Serverless 101

Welcome to the world of functions & BaaS

Serverless?

No servers?

Zero infrastructure?

Not having to think about servers!

The evolution of IT infrastructure

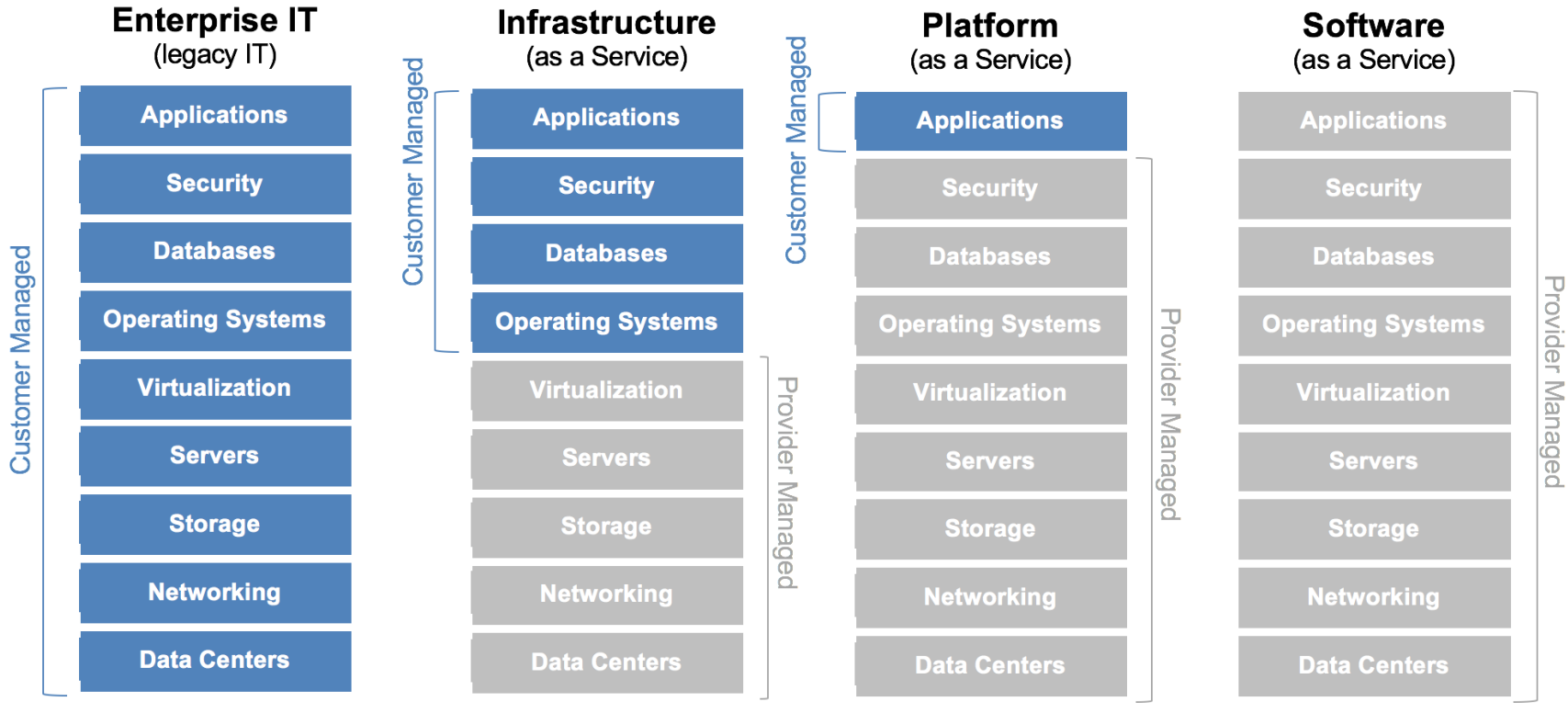
In the beginning there were physical servers...

Problems of infrastructure handling in the dark age

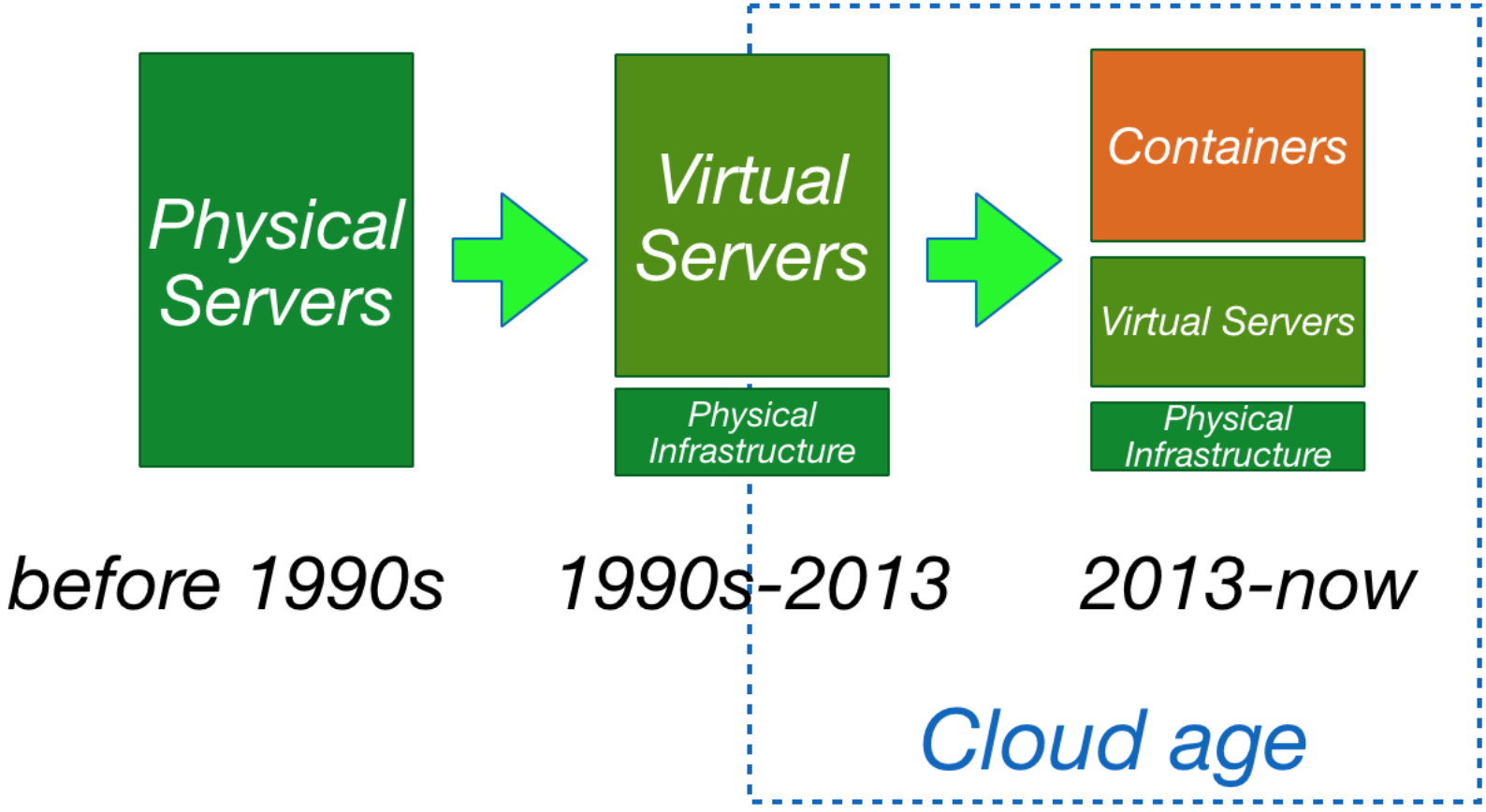
- Compute resources acquirement was hard and tedious
- Inflexible compute resources cannot meet the demand of ever-changing business needs
- Inflexible compute resources hinder value creation and innovation

