time-to-Market
- From development to production in matter of seconds
- Very affordable (PAYG only, no up-front costs)

So far so good, right?

FaaS - Another Look

- Functions as first class citizens
- Run your code (function) in an external, sandboxed, stateless, transient compute container (in the cloud)



But…isn't that PaaS?!

PaaS vs FaaS

- With PaaS, you still manage “applications”
- With PaaS, you take care of scaling
- With PaaS, you manage runtime environment configuration
- With PaaS, you’ll pay for all of the above too

- With FaaS, you have <u>neither</u>

IaaS	CaaS	PaaS	FaaS
🟦 Functions	🟦 Functions	🟦 Functions	🟧 Functions
🟦 Applications	🟦 Applications	🟧 Applications	🟨 Applications
🟦 Runtime	🟦 Runtime	🟨 Runtime	🟨 Runtime
🟦 Containers	🟧 Containers	🟨 Containers	🟨 Containers
🟧 Operating System	🟨 Operating System	🟨 Operating System	🟨 Operating System
🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization
🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware

🟦 Customer Managed

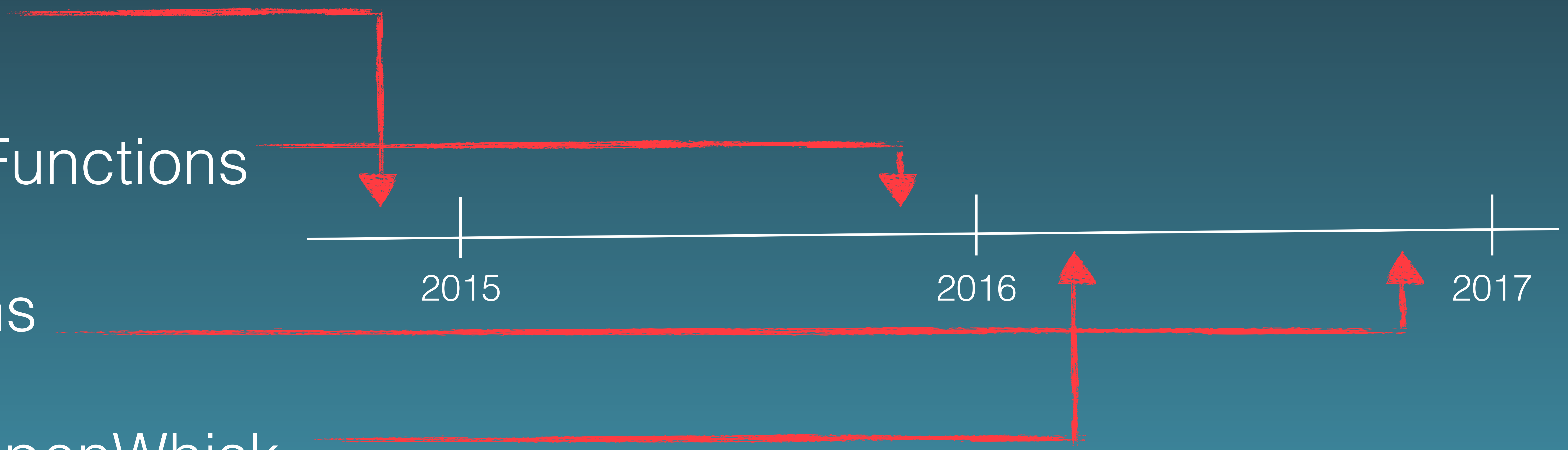
🟧 Customer Managed Unit of Scale

🟨 Vendor Managed

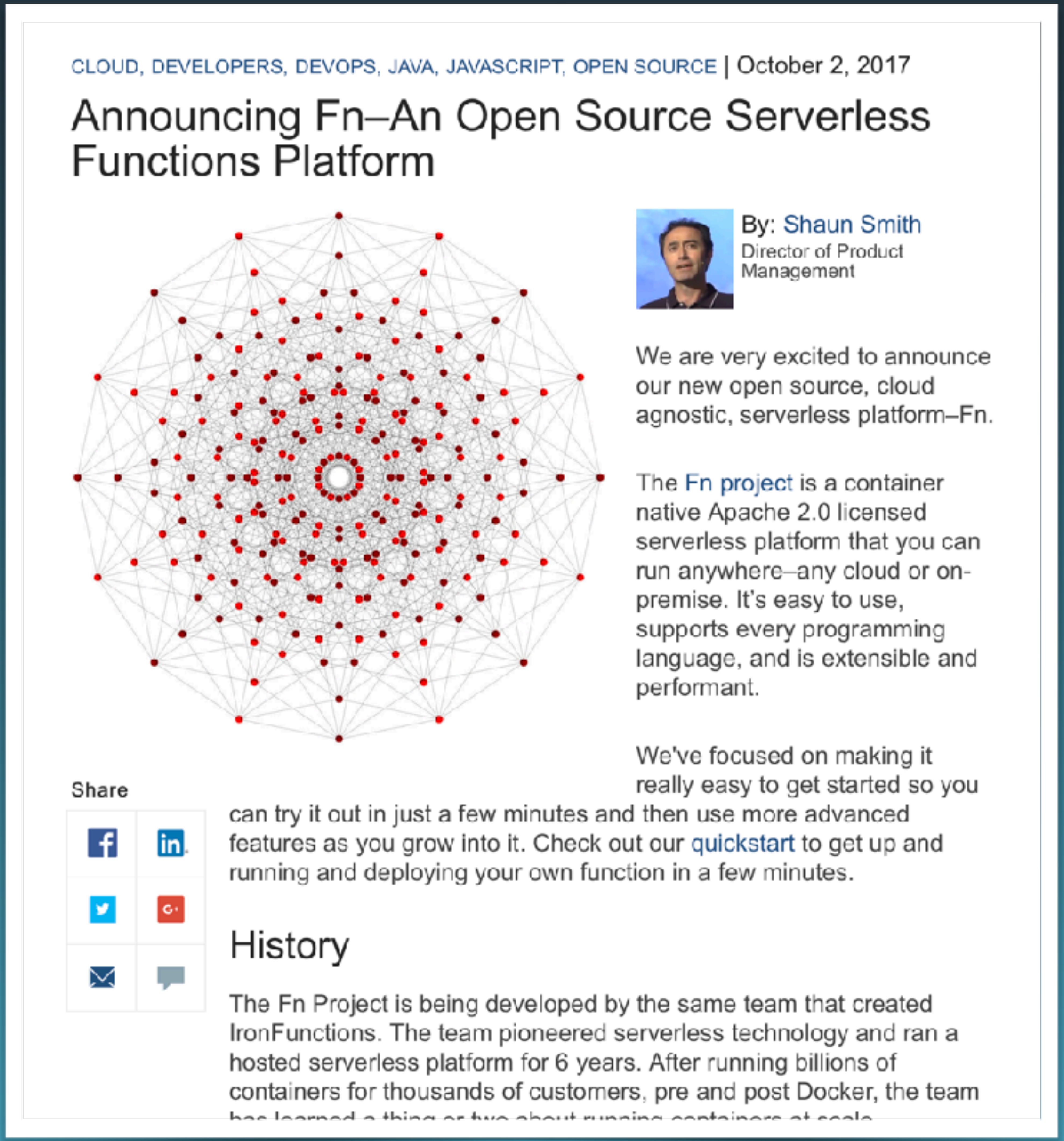
Slide borrowed, with permission, from Arun Gupta @arungupta

Serverless implementations

- Several Cloud vendors have implementations:
  - AWS Lambda
  - Google CloudFunctions
  - Azure Functions
  - IBM BlueMix OpenWhisk



Serverless implementations



CLOUD, DEVELOPERS, DEVOPS, JAVA, JAVASCRIPT, OPEN SOURCE | October 2, 2017

Announcing Fn–An Open Source Serverless Functions Platform

By: Shaun Smith
Director of Product Management

We are very excited to announce our new open source, cloud agnostic, serverless platform–Fn.

The Fn project is a container native Apache 2.0 licensed serverless platform that you can run anywhere–any cloud or on-premise. It's easy to use, supports every programming language, and is extensible and performant.

We've focused on making it really easy to get started so you can try it out in just a few minutes and then use more advanced features as you grow into it. Check out our quickstart to get up and running and deploying your own function in a few minutes.

