Getting Started with Serverless Architectures

Daniel Geske, Solutions Architect, AWS

29. März 2017

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

amazon web services

Agenda

Background

AWS Lambda

Amazon API Gateway

Serverless Architecture Patterns

Serverless Best Practices

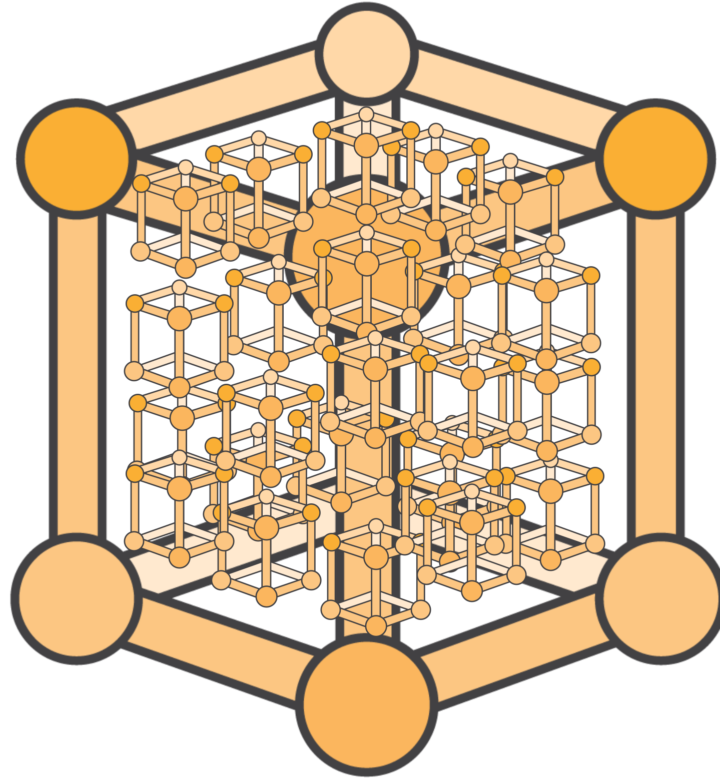
amazon web services

Background

How serverless architecture patterns with AWS Lambda are the next evolution of application design