Let there be cloud

Traditional Cloud Models

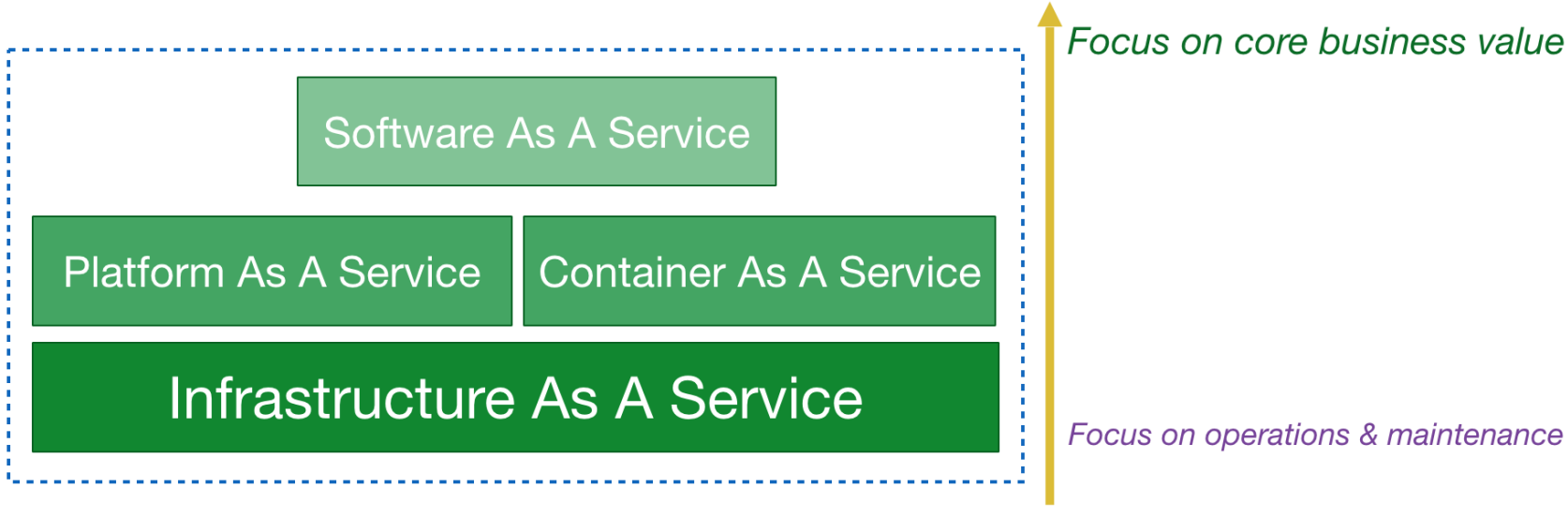


Evolution of IT infrastructure



Benefits of cloud

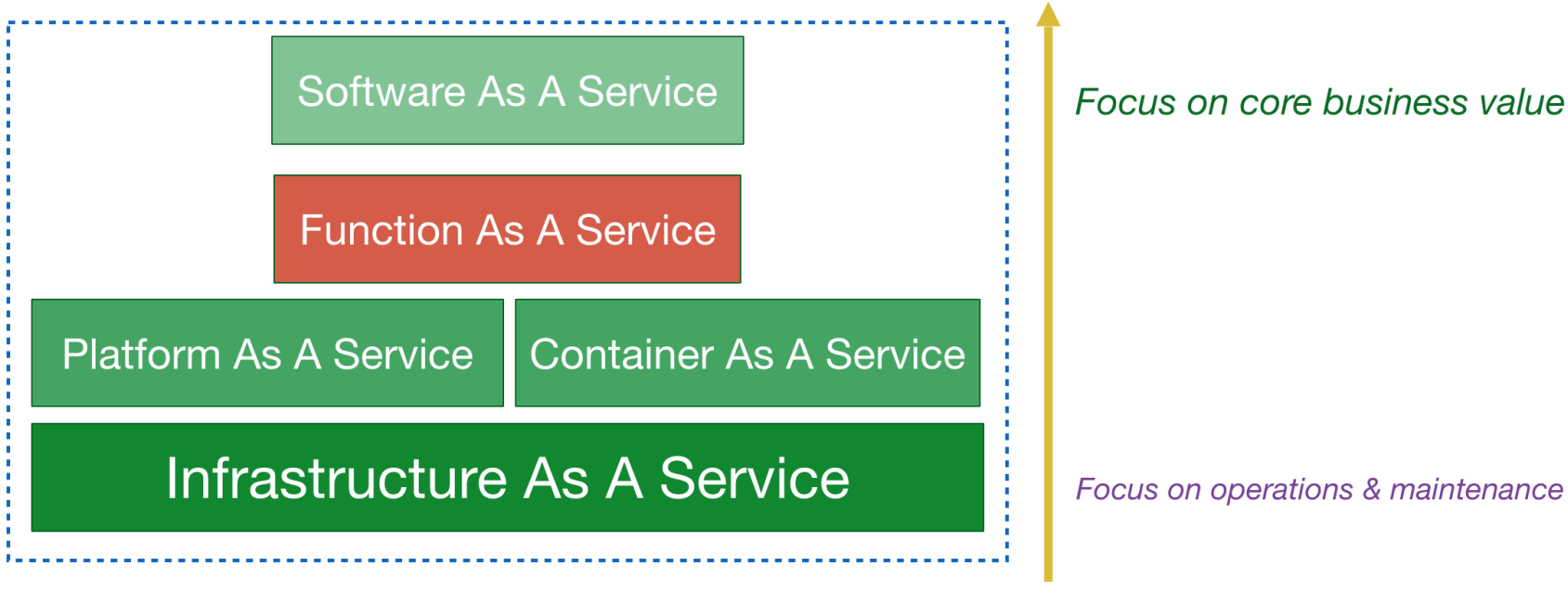
- ***Convenient and timely*** access to compute resource empowers faster value delivery and innovation
- ***Flexible and scalable*** compute resources embraces rapid growth in business needs
- ***Better utilized*** compute resource usage reduces cost


Software As A Service
Platform As A Service
Container As A Service
Infrastructure As A Service
Focus on core business value
Focus on operations & maintenance

Beyond traditional cloud...