Share

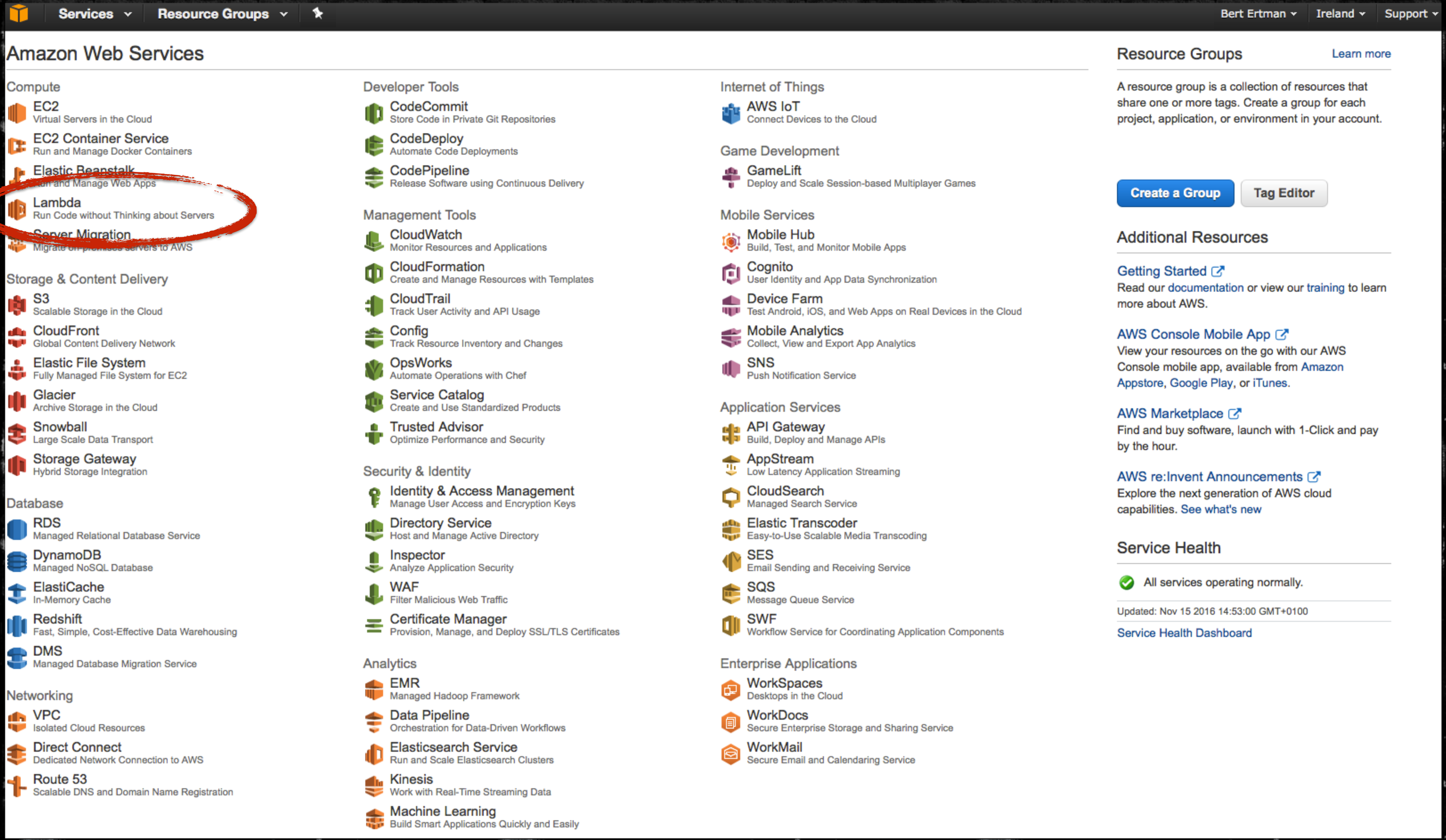
History

The Fn Project is being developed by the same team that created IronFunctions. The team pioneered serverless technology and ran a hosted serverless platform for 6 years. After running billions of containers for thousands of customers, pre and post Docker, the team

https://github.com/fnproject/fn

AWS Lambda

- Event-driven, serverless computing platform provided by Amazon
- Runs code in response to events and automatically manages the compute resources required by that code
- First introduced in Nov 2014
- Part of the Amazon Web Services offerings



Amazon Web Services

Compute

EC2
Virtual Servers in the Cloud

EC2 Container Service
Run and Manage Docker Containers

Elastic Beanstalk
Run and Manage Web Apps

Lambda
Run Code without Thinking about Servers

Server Migration
Migrate on-premises servers to AWS

Storage & Content Delivery

S3
Scalable Storage in the Cloud

CloudFront
Global Content Delivery Network

Elastic File System
Fully Managed File System for EC2

Glacier
Archive Storage in the Cloud

Snowball
Large Scale Data Transport

Storage Gateway
Hybrid Storage Integration

Database

RDS
Managed Relational Database Service

DynamoDB
Managed NoSQL Database

ElastiCache
In-Memory Cache

Redshift
Fast, Simple, Cost-Effective Data Warehousing

DMS
Managed Database Migration Service

Networking

VPC
Isolated Cloud Resources

Direct Connect
Dedicated Network Connection to AWS

Route 53
Scalable DNS and Domain Name Registration

Developer Tools

CodeCommit
Store Code in Private Git Repositories

CodeDeploy
Automate Code Deployments

CodePipeline
Release Software using Continuous Delivery

Management Tools

CloudWatch
Monitor Resources and Applications

CloudFormation
Create and Manage Resources with Templates

CloudTrail
Track User Activity and API Usage

Config
Track Resource Inventory and Changes

OpsWorks
Automate Operations with Chef

Service Catalog
Create and Use Standardized Products

Trusted Advisor
Optimize Performance and Security

Security & Identity

Identity & Access Management
Manage User Access and Encryption Keys

Directory Service
Host and Manage Active Directory

Inspector
Analyze Application Security

WAF
Filter Malicious Web Traffic

Certificate Manager
Provision, Manage, and Deploy SSL/TLS Certificates

Analytics

EMR
Managed Hadoop Framework

Data Pipeline
Orchestration for Data-Driven Workflows

Elasticsearch Service
Run and Scale Elasticsearch Clusters

Kinesis
Work with Real-Time Streaming Data

Machine Learning
Build Smart Applications Quickly and Easily

Internet of Things

AWS IoT
Connect Devices to the Cloud

Game Development

GameLift
Deploy and Scale Session-based Multiplayer Games

Mobile Services

Mobile Hub
Build, Test, and Monitor Mobile Apps

Cognito
User Identity and App Data Synchronization

Device Farm
Test Android, iOS, and Web Apps on Real Devices in the Cloud

Mobile Analytics
Collect, View and Export App Analytics

SNS
Push Notification Service

Application Services

API Gateway
Build, Deploy and Manage APIs

AppStream
Low Latency Application Streaming

CloudSearch
Managed Search Service

Elastic Transcoder
Easy-to-Use Scalable Media Transcoding

SES
Email Sending and Receiving Service

SQS
Message Queue Service

SWF
Workflow Service for Coordinating Application Components

Enterprise Applications

WorkSpaces
Desktops in the Cloud

WorkDocs
Secure Enterprise Storage and Sharing Service

WorkMail
Secure Email and Calendaring Service

Resource Groups
Learn more

A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.

Create a Group
Tag Editor

Additional Resources

Getting Started
Read our documentation or view our training to learn more about AWS.

AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.

AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.

AWS re:Invent Announcements
Explore the next generation of AWS cloud capabilities. See what's new

Service Health

✓ All services operating normally.