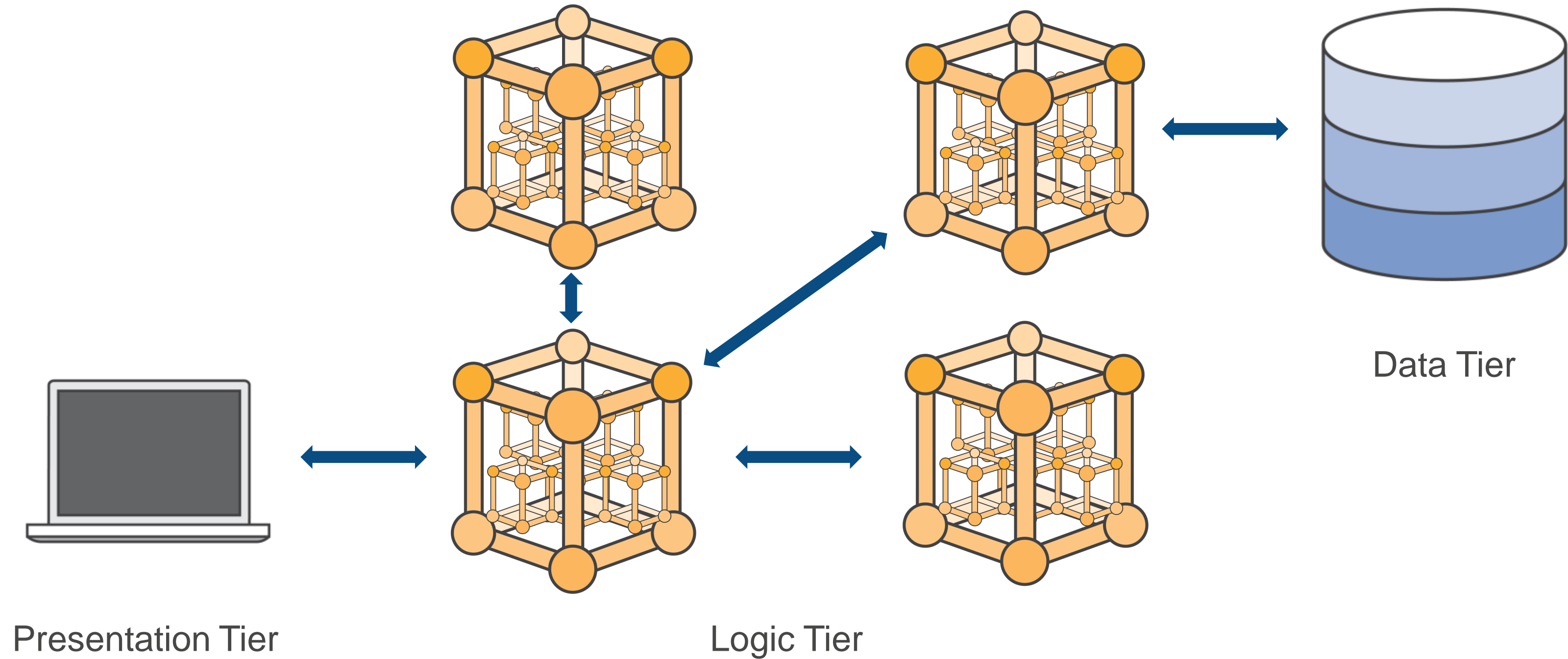
amazon web services

The Monolithic Architecture



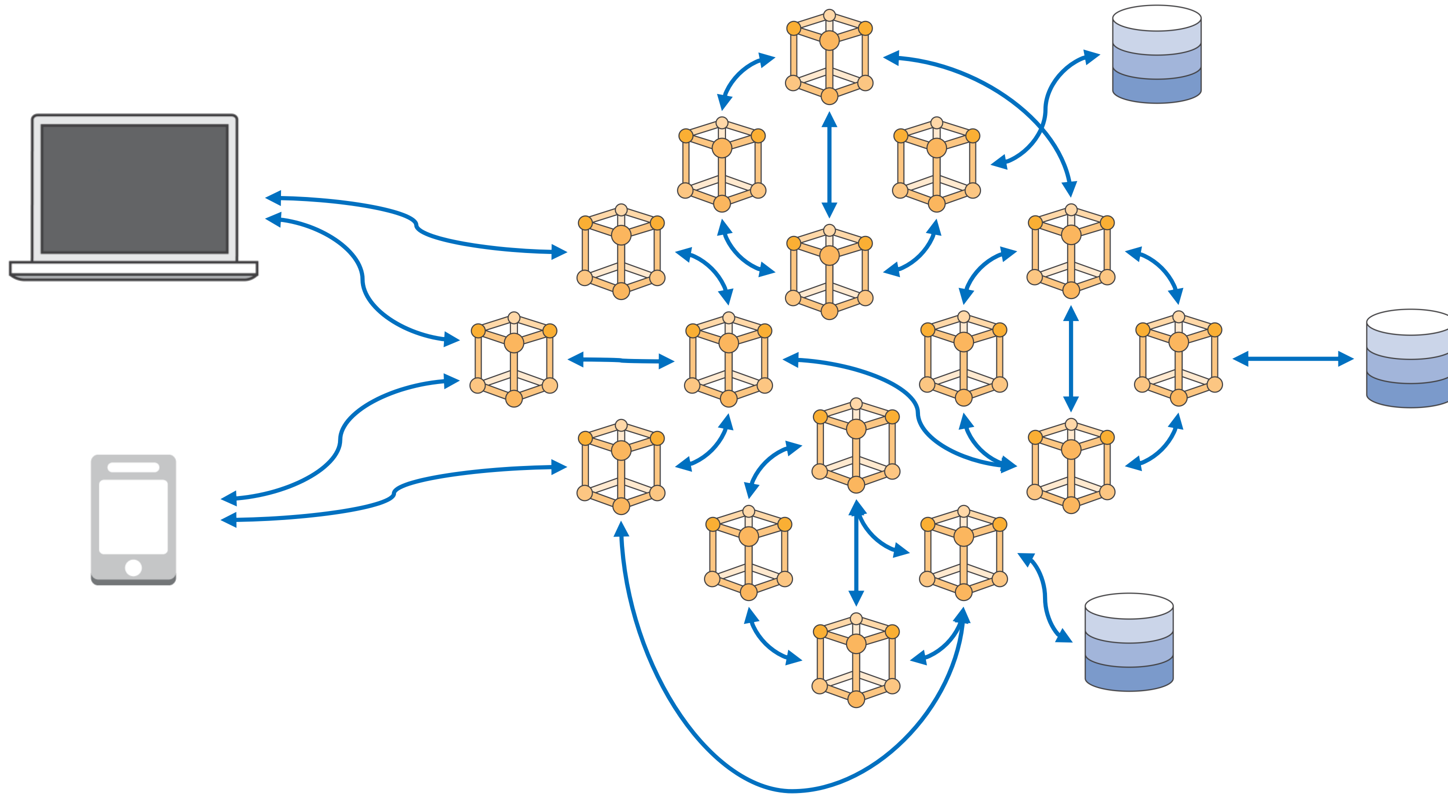
amazon
web services

The Service-Oriented Architecture



amazon web services

The Microservices Architecture



amazon web services

Tools to Help This Pattern are VAST

Web servers
Code libraries
Web service/application frameworks
Configuration management tools
API management platforms
Deployment patterns
CI/CD patterns
Containers
Etc. Etc. Etc.

amazon web services

AWS Has Helped Too!

Amazon EC2
Auto Scaling
Elastic Load Balancing
Amazon EC2 auto recovery
AWS Trusted Advisor
AWS Elastic Beanstalk
AWS OpsWorks
Amazon ECS
Etc. Etc. Etc.

But….
many of these tools and innovations are still coupled to a shared dependency…

amazon web services

Servers (AAHHHHHHHHH!!)

What size servers are right for my budget?

How many users create **too much load** for my servers?

How much remaining **capacity** do my servers have?

How can I **detect** if a server has been **compromised**?

How many servers should I **budget** for?

Which **OS** should my servers run?

Which **users** should have **access to** my servers?

How can I **control** **access from** my servers?

How will I keep my server **OS patched**?

How will new code be **deployed** to my servers?

How can I increase **utilization** of my servers?

When should I decide to **scale out** my servers?

What **size** server is right for my **performance**?

Should I **tune OS settings** to **optimize** my application?

Which **packages** should be **baked** into my **server images**?

When should I decide to **scale up** my servers?