- If ***better utilization of compute resources*** reduces cost, can we push utilization to the extreme?
- If ***better scalability meets rapid-changing business needs***, can we make software ***scalable by default***?
- If ***more of delivering business value and less of system administration*** are desirable, can we further reduce the burden of system administration operations?

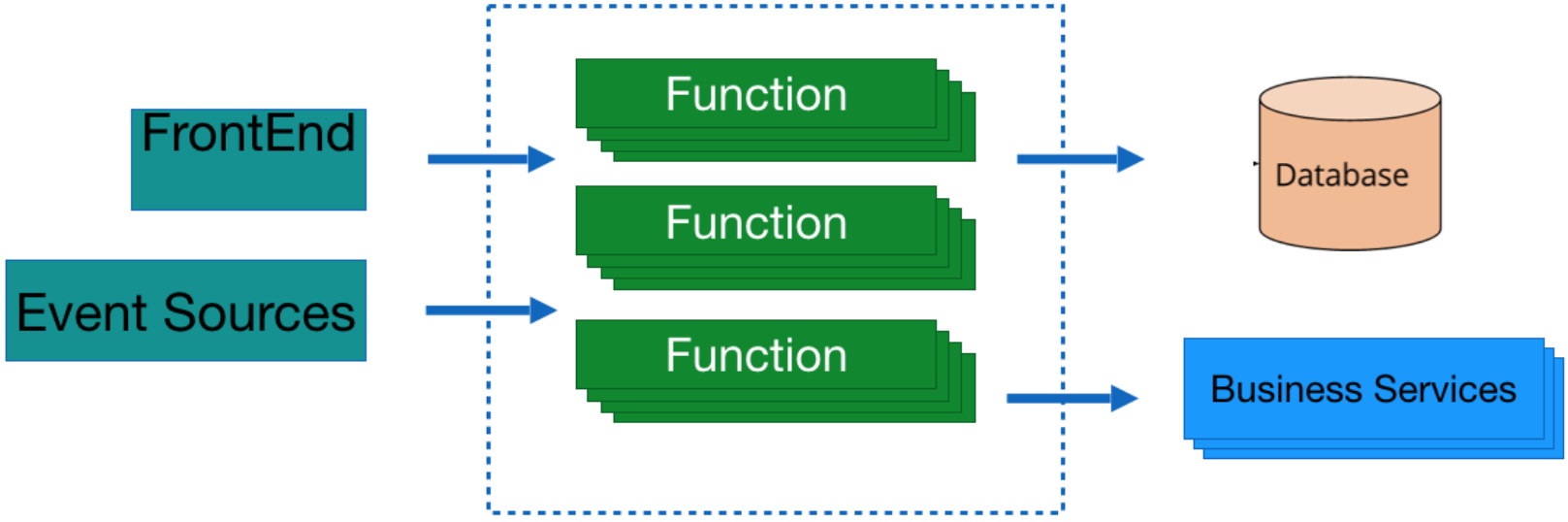
Here comes FaaS & Serverless



But serverless is more than FaaS

Continue on..

First glance at Serverless



1. Functions are triggered by upstream events (message from message queue, user clicking a button, etc.).
   Both push and pull model can be used for fetching events.
2. Functions run in stateless transient containers.
   Containers are created on demand and destroyed as soon as function execution finishes.
3. Functions are fully managed by cloud service provider.
   Usual operation concerns on monitoring, logging, security patching, etc. are also offloaded to 3rd party cloud providers.

What about the database and these "Business services"?

- Functions act as ***"glues" to different business services***
- Business services include authentication services (Auth0, AWS Cognito), Monitoring (CloudWatch), Data source & analytic service (AWS Kinesis), etc.
  These are sometimes referred to as ***Backend as a Service (BaaS)***.
- Database => cloud accessible ***Database as a Service (DBaaS)***, e.g. AWS Dynamodb, Firebase

Serverless == No servers?

Serverless == No server administration!

Stateless transient functions & these ba run on servers fully managed by cloud platform provider. Delivery teams focus on development and deployment, thus zero server maintenance.

In the dark age servers are treated as pets

In the cloud age servers are treated as cattle

In the serverless world

Serverless providers

	AWS Lambda
	Azure Functions
	Google Functions
	Function Compute
	OpenWhisk

Serverless Architecture & Benefits

*AWS is used as example, but most of the following apply to other cloud vendors

Serverless Benefits: only pay for compute time

Typical application:
CDE-LOGGER
used to process MQ
messages specific to
certain customers

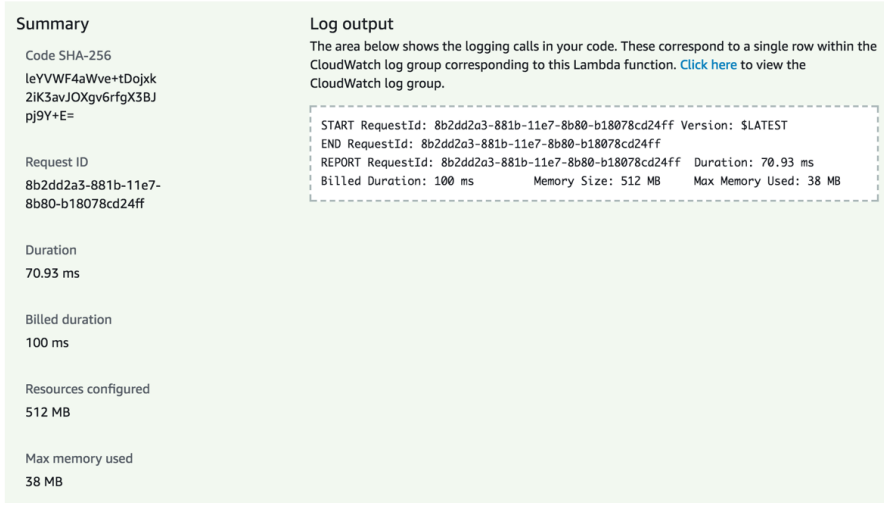
Average requests:
~20k per day
Time taken to process
each request:
~500ms

Single EC2 instance
(which means SPoF)
t2.Medium
0.047$/hr

AWS Lambda

Monthly Spending	0.047 * 24 * 30 = 33.84$	20000 * 30 * 500/100 * 0.000000208 = 0.624$ (Actually within free tier)