Updated: Nov 15 2016 14:53:00 GMT+0100

Service Health Dashboard

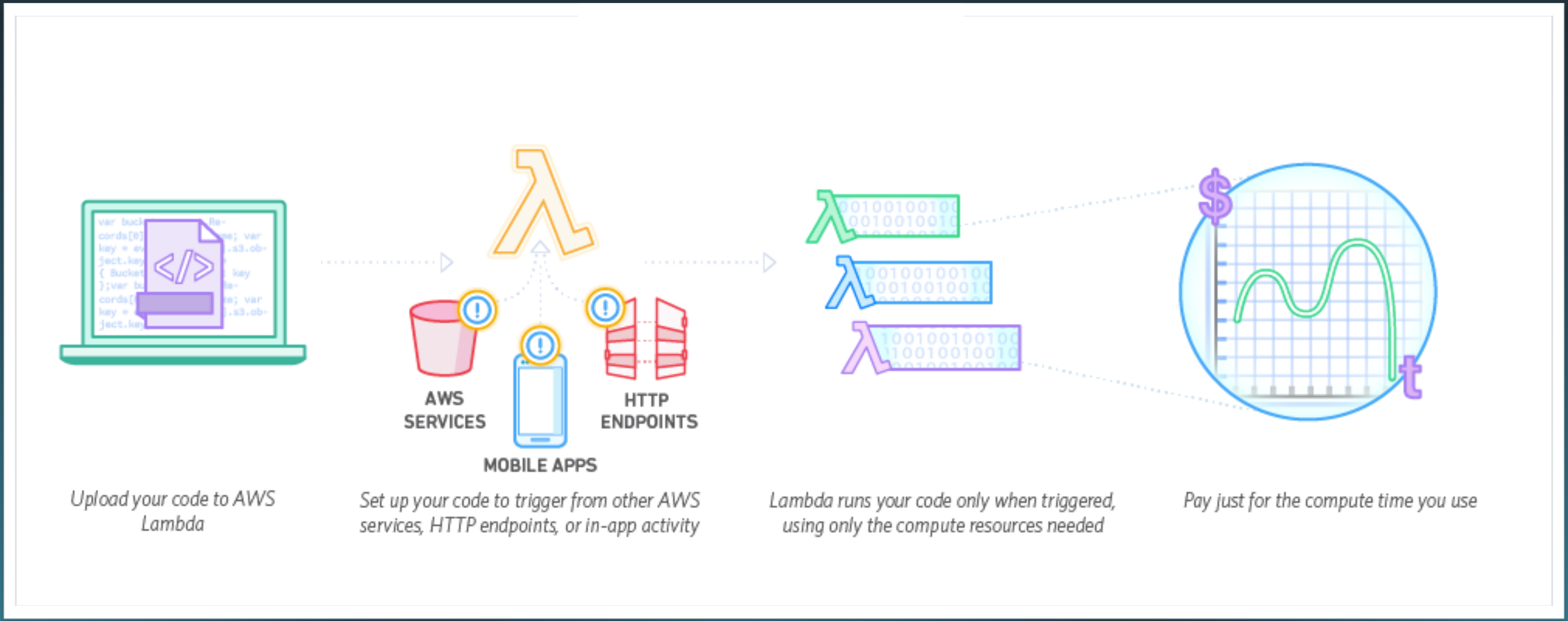
Comparing AWS Compute Types

- EC2 (IaaS)
- EC2 Container Service (CaaS)
- Elastic Beanstalk (PaaS)
- Lambda (FaaS)

Runtime Support

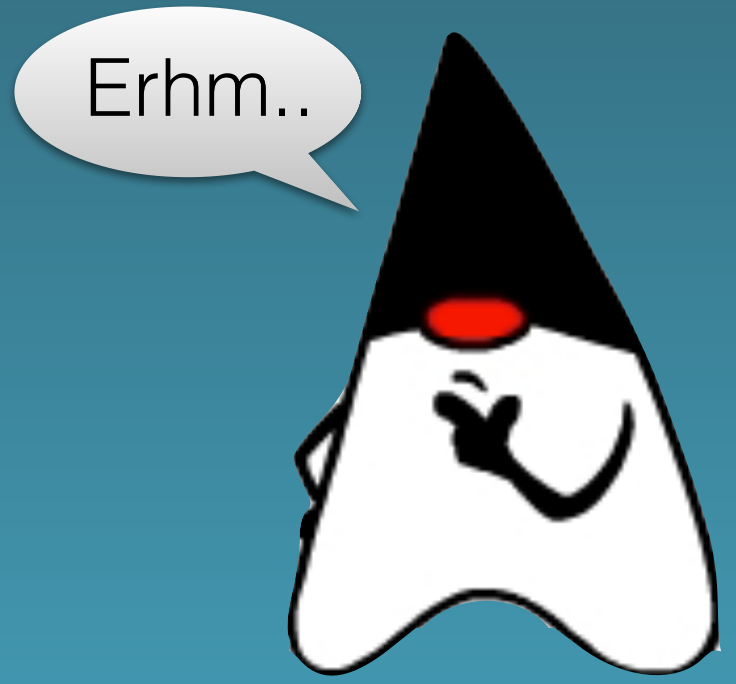
- Python 2.7
- NodeJS 4.3
- Java 8
- C#
- Go

How it works


AWS SERVICES
MOBILE APPS
HTTP ENDPOINTS
Upload your code to AWS Lambda
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity
Lambda runs your code only when triggered, using only the compute resources needed
Pay just for the compute time you use

Upload **Trigger** **Run** **Pay**

lambda



Getting Started

- Signup for a free AWS account
- Install Amazon AWS plugin for Eclipse
    - comes with Amazon Java SDK for AWS
- Install AWS CLI (optional)
- Edit code offline and upload artefact to AWS Lambda