How should I handle server **configuration changes**?

How will the **application** handle server **hardware failure**?

amazon web services

Architect to be Serverless

Fully managed
- No provisioning
- Zero administration
- High availability

Developer productivity
- Focus on the code that matters
- Innovate rapidly
- Reduce time to market

Continuous scaling
- Automatically
- Scale up and scale down

amazon web services

AWS Lambda

amazon
web services

Serverless, event-driven compute service

Lambda = microservice without servers

amazon web services

Components of Lambda

- A Lambda function (that you write)
- An event source
- The AWS Lambda service
- The function networking environment

amazon web services

Lambda Function

- Your code
  (Java, NodeJS, Python)
- The IAM role that code assumes during execution
- The amount of memory allocated to your code (affects CPU and network as well)

A valid, complete Lambda function

```
var AWS = require('aws-sdk');
var s3 = new AWS.S3();

exports.handler = function(event, context) {
    var params = {
                    Bucket: '[input bucket name here]',
                    Key: "[insert keyname here]",
                    Body: "[object body]"
                };
    s3.putObject(params, function(err, data) {
        if (err) {
            console.log(err, err.stack); // an error occurred
        }
        else {
            console.log(data);
        }
        context.done();
    } );
};
```

amazon web services

Event Sources

- When should your function execute?
- Many AWS services can be an event source today:

- Amazon S3
- Amazon Kinesis
- Amazon SNS
- Amazon DynamoDB
- Amazon CloudWatch
- AWS Config Rules
- Amazon Echo
- Etc.
- …and Amazon API Gateway (more later)

amazon web services

AWS Lambda

- Runs your function code without you managing or scaling servers.
- Provides an API to trigger the execution of your function.
- Ensures function is executed when triggered, in parallel, regardless of scale.
- Provides additional capabilities for your function (logging, monitoring).

amazon web services

Function Networking Environment

Default - a default network environment within VPC is provided for you

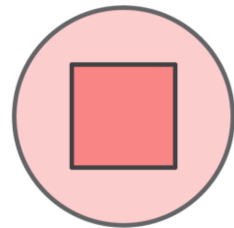
- Access to the Internet always permitted to your function
- No access to VPC-deployed assets

Customer VPC - Your function executes within the context of your own VPC.

- Privately communicate with other resources within your VPC.
- Familiar configuration and behavior with:
  - Subnets
  - Elastic network interfaces (ENIs)
  - EC2 security groups
  - VPC route tables
  - NAT gateway

amazon web services

“Hold on…” – you (maybe)



Lots of Existing Ways to Abstract Away Servers

SaaS

PaaS

MBaaS

*aaS

Application Engines/Platforms

amazon web services

What's Unique About Lambda?

Abstraction at the code/function level (arbitrary, flexible, familiar)

The security model (IAM, VPC)

The pricing model

The community

Integration with the AWS service ecosystem!