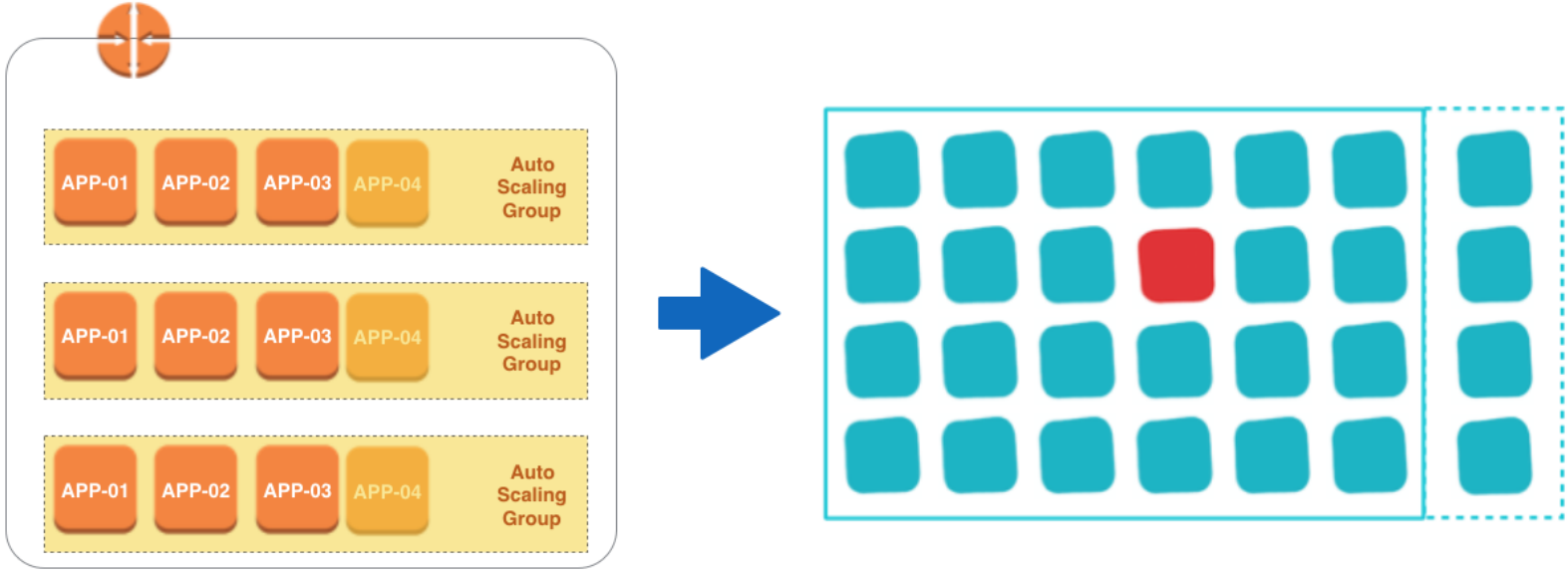
Serverless Benefits: easier operations & maintenance



Time (UTC +00:00)	Message
2017-08-23	
15:55:25	END RequestId: 7927def6-881b-11e7-9851-81f93c8cd066
15:55:25	REPORT RequestId: 7927def6-881b-11e7-9851-81f93c8cd066 Duration: 234.71 ms Billed Duration: 300 ms Memory Size: 512 MB Max Memory Used: 37 MB
15:55:28	START RequestId: 7bd6f910-881b-11e7-9557-335b1ec2b494 Version: $LATEST
15:55:28	END RequestId: 7bd6f910-881b-11e7-9557-335b1ec2b494
15:55:28	REPORT RequestId: 7bd6f910-881b-11e7-9557-335b1ec2b494 Duration: 46.00 ms Billed Duration: 100 ms Memory Size: 512 MB Max Memory Used: 37 MB
15:55:29	START RequestId: 7cdd6d69-881b-11e7-aa94-ebbe68200133 Version: $LATEST
15:55:29	END RequestId: 7cdd6d69-881b-11e7-aa94-ebbe68200133
15:55:29	REPORT RequestId: 7cdd6d69-881b-11e7-aa94-ebbe68200133 Duration: 58.89 ms Billed Duration: 100 ms Memory Size: 512 MB Max Memory Used: 38 MB
15:55:34	START RequestId: 7f632d2a-881b-11e7-bfc7-531f8199a347 Version: $LATEST
15:55:34	END RequestId: 7f632d2a-881b-11e7-bfc7-531f8199a347
15:55:34	REPORT RequestId: 7f632d2a-881b-11e7-bfc7-531f8199a347 Duration: 44.27 ms Billed Duration: 100 ms Memory Size: 512 MB Max Memory Used: 38 MB
15:55:39	START RequestId: 82b9f98a-881b-11e7-ac1d-7d1643519fb8 Version: $LATEST
15:55:39	END RequestId: 82b9f98a-881b-11e7-ac1d-7d1643519fb8
15:55:39	REPORT RequestId: 82b9f98a-881b-11e7-ac1d-7d1643519fb8 Duration: 38.63 ms Billed Duration: 100 ms Memory Size: 512 MB Max Memory Used: 38 MB
15:55:41	START RequestId: 83ef1df2-881b-11e7-afcc-d3f3aa56ab8f Version: $LATEST
15:55:41	END RequestId: 83ef1df2-881b-11e7-afcc-d3f3aa56ab8f
15:55:41	REPORT RequestId: 83ef1df2-881b-11e7-afcc-d3f3aa56ab8f Duration: 44.40 ms Billed Duration: 100 ms Memory Size: 512 MB Max Memory Used: 38 MB
15:55:47	START RequestId: 87549081-881b-11e7-8b80-1f181af27408 Version: $LATEST
15:55:47	END RequestId: 87549081-881b-11e7-8b80-1f181af27408