demo

Lambdas are event-driven

Many Event Sources

API Gateway

Amazon S3

Amazon DynamoDB

Amazon Aurora

Amazon Simple Notification Service

Amazon Simple Email Service

Amazon Cognito

Amazon CloudWatch

Amazon Kinesis Streams

AWS CodeCommit

AWS CloudFormation

AWS Config

Amazon Lex

Many Event Sources

HTTP(s) requests

API Gateway

CRUD events on data sources

Amazon S3

Amazon DynamoDB

Amazon Aurora

messaging events

Amazon Simple Notification Service

Amazon Simple Email Service

Amazon Cognito

log/stream processing

Amazon CloudWatch

Amazon Kinesis Streams

commit hooks / AWS services setup

AWS CodeCommit

AWS CloudFormation

configuration mgmt

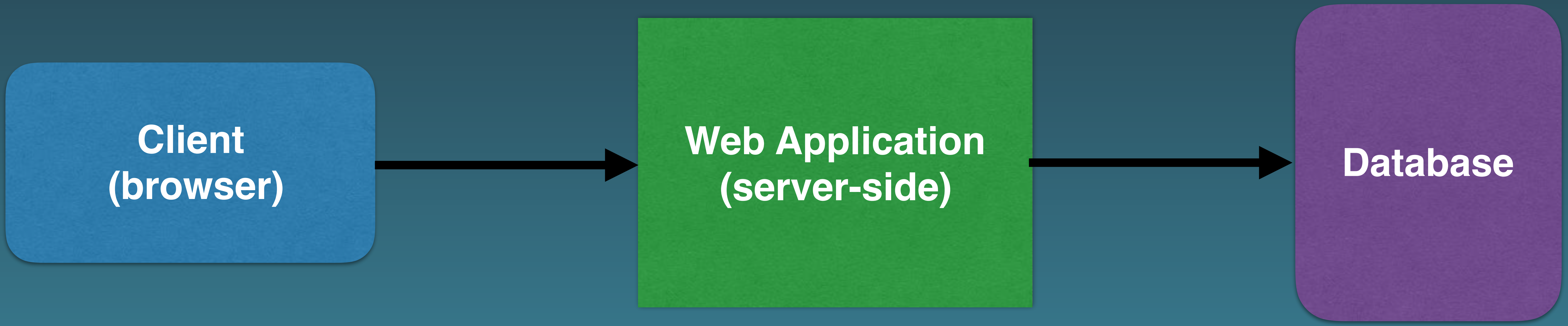
AWS Config

voice & text

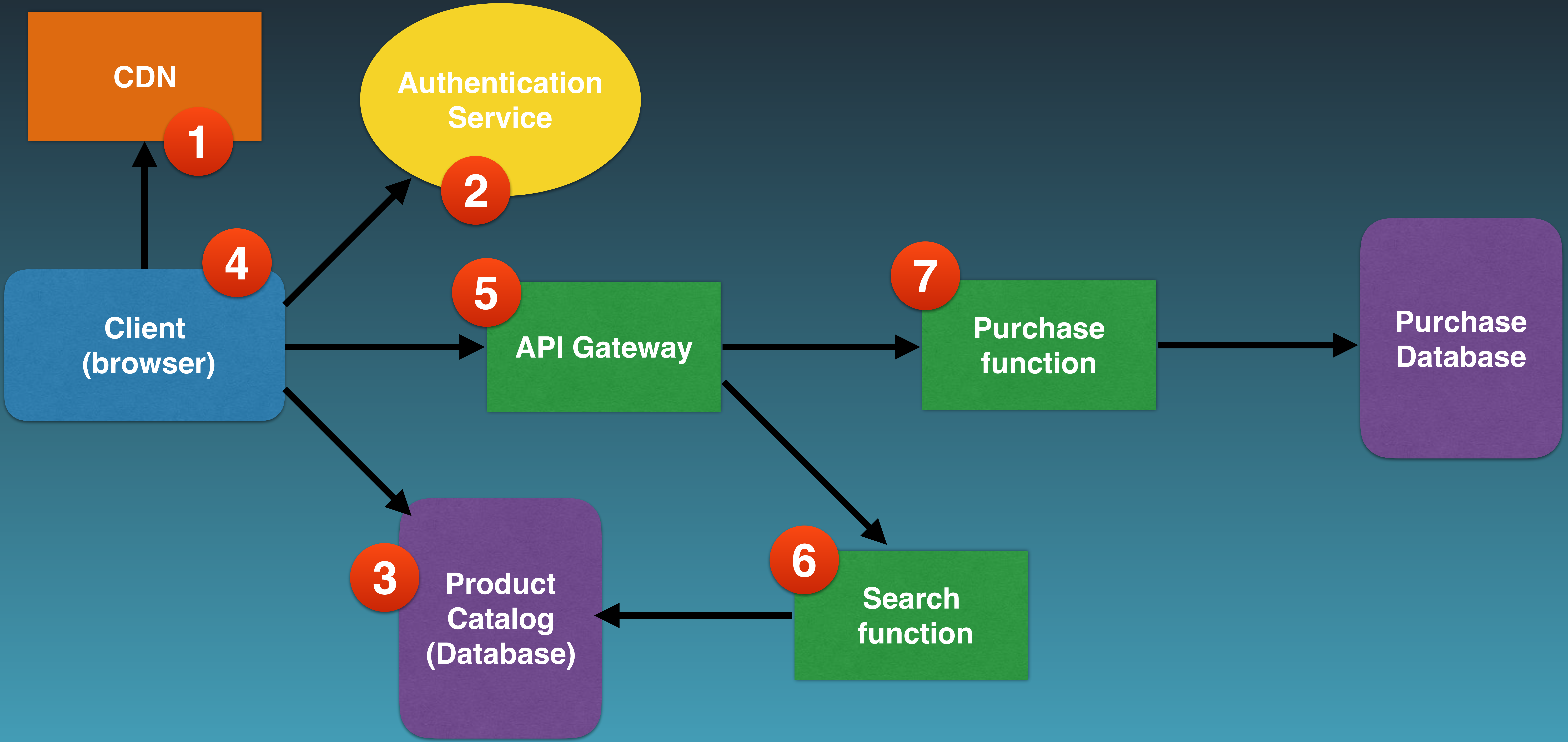
Amazon Lex

Rethinking Traditional Architectural Concepts

Rethinking Traditional Architectural Concepts



Rethinking Traditional Architectural Concepts



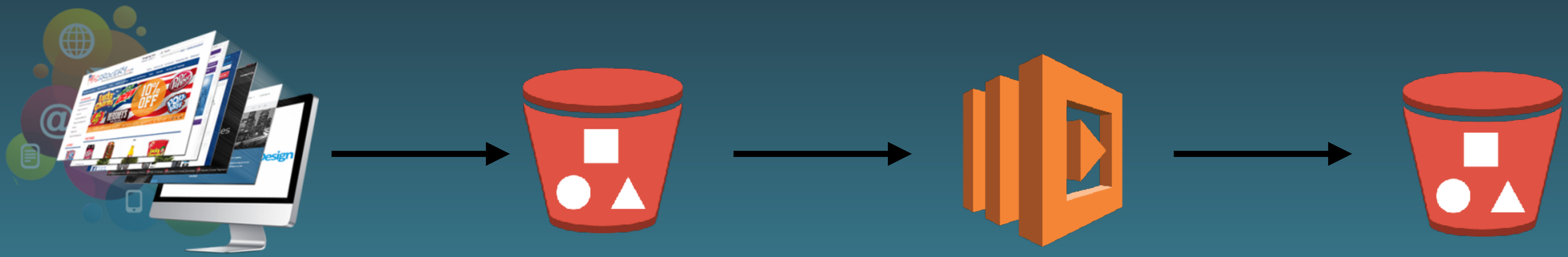
Just because you can might not always be the right reason

Serverless **>** FaaS

Example: Event-based processing

- Respond to incoming data, such as an S3 Bucket insert
- Useful for data/stream processing, MapReduce, or batch processing

Typical Scenario



AWS S3

- Amazon Simple Storage Service (S3) is object based storage designed for extreme scalability
- Primary storage type for cloud-native applications

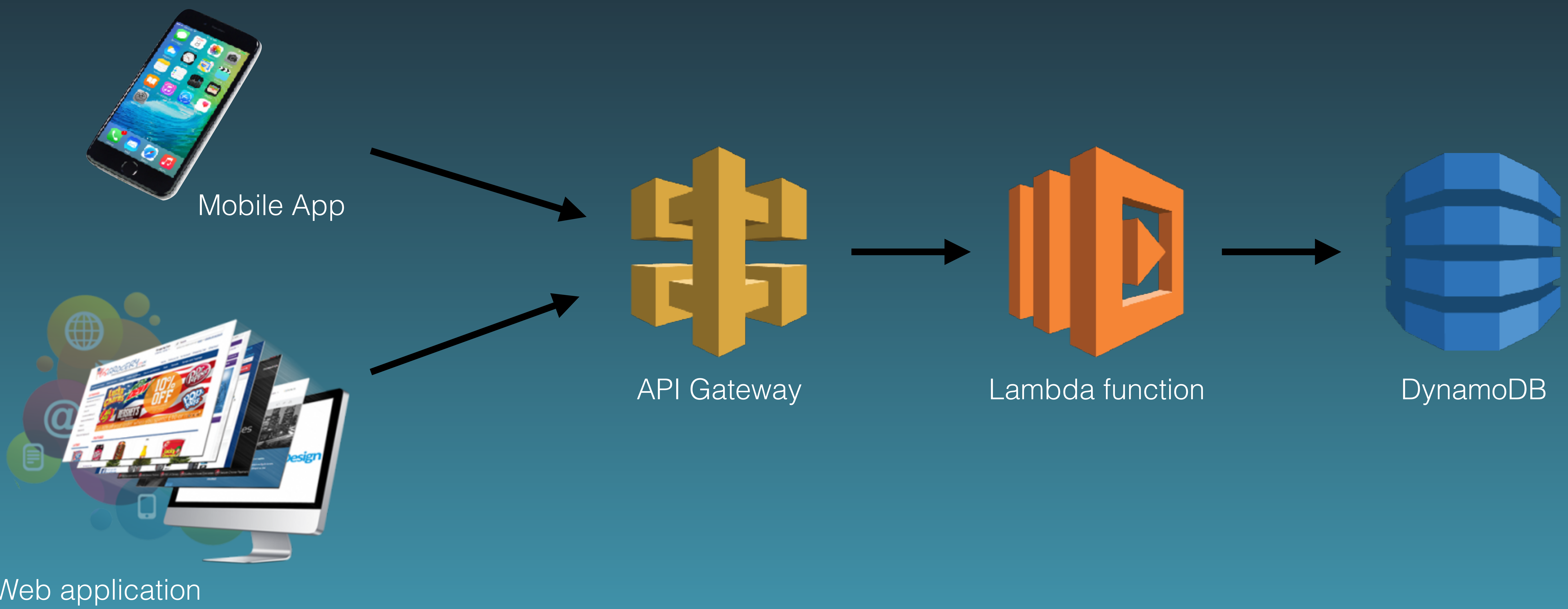
demo

Dude, this is just database triggers all over again!

Better Example: BaaS

- Backend-as-a-Service for a (mobile/web) app
- Responds to incoming HTTP GET/POST requests
- Stateless

Typical Scenario

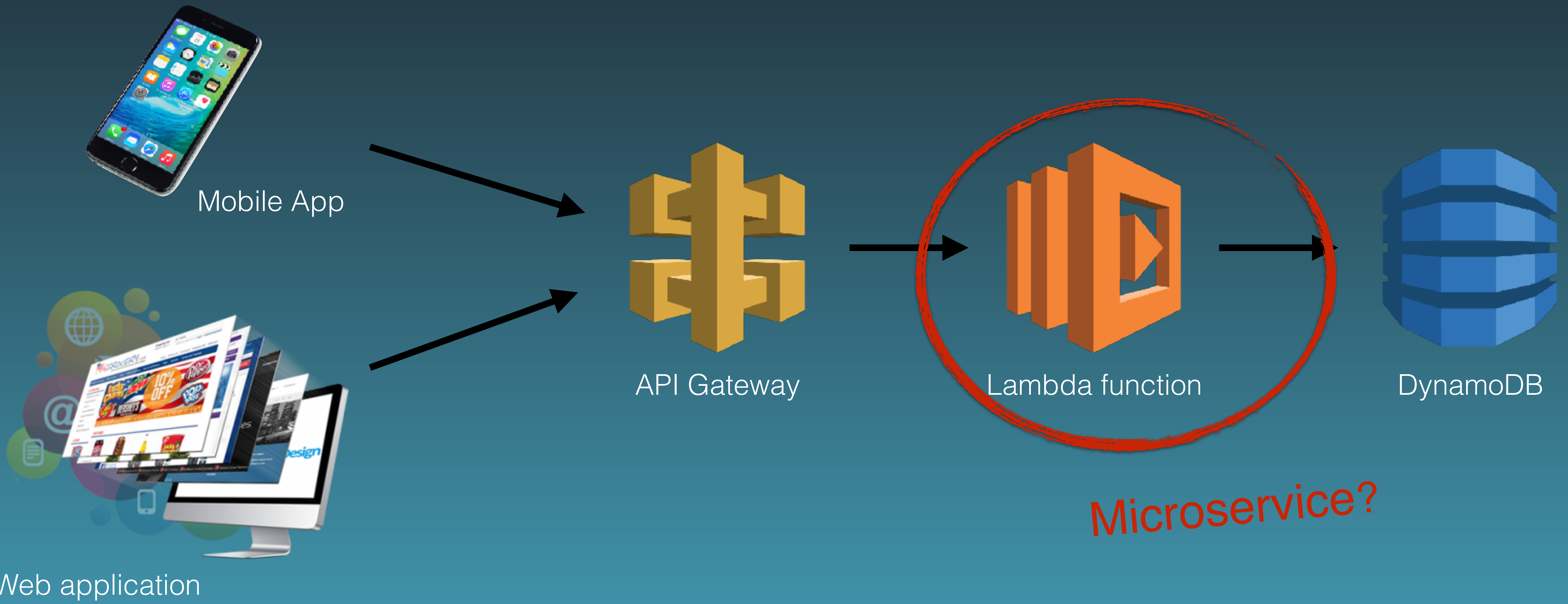


AWS API Gateway

- Managed service to create/publish/maintain secure APIs at scale
- Define REST APIs for Lambdas
- Documentation support for APIs (Swagger)