- Scale
- Triggers

amazon web services

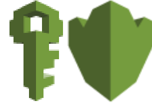
Many Serverless Options on AWS

Storage

Network

Database

Compute

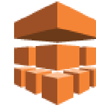

Security

Messaging and Queues

Content Delivery

User Management

Monitoring & Logging


Machine Learning

Gateways

Streaming Analytics

Internet of Things

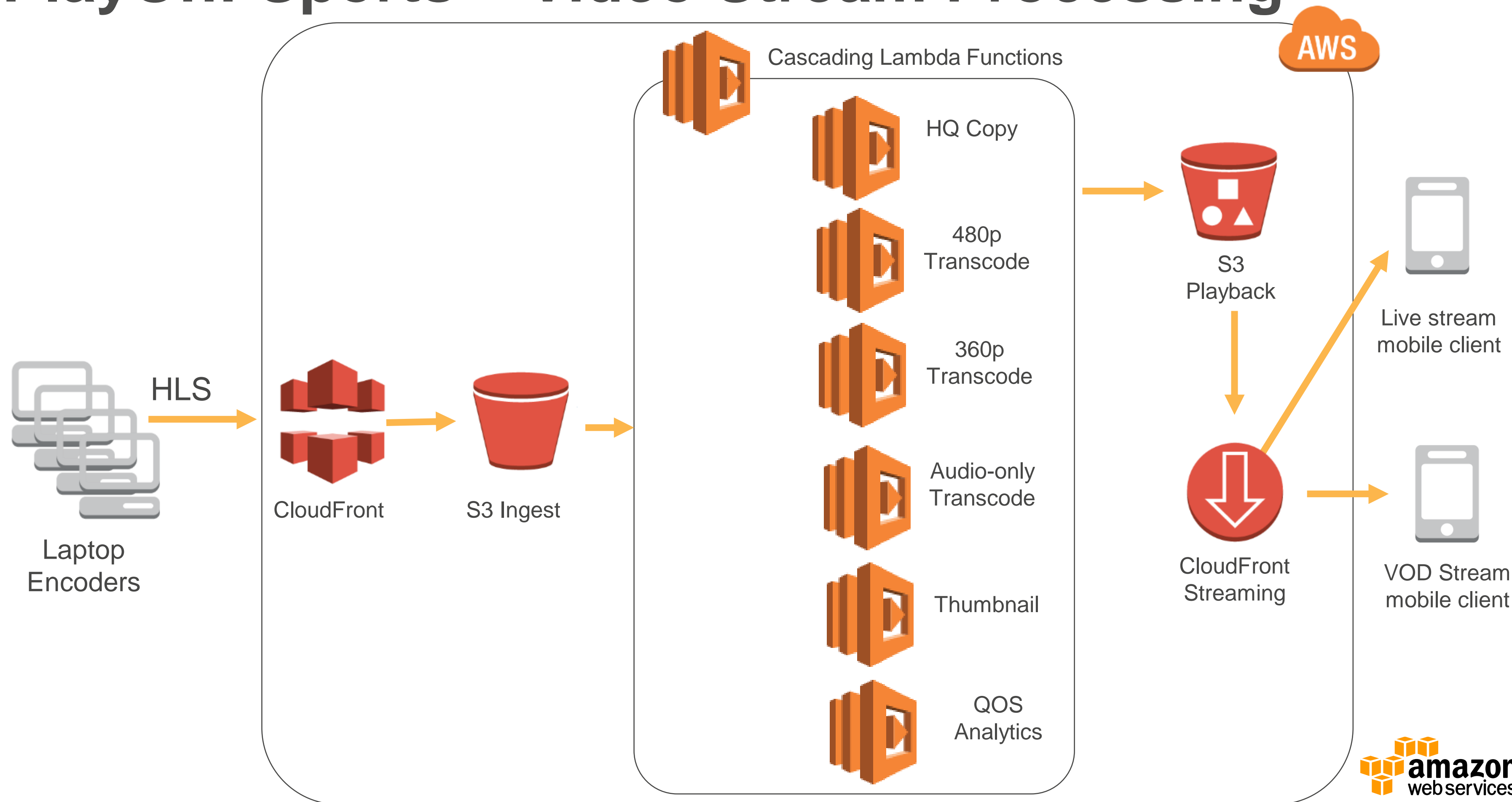
amazon web services

Example Serverless Architecture

amazon web services

http://www.slideshare.net/AmazonWebServices/arc308-the-serverless-company-using-aws-lambda

PlayOn! Sports – Video Stream Processing



“But…
in order to utilize Lambda, do I really need to architect event-driven applications?” – you (maybe)

amazon web services

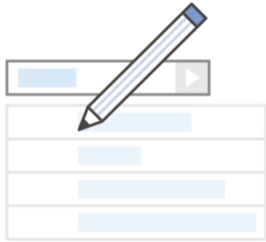
SOA still works.