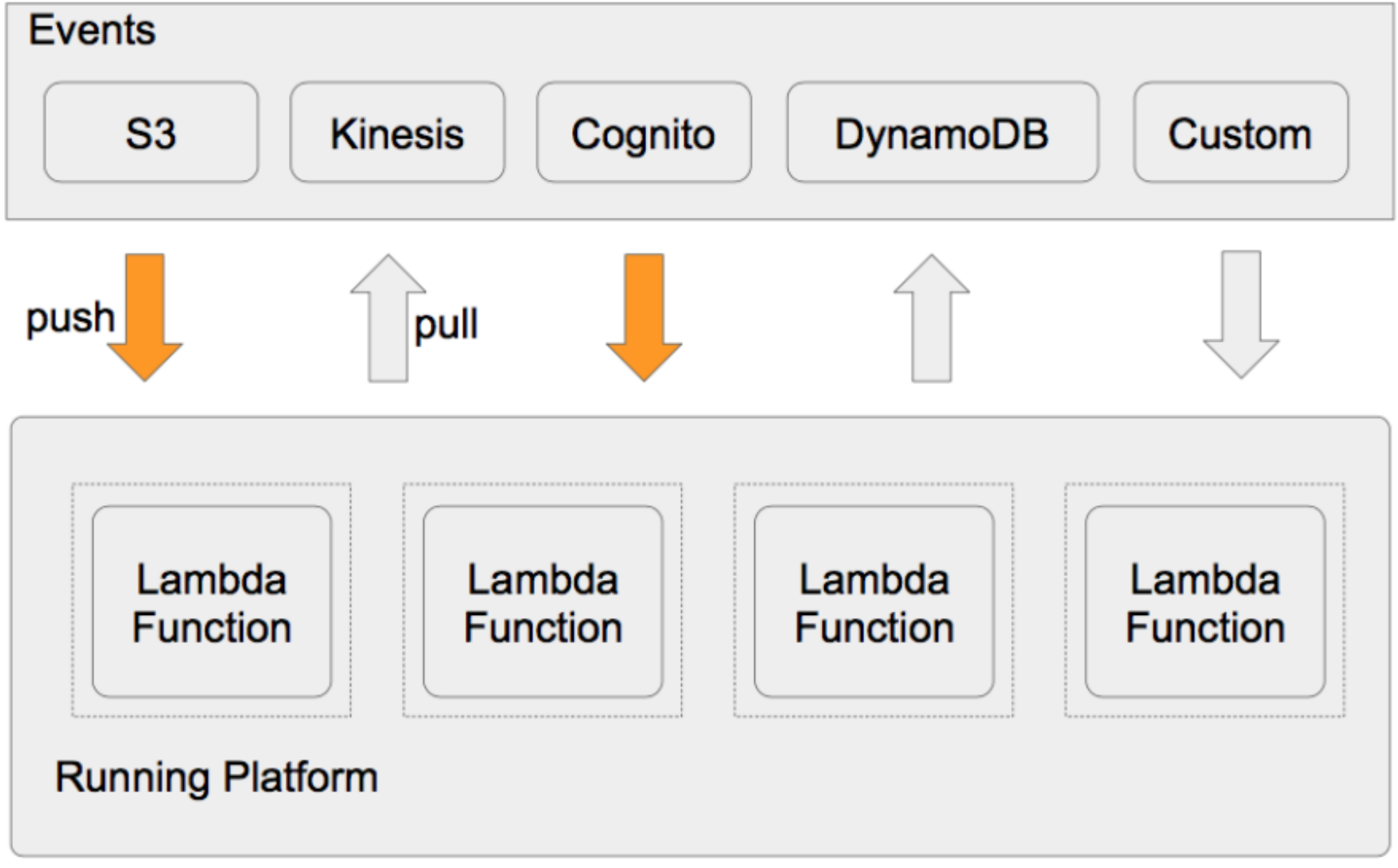
Cloud providers like AWS provided integrated services like monitoring, logging management, auto-scaling, exceptions management, etc.

Serverless Benefits: scalability by default



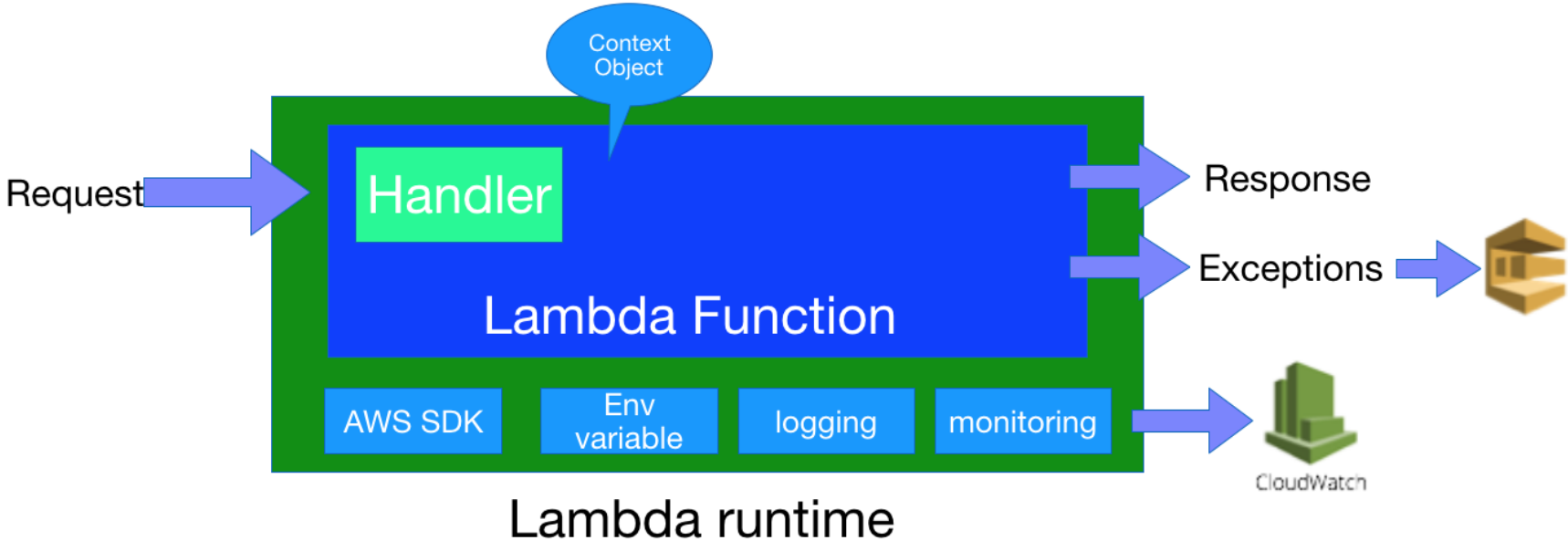
concurrent executions = events (or requests) per second * function duration

Serverless Benefits: seamless integration with other cloud services



Lambda functions need to be bound to event sources (either custom event or AWS native services)

AWS Lambda Programming model



- AWS SDK included in function runtime
- Environment variables can be dynamically injected at runtime