demo

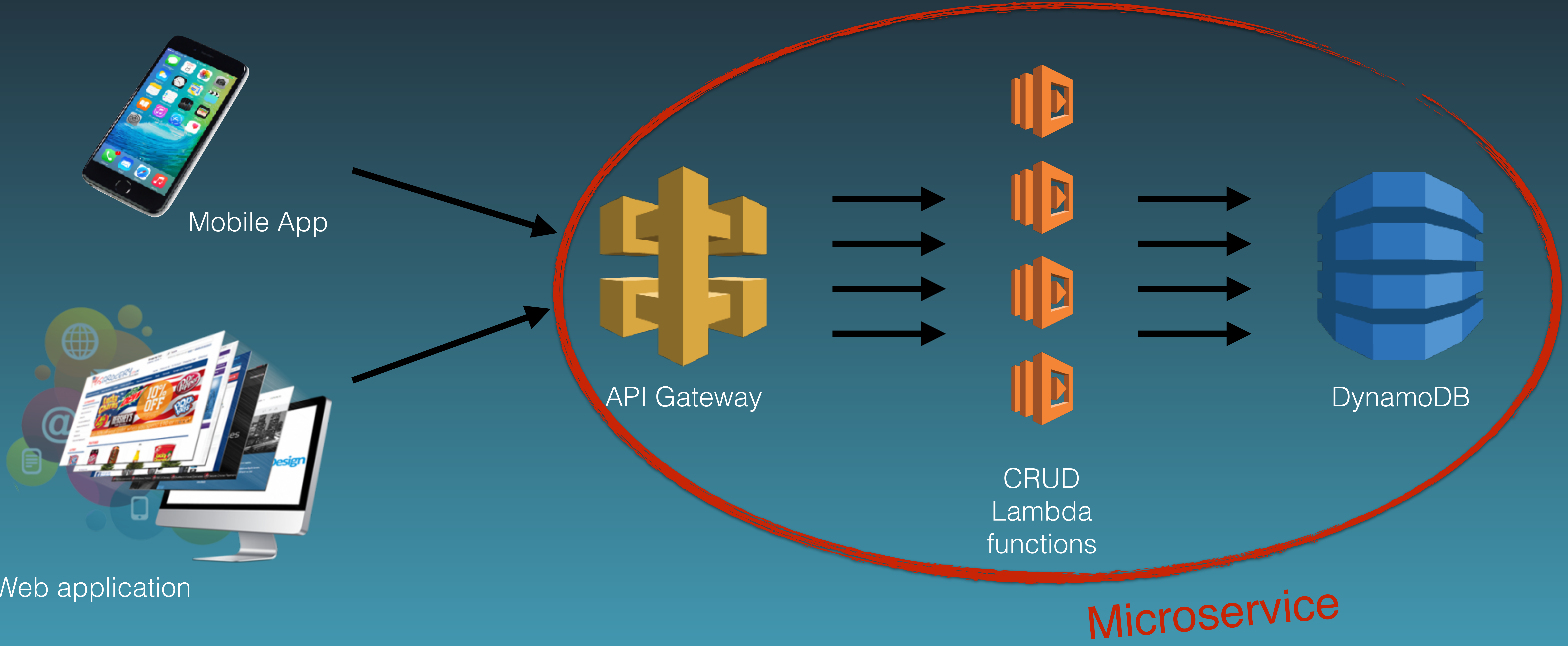
Typical Scenario



Are Lambda functions Microservices?

- Similarities:
  - Do one thing, and one thing well
  - Event-based interaction == choreography model
- Differences:
  - One Lambda is equal to one action == NanoService
  - Microservice == bounded context of actions with autonomous storage

Typical Scenario



Developing ZeroOps Serverless Microservices running in the Cloud

using

AI & Machine Learning

Service Composition

- Most scenarios require other services, such as storage, messaging, mail, compute, and analysis, etc

- Amazon recently launched SAM
  - Semi-standard DSL for Serverless computing (yaml/json)
  - Extension for AWS CloudFormation

AWS CloudFormation

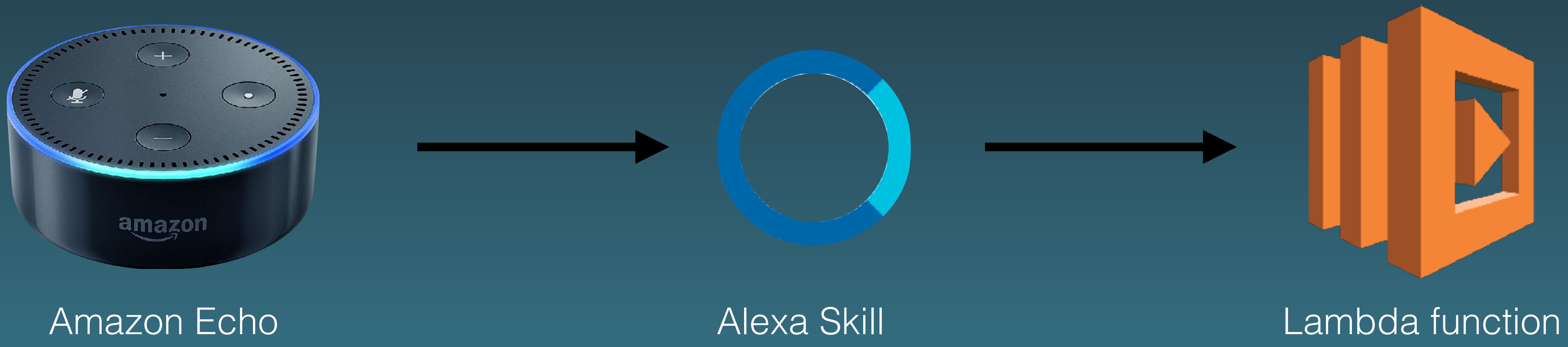
- Managed service to create/manage/provision collections of AWS resources
  - create/manage stacks from templates
  - figures out deployment order automagically

demo

Other Use-Cases

- Implement custom CI/CD pipeline on AWS
- Bots
- Voice Control :)