amazon web services

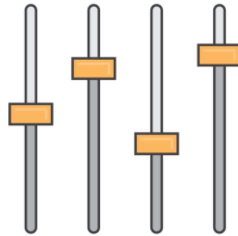
Amazon API Gateway

amazon web services

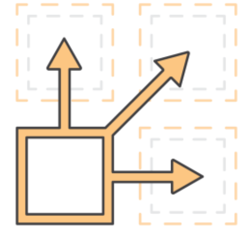
A Fully Managed Service for Your APIs



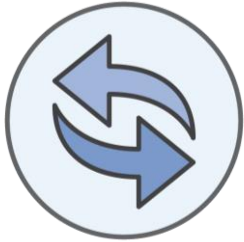
Create



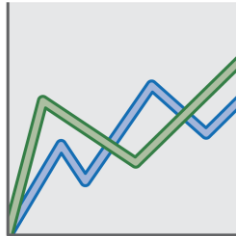
Configure



Publish



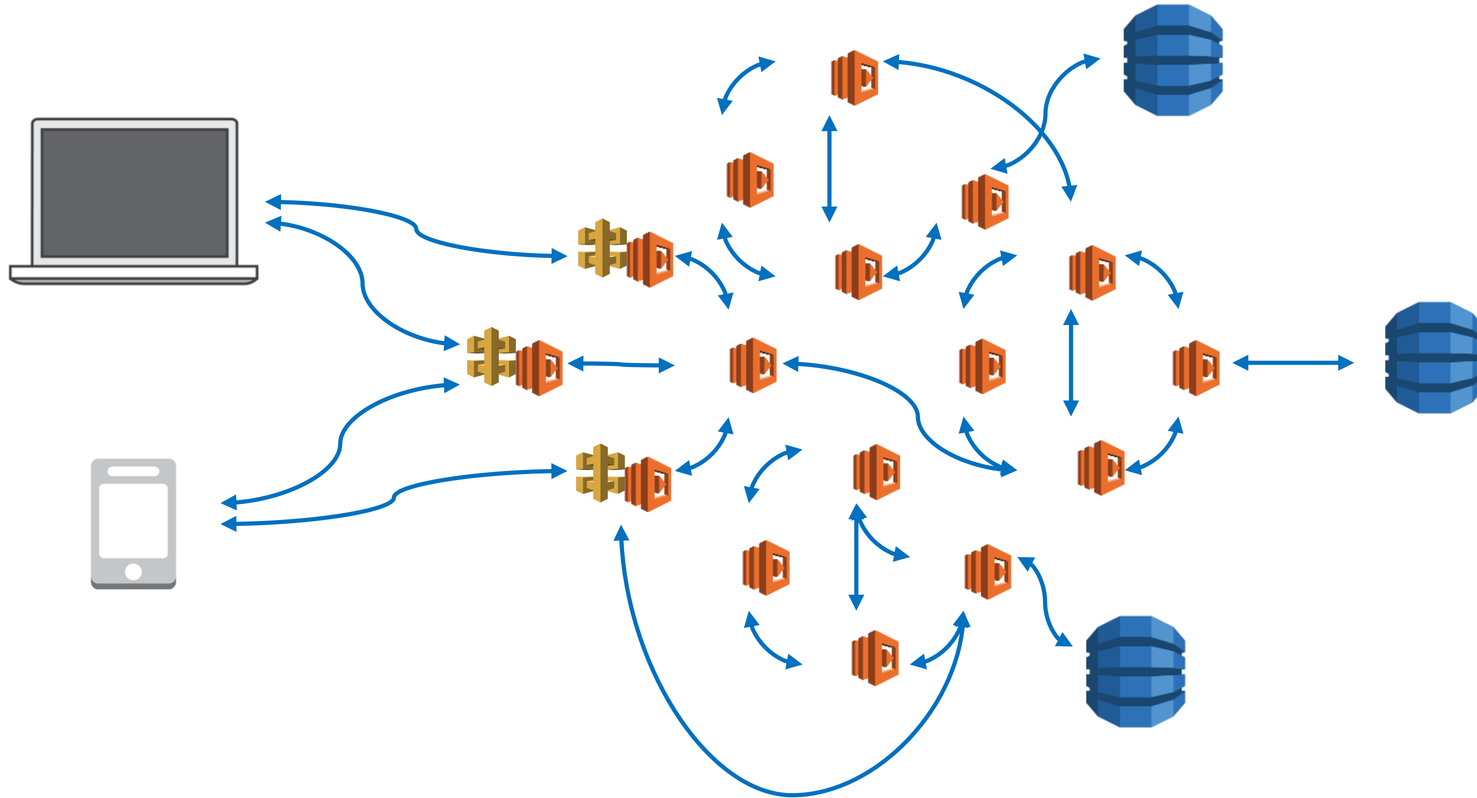
Maintain



Monitor

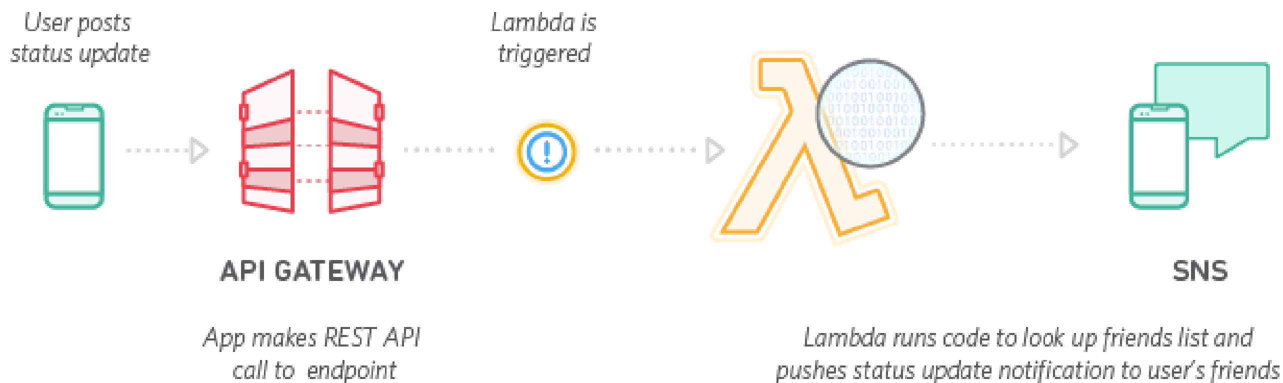
Secure

The Serverless Architecture



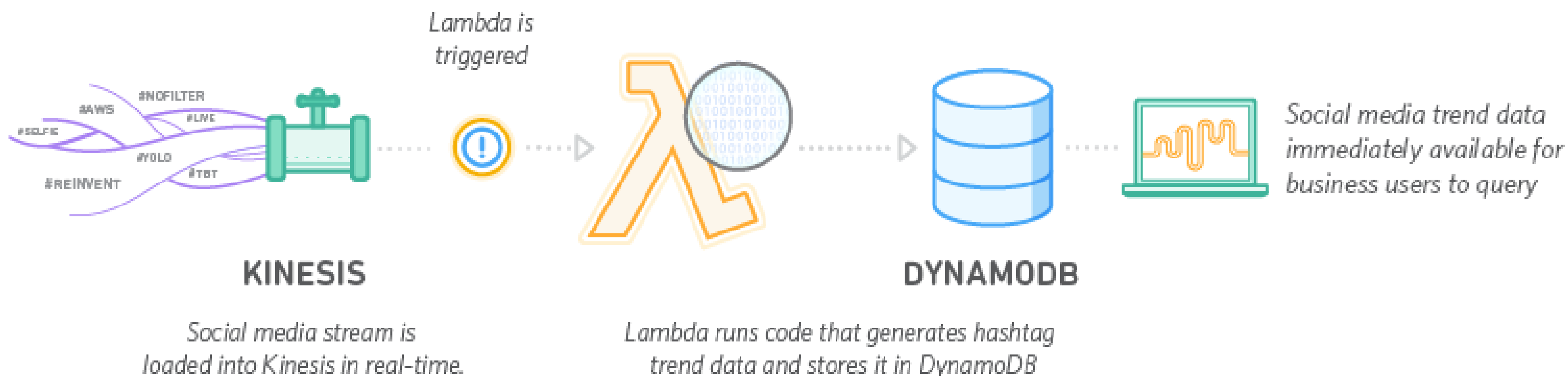
amazon web services

Mobile Backend



amazon web services

Real-time Analytics Engine



amazon web services

Serverless Best Practices

amazon web services

AWS Lambda Best Practices

1. Limit your function size – especially for Java (starting the JVM takes time).
2. Node – remember execution is asynchronous.
3. Don't assume function container reuse – but take advantage of it when it does occur.
4. Don't forget about disk (500 MB /tmp directory provided to each function).
5. Use function aliases for release.
6. Use the included logger (include details from service-provided context).
7. Create custom metrics (operations-centric, and business-centric).

amazon web services

Amazon API Gateway Best Practices

1. Use mock integrations
2. Combine with Amazon Cognito for managed end user-based access control.
3. Use stage variables (inject API config values into Lambda functions for logging, behavior).
4. Use request/response mapping templates everywhere within reason, not passthrough.
5. Take ownership of HTTP response codes.
6. Use Swagger import/export for cross-account sharing.

amazon web services

Additional Best Practices

1. Use strategic, consumable naming conventions (Lambda function names, IAM roles, API names, API stage names, etc.).
2. Use naming conventions and versioning to create automation.
3. Externalize authorization to IAM roles whenever possible.
4. Least privilege <u>and</u> separate IAM roles.
5. Externalize configuration – DynamoDB is great for this.
6. Contact AWS Support before known large scaling events.
7. Be aware of service throttling, engage AWS support if so.

amazon web services

A Call to Action

Go build something!

Amazon API Gateway

AWS Lambda

Amazon DynamoDB

amazon web services

Thank you!

amazon
web services