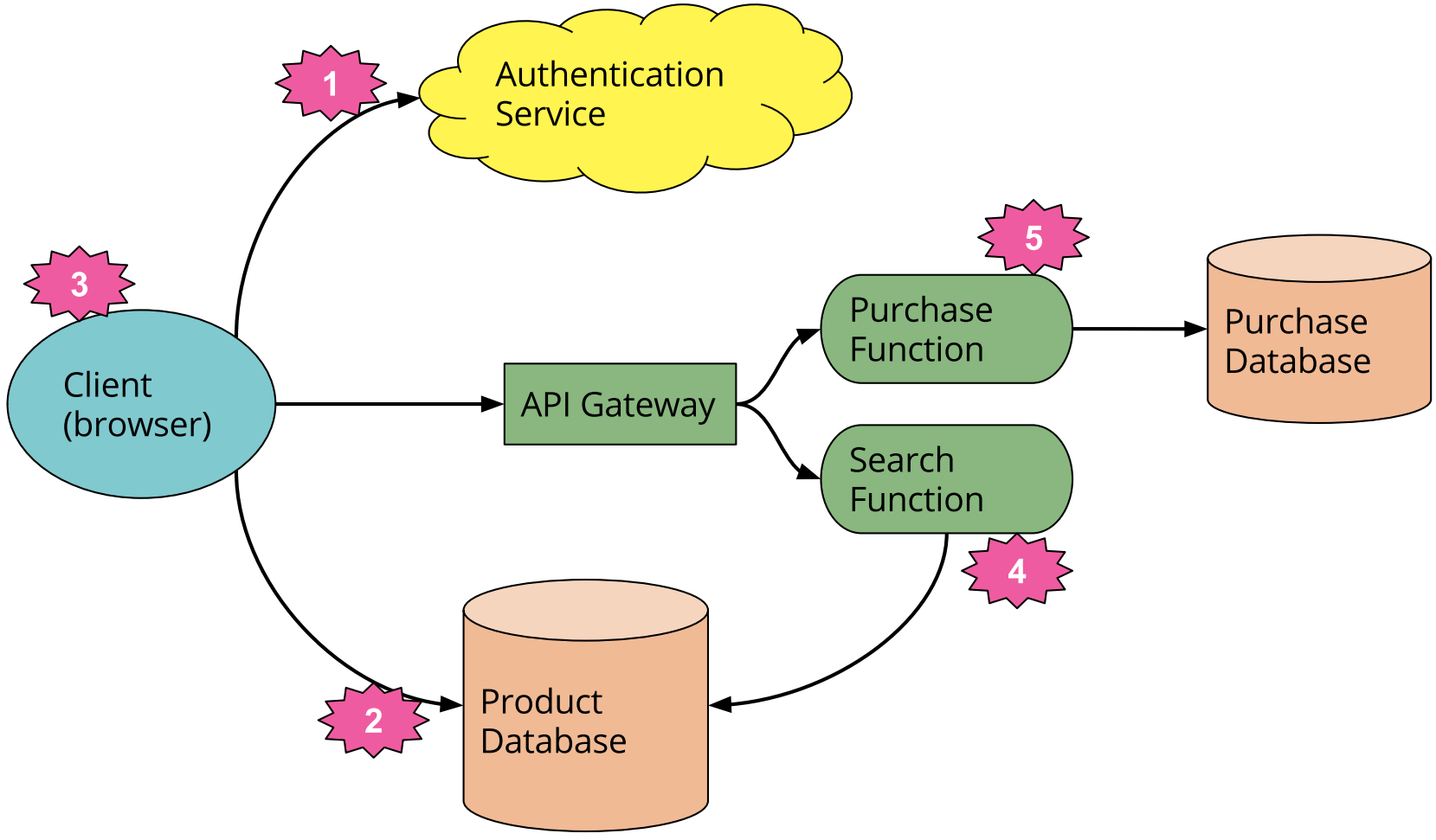
Serverless architecture typical examples

*Examples shamelessly taken from Serverless Architectures by Mike Roberts

UI Driven application: traditional architecture

- traditional 3-tier client-oriented system with server-side logic
- logic-thin client, most logic including authentication, page navigation, searching, transactions implemented by the server application

UI Driven application: serverless architecture



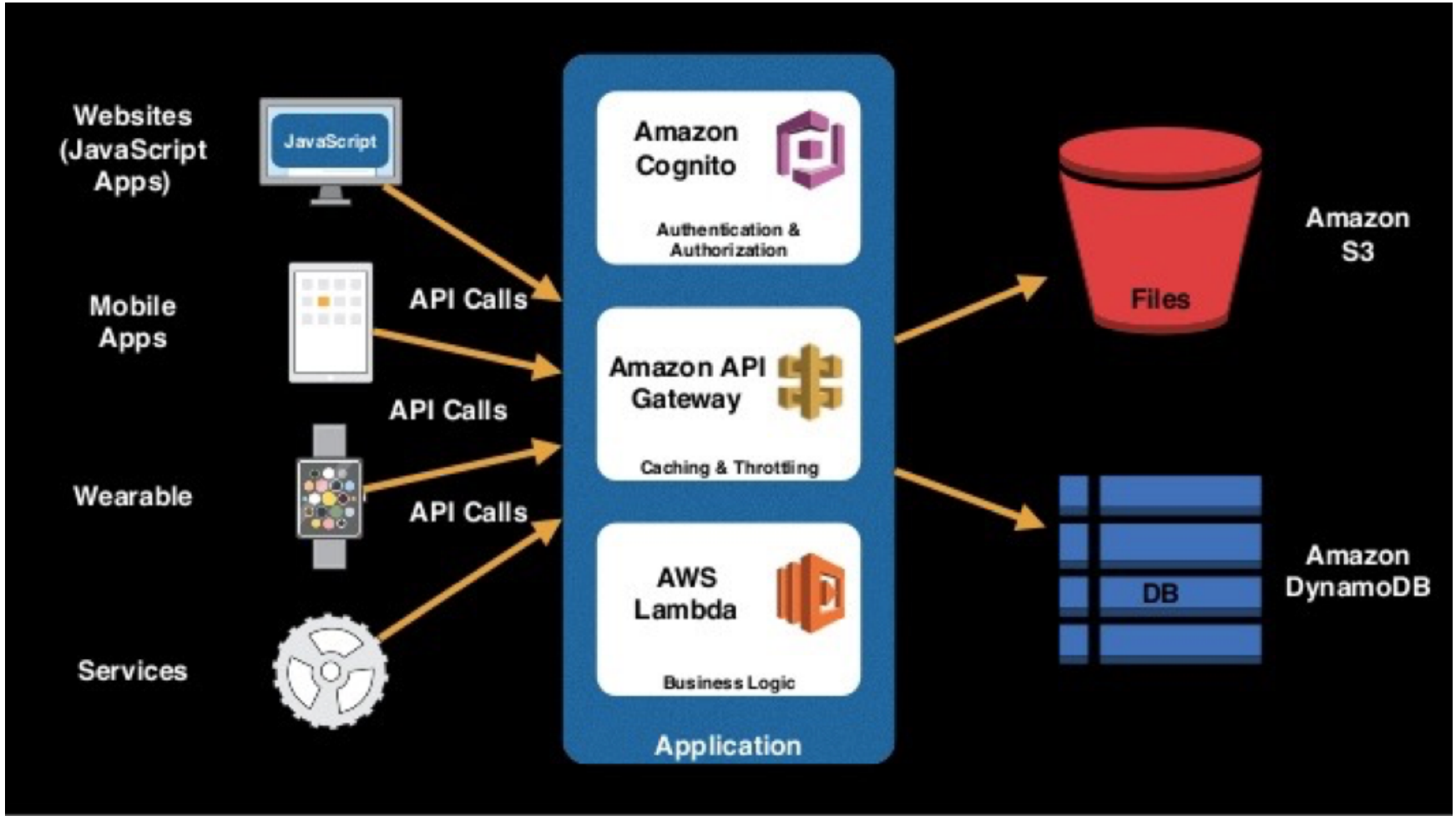
Differences between traditional and serverless architecture

- custom developed authentication logic => replaced with 3rd party BaaS service
- DBaaS (e.g. AWS DynamoDB) allowed client direct access to subset of database
- client becomes business logic rich: keeping track of a user session, understanding the UX structure of the application (e.g. page navigation), reading from a database and translating that into a usable view, etc.
- Product search feature still reside within server-side, with a FaaS function responding to HTTP requests via API Gateway. Both client and server function read from the same product database.