Typical Scenario



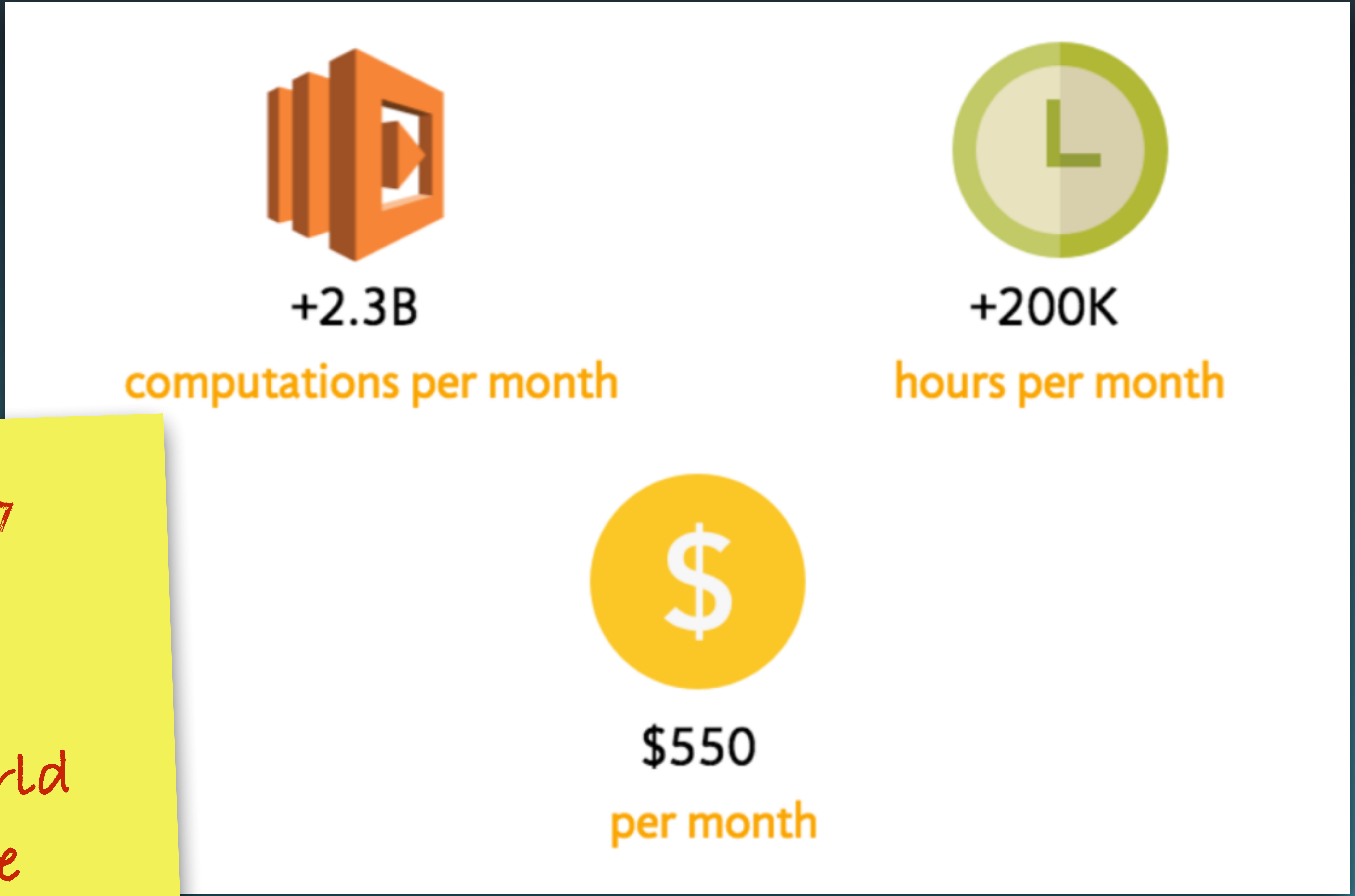
demo

Lambdas can be monetized as well…

Going beyond hello world

```
>HELLO WORLD!
>_
```

What Expedia is doing with Lambda

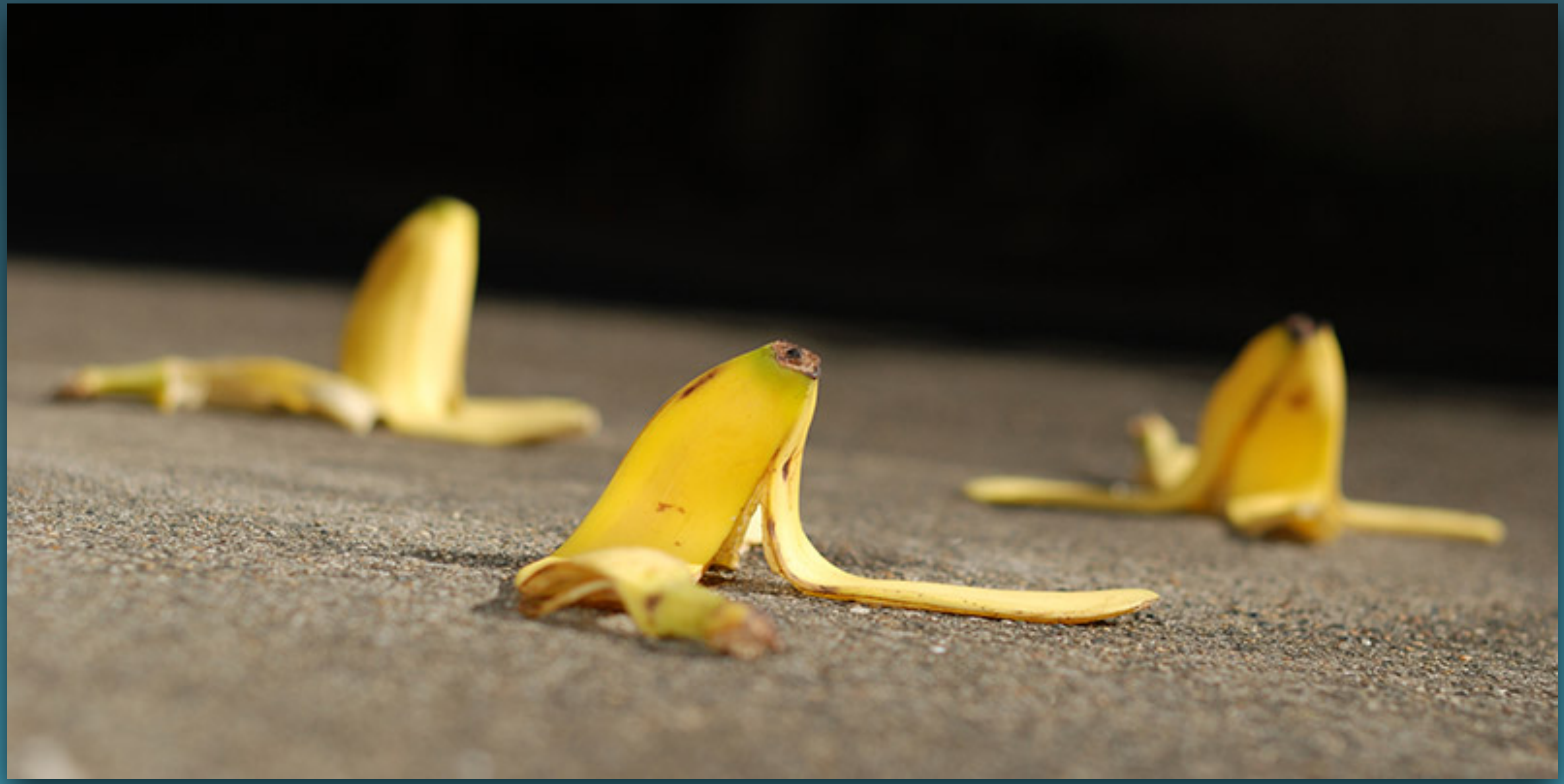


beyond
Hello World
example

source: AWS re:invent 2016: Serverless Computing Patterns at Expedia (SVR306)

Some pointers beyond hello world…

- Logging
- Testing
- Advanced configuration
- CI/CD - how to integrate?
- Upload size
- Dealing with lock-in
- Performance



Logging

- Simple print to console statements will end up in the application log
    - which will be picked up by CloudWatch
- Context API offers a Logger
- Allows for log4J configuration

What about Testing?

- Functions are easy to unit test
  - stateless
  - sometimes just a few dependencies (that can be mocked)

What about **Integration** Testing?

Integration Testing

- Requires you to have (or simulate) the environment and infrastructure underneath your lambda function
- You can’t run a local AWS cloud* on your laptop or build server
- API Gateway supports staging
- Max. 1000 parallel running Lambdas in production (default)

*) Some AWS services can be mocked locally

Advanced Configuration

- Externalize configuration using Environment Variables
- AWS Resource and Role permissions and configuration thereof can be a real pain in the butt sometimes
- AWS API Gateway is cumbersome to configure as well
- SAM offers some relieve
- AWS online documentation mostly sucks :(

Deployment

- AWS Lambda Eclipse IDE plugin
- AWS Web Console
- AWS CLI

CI/CD

- Jenkins (can be setup and run from EC2 instance)
- AWS Lambda plugin
- Trigger from version control (GitHub or CodeCommit) or S3 bucket upload

Upload Size

- AWS Java SDK plus third party libs: ~63MB
  - Eclipse AWS plugin adds it in by default
  - Solution: manually add only minimum required separate libs in pom

🔐 Lock-in

- You tie into AWS specific solutions easily:
  - documentation, metrics, and monitoring (CloudWatch)
  - minimize risk by separating implementation code from the function handler
  - but you probably tie into more AWS specific solutions beyond lambdas like: API Gateway, S3, SNS, SQS, SES, etc.

Performance

- JavaScript and PHP are interpreted on the go and hardly incur a start-up time performance penalty

- Java does have a performance penalty in firing up the JVM, but depending on your usage this doesn't have to be a problem

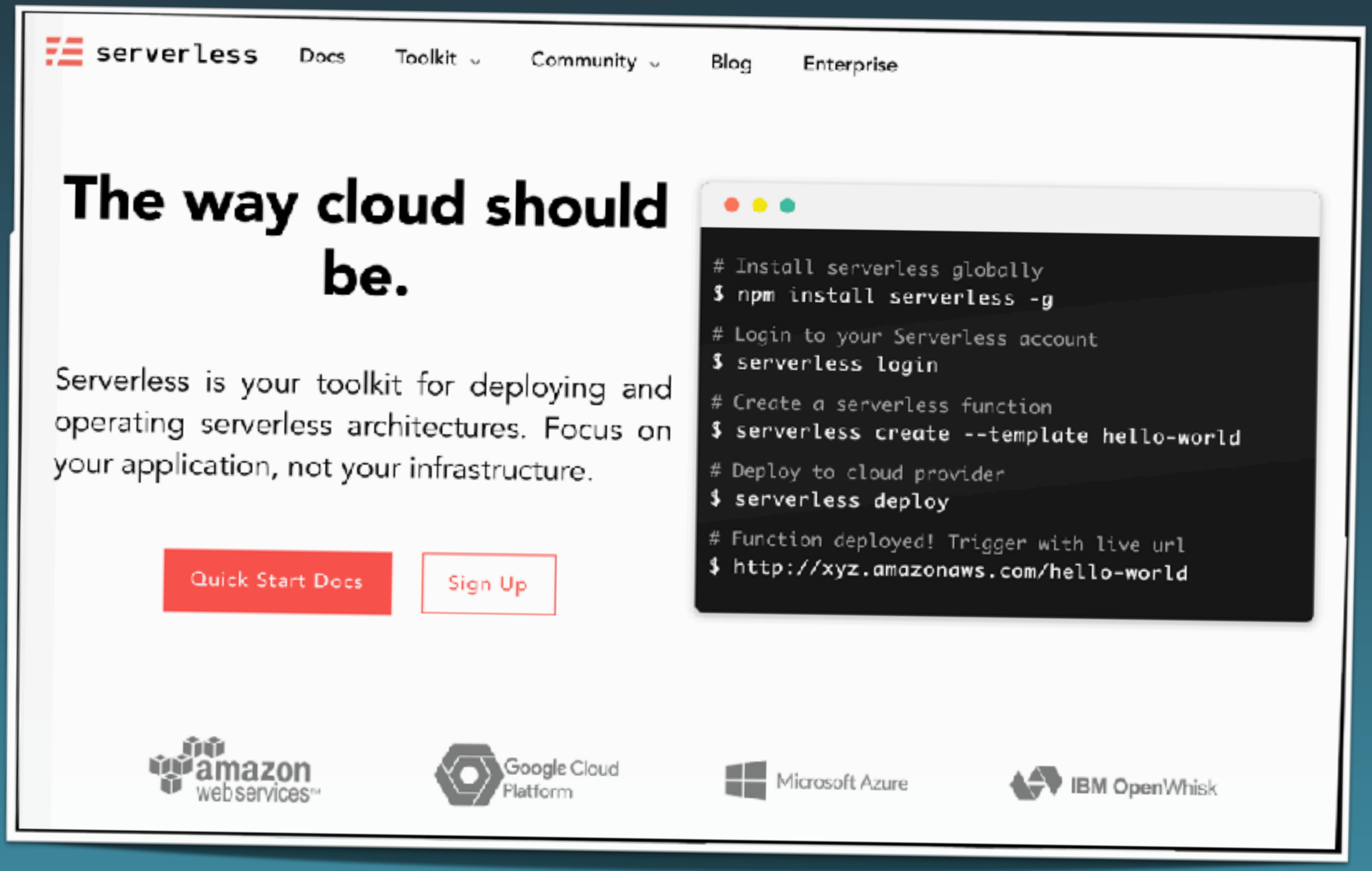
- AWS will autoscale your functions when load increases

    - up to a max amount of 1000 running lambda functions (default)

Mooaaarrrr abstractions pleaz!

Alternatively

- Give serverless.com a spin
- A framework for creating AWS Lambda powered functions with (even) less hassle
- Google, IBM, and Microsoft offerings also supported



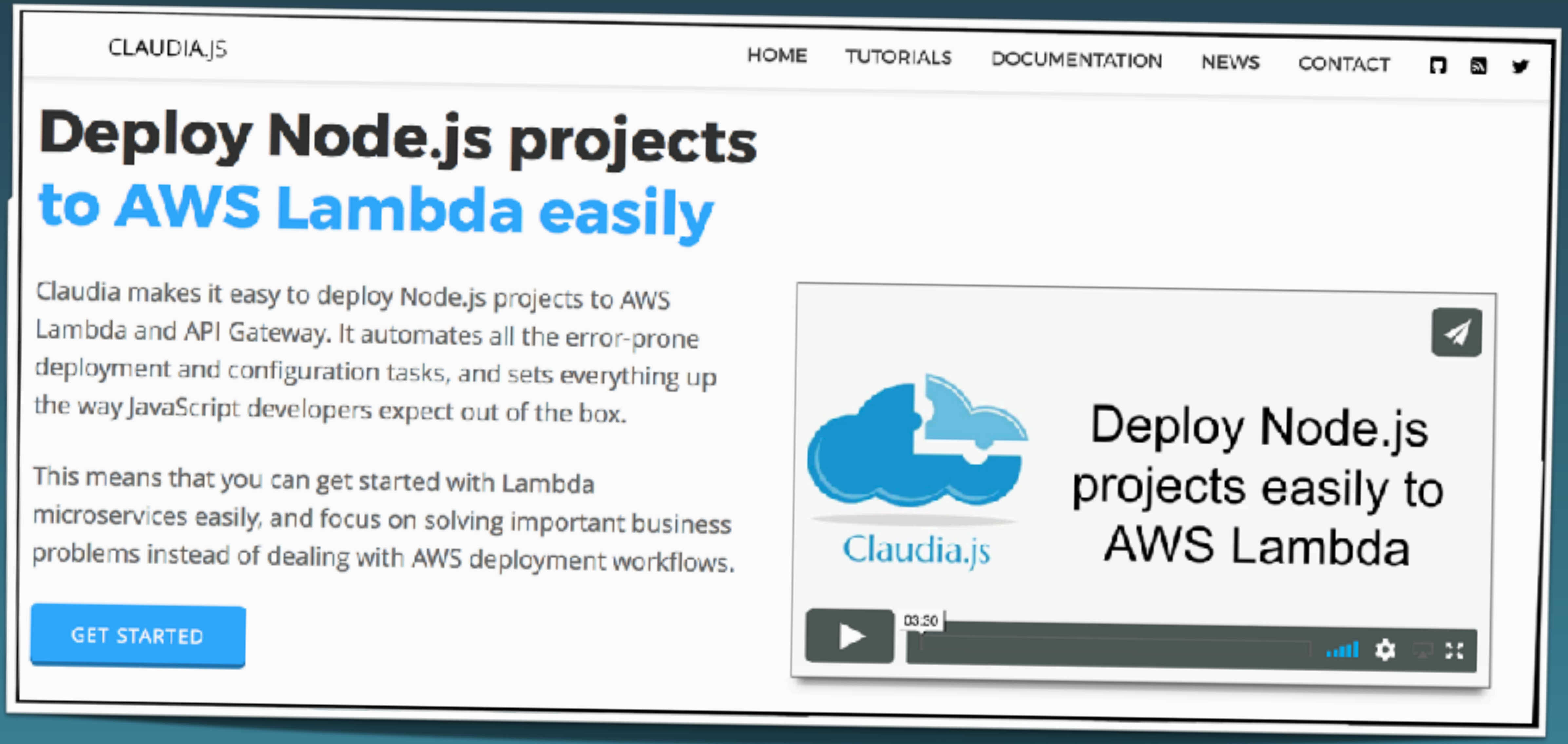
The way cloud should be.

Serverless is your toolkit for deploying and operating serverless architectures. Focus on your application, not your infrastructure.

```
# Install serverless globally
$ npm install serverless -g
# Login to your Serverless account
$ serverless login
# Create a serverless function
$ serverless create --template hello-world
# Deploy to cloud provider
$ serverless deploy
# Function deployed! Trigger with live url
$ http://xyz.amazonaws.com/hello-world
```

And for JavaScript fanboys…

- Try claudiajs.com
- A framework for creating AWS Lambda powered JavaScript microservices the easy way



Deploy Node.js projects to AWS Lambda easily

Claudia makes it easy to deploy Node.js projects to AWS Lambda and API Gateway. It automates all the error-prone deployment and configuration tasks, and sets everything up the way JavaScript developers expect out of the box.