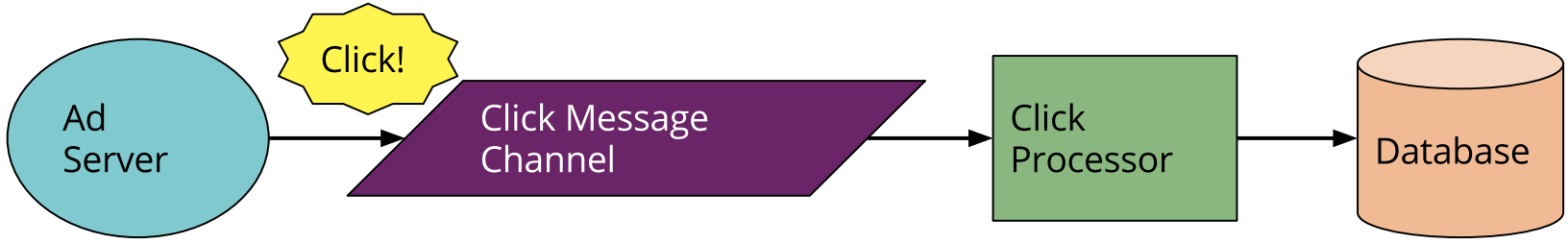
API Gateway

Resource	+	HTTP Verb	→	Method
/books	+	GET	→	GetAllBooks
/books	+	POST	→	CreateNewBook
/books/{id}	+	GET	→	GetBookById
/books/{id}	+	PUT	→	CreateOrUpdateBookById
/books/{id}	+	DELETE	→	DeleteBookById

- fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
- traffic management, authorization and access control, monitoring, and API version management

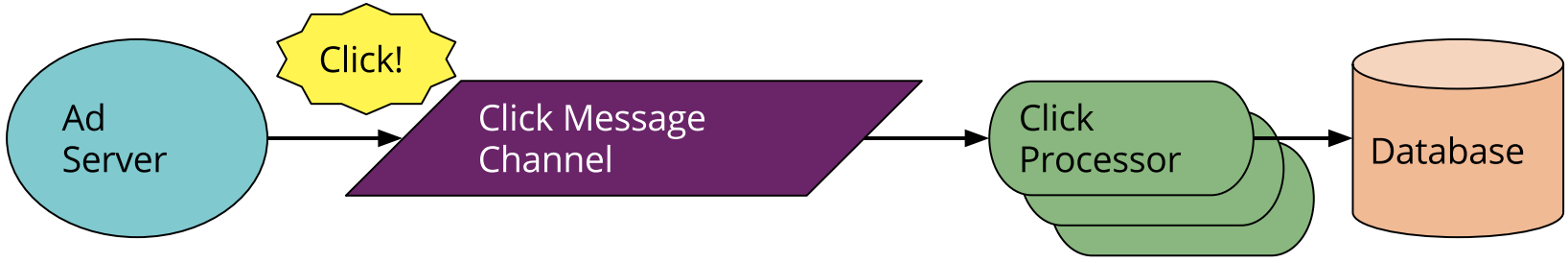

Websites (JavaScript Apps)
JavaScript
Mobile Apps
API Calls
API Calls
Wearable
API Calls
Services
Amazon Cognito
Authentication & Authorization
Amazon API Gateway
Caching & Throttling
AWS Lambda
Business Logic
Application
Files
Amazon S3
DB
Amazon DynamoDB

Message Driven application: traditional architecture



- backend data-processing service
- user clicks on an ad redirect them to the target of the ad
- also posts a message to a channel that can be asynchronously processed by a â€˜click processorâ€™ application that updates a database, to charge the adertiser

Message Driven application: serverless architecture



- much smaller difference
- long lived consumer application => replaced with a FaaS function
- vendor supplies the message broker (e.g. AWS SQS) and FaaS environment(e.g. AWS Lambda), closely integrated with each other

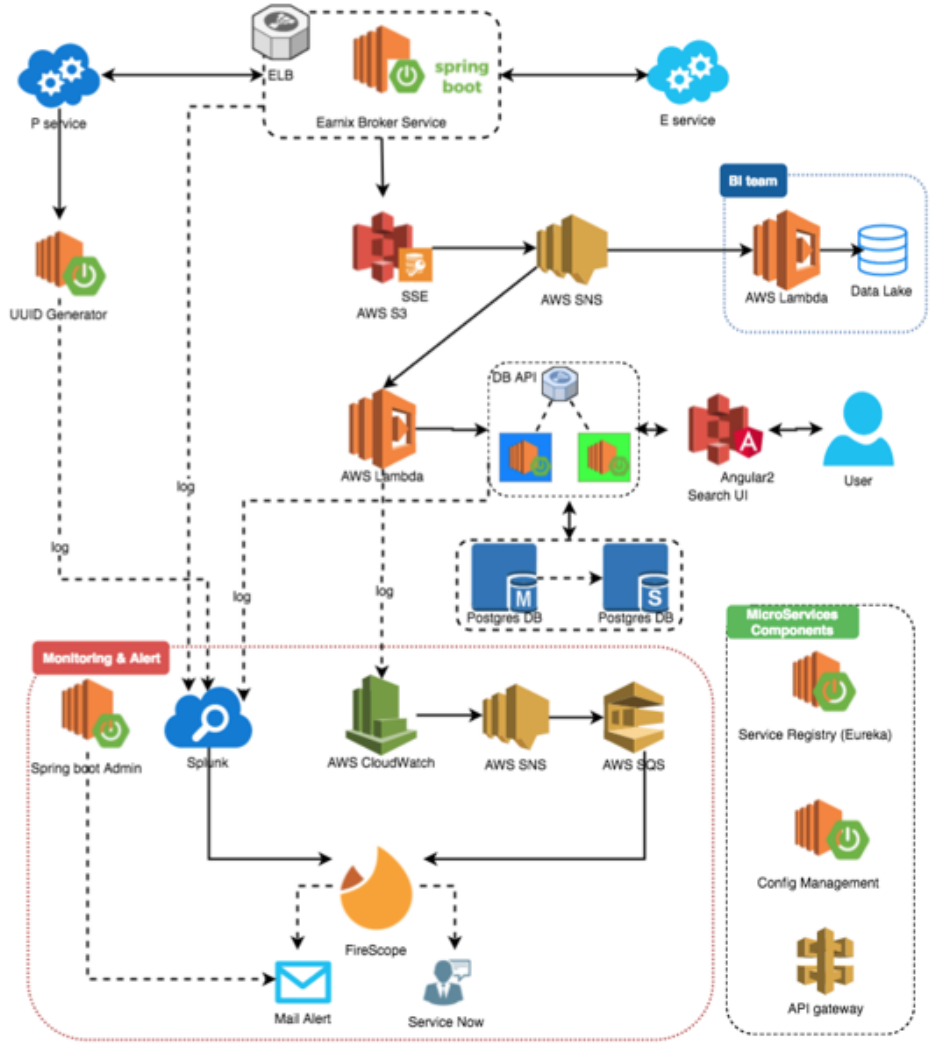
Mixed architectures in reality

Charactistics:

- Lambda co-exist with microservices
- Smaller service units

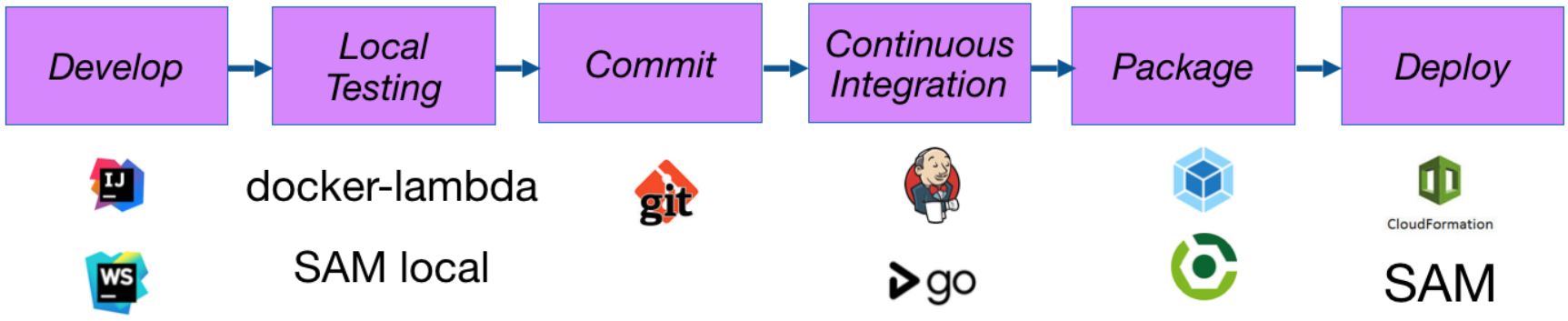
Gains:

- EC2 instances cost reduced 30%
- Less Infrastructure config
- More flexible tech stack choices



Continuous Delivery with serverless

CD pipeline with serverless



Local Development & Testing

- Focus on event interface design of FaaS functions
- Easy unit testing & TDD
- For local integration testing, use SAM local or docker-lambda to simulate FaaS environment
- Use SAM Local to simulate API Gateway and generate event payload

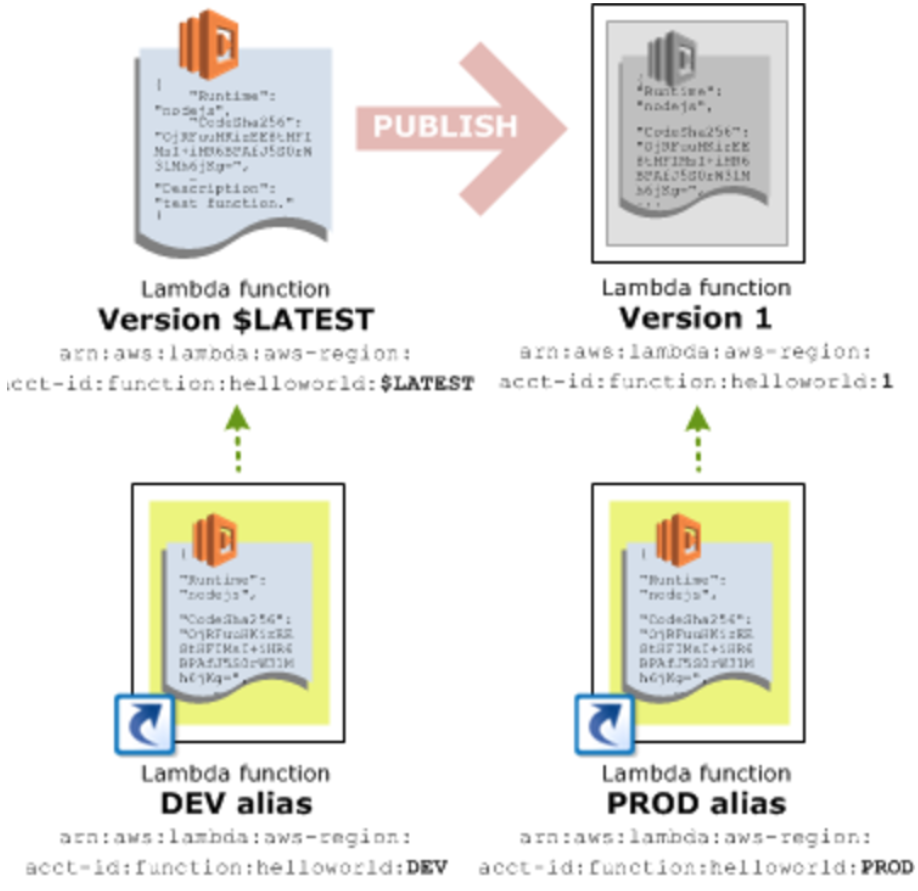
Deployment

- SAM to describe FaaS functions
- Cloudformation to deploy to AWS S3

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  MyFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: nodejs6.10
      CodeUri: 's3://my-bucket/function.zip'
```

aws cloudformation deploy -- template-file serverless-output.yaml --stack-name lambdaExample --capabilities CAPABILITY_IAM

Environment management



- Versioning: Each function uploaded has unique version number (ARN) and it's immutable.
- Alias: Function can have alias, source can use alias to call functions

Serverless limitations & future

Serverless limitations

Resource	Limitation
Memory	128M-1536M
Temporary Storage	512M
Max execution time for each request	300 sec
request/response body size	6MB
Deploy package size	50MB
Unzipped package size	250MB

AWS Supported runtime environments

- Node.js 4.3.2 and 6.10.3
- Java 8
- Python 3.6 and 2.7
- .NET Core (C#) 1.0.1

Scripting languages runtime startup time: several hundred milliseconds.
Java startup time can take as much as 10 secs.

Vendor dependent & Vendor lock

- FaaS environment
- Configurable event sources and services integration

Ecosystem still in its infancy

SERVERLESS

Docker Lambda

Apex

AWS CLI

Conclusion

- Serverless is superb in event based scenarios, with short development cycles and easy integration with event sources, zero server configuration result in improved delivery efficiency.
- Limitations apply. Single FaaS function cannot be too complex and hardly suitable for every scenario.
- Future is likely to be mixed solution of serverless & container-based systems.

Thanks!

Any questions/issues please let me know at:

https://github.com/richardzone/ansible-training-workshop/issues

Presented with ♥ by Richard Liu