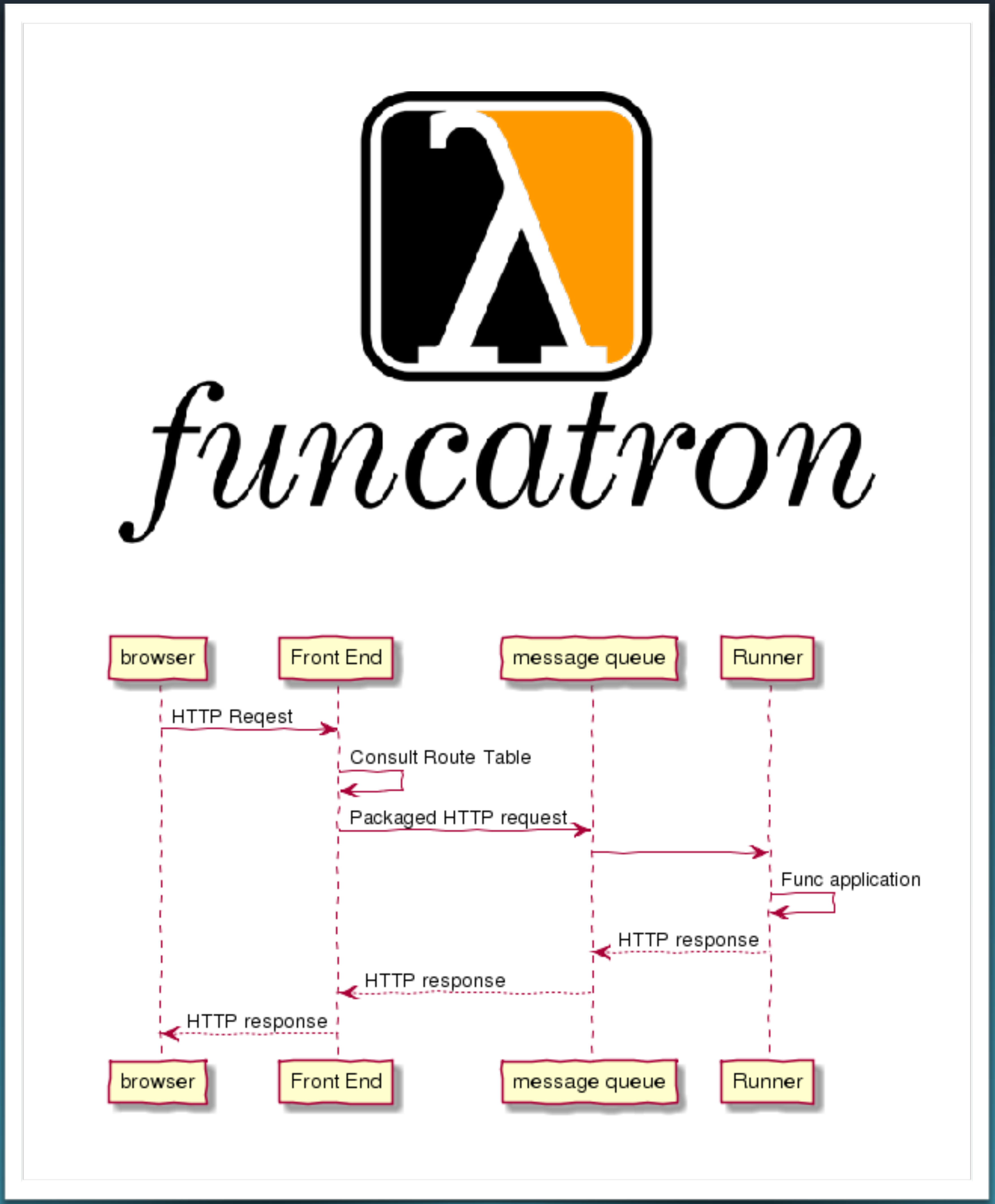
This means that you can get started with Lambda microservices easily, and focus on solving important business problems instead of dealing with AWS deployment workflows.

HOLD YOUR HORSES!

Yeah, nice and all, but all of this $**t is running in the cloud…

Funcatron

- Framework that uses the Lambda-paradigm
- Created by David Pollack (Lift)
- Deploy to Mesos, Kubernetes, Docker Swarm



Apache OpenWhisk

- Runs in IBM’s and Red Hat’s cloud offerings, but can also run on-premise
- OpenWhisk is Apache licensed and Open Source
- Currently supports: JavaScript, Java, Python, and Swift(!)
- Possible to run functions in provided Docker images

fn

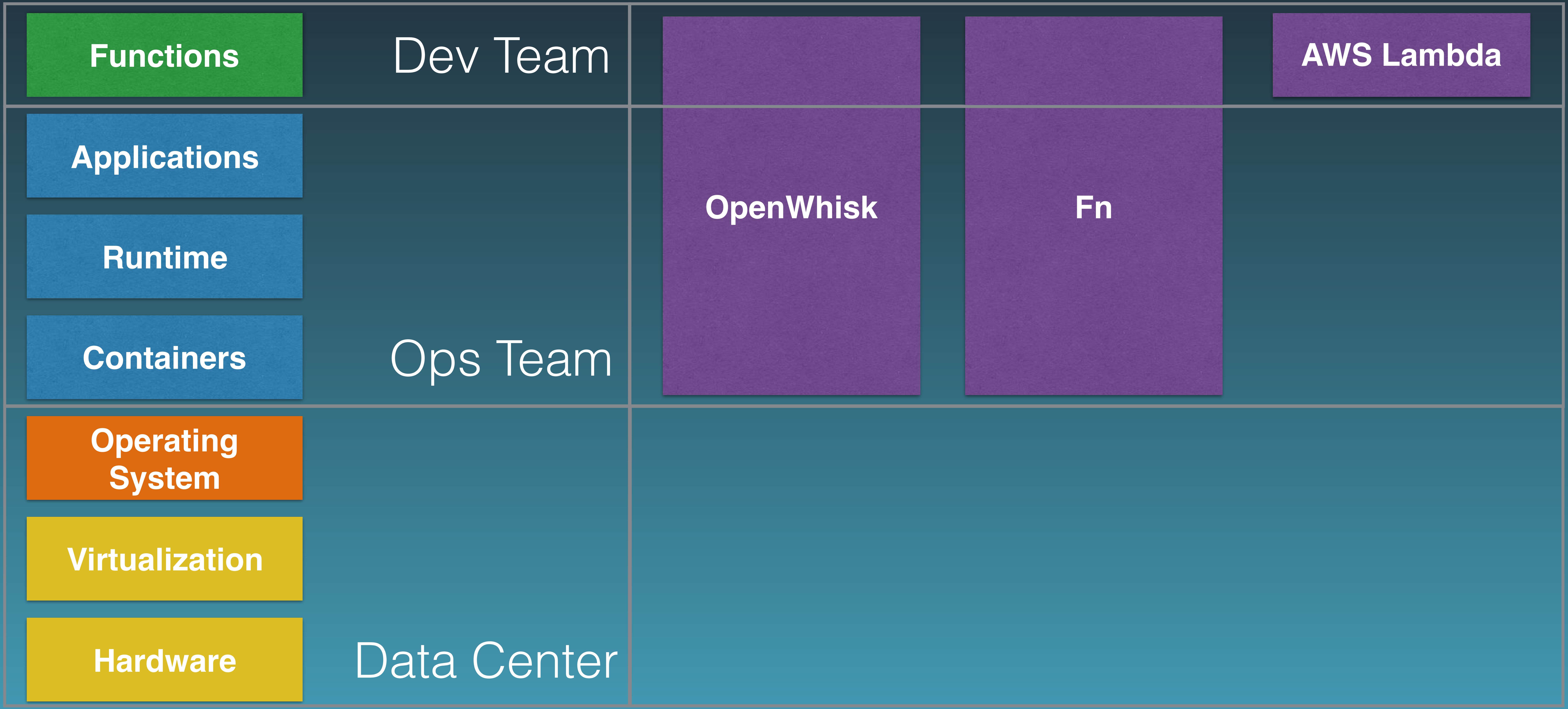
Fn - Why another framework?

- Most serverless FaaS offerings are proprietary, only some are open source
- Many common concepts, but no standards
- Poor development experience - low fidelity between DEV and PROD
- Poor Java support

Is this still Serverless?

- Functions are the unit of deployment and scaling
- No Machines, VMs, or Containers are visible in the programming model
- Permanent storage lives elsewhere
- Scales per request. Users cannot over- or under-provision capacity
- Never pay for idle (no cold servers/containers or their costs)
- Implicitly fault-tolerant because functions can run anywhere
- BYOC - Bring Your Own Code
- Metrics and Logging are a universal right

Is this still Serverless?



demo

Now Serverless is cool, but there are some drawbacks too…

Drawbacks

- Vendor control and lock-in
- Multi-tenancy
- Security concerns (increasing the attack surface)
- Loss of server optimizations
- Execution time is limited
- Start-up latency
- Testing
- Discovery

summary

Serverless

- ..is rapidly being embraced by major cloud players
- ..is promoting functions as first class citizens
- ..is event-based, stateless, and transient
- ..is infinitely scalable (in theory)
- ..is different from traditional deployment models
- ..is giving the cloud a run for its money
- ..is lots of bang for the buck
- ..is still very much proprietary, so lock-in is your choice!

Serverless, the future of the Cloud!

Thank you!

@BertErtman