Building Enterprise-Grade Serverless Apps

Alex Casalboni

Technical Evangelist, AWS

 @alex_casalboni

About me

- Software Engineer & Web Developer
- Serverless Lover & AI Enthusiast
- AWS Customer since 2013

aws

CUSTOMERS INNOVATING WITH AWS LAMBDA

Customer Benefits

iRobot does >1,000 Lambda deployments a day for its serverless IoT backend that runs internet connected-vacuums, with 2M connected robots by 2018 (FY17 projected).

HomeAway uses Lambda to process and prepare 6M user-uploaded photos a month for its vacation rental marketplace.

Fannie Mae is replacing on-premises data centers with a Lambda-based solution that can run a Monte Carlo simulation on 20M mortgage calculations in 1.5 hours.

Agero's accident detection and driver behavior analysis platform handles over 1B Lambda requests each month and scales to handle 20x at peak load.

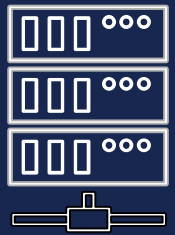
Nextdoor replaced its Apache Flume platform with a serverless data ingestion pipeline that handles 3B events daily.

Revvel reduced video transcoding time by >95% at a fraction of the cost of transcoding videos on server-based solutions.

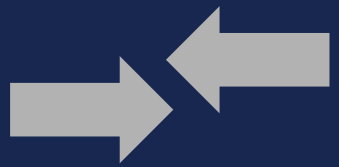
Key Trends

The Rise of Managed Services

Adoption of microservices

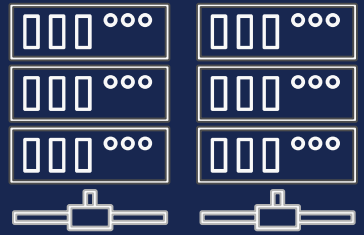


Stream processing and
“embarrassingly parallel” computing

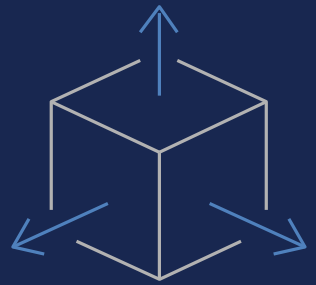


Event-driven architectures

Lambda Means…



No Server Management



Flexible Scaling

High Availability

No Idle Capacity

Agenda: 6 ways to improve your serverless mojo

1. Think in patterns: the serverless architecture playbook
2. Use APIs effectively
3. SecOps like you mean it: permissions & privileges
4. Software lifecycle: from coding to deployment
5. Think in apps
6. When things go wrong: Error-handling and diagnostics

1. Serverless Patterns:
Managed services + Lambda functions

Managed services as building blocks

Compute

AWS Lambda

Storage

Amazon S3

Database

Amazon DynamoDB

Amazon Aurora Serverless

API Proxy

Amazon API Gateway

Messaging

Amazon SQS

Amazon SNS

Analytics

Amazon Kinesis

Amazon Athena

Orchestration

AWS Step Functions

Monitoring and Debugging

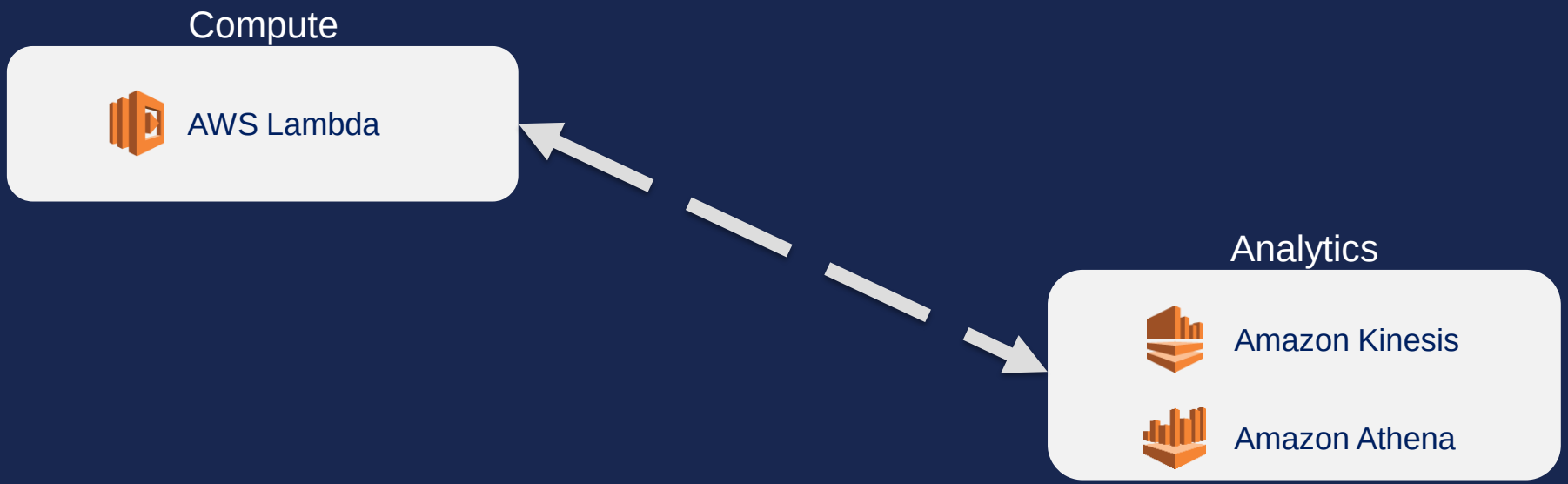
AWS X-Ray

Edge Compute

AWS Greengrass

Lambda@Edge

Example: Serverless analytics



Example: Serverless web app

Compute

AWS Lambda

Static Content

Amazon S3

Dynamic Content

Amazon DynamoDB

Amazon Aurora Serverless (coming soon)

API Serving

Amazon API Gateway

Patterns for the Cloud Era

- Media transform on upload: Amazon S3 event + Lambda
- NoSQL data cleansing: Amazon DynamoDB change streams + Lambda
- Serverless website: Amazon S3 + Amazon DynamoDB + Amazon API Gateway + Lambda
- Click-stream analytics: Amazon Kinesis Data Firehose + Lambda
- Ordered event processing: Kinesis + Lambda
- Multi-function fanout: Amazon SNS (or Lambda) + Lambda
- Workflows: AWS Step Functions + Lambda
- Event distribution: Amazon CloudWatch Events + Lambda
- Serverless cron jobs: CloudWatch timer events + Lambda
- GraphQL actions: AWS AppSync + Lambda
- On-the-fly image resizing: AWS Lambda@Edge + Amazon CloudFront
- Email rules: Amazon SES + Lambda
- Configuration policy enforcement: AWS Config + Lambda
- Stored procedures: Amazon Aurora + Lambda
- Custom authorizers for APIs: API Gateway auth + Lambda
- DevOps choreography: CloudWatch alarms + Lambda
- Alexa skills: Amazon Alexa + Lambda
- Chatbots: Slack + Amazon Lex + Lambda
- IoT automation: AWS IoT + Lambda
- Smart devices: AWS Greengrass + Lambda
- On-prem file encrypt for transit: AWS Snowball Edge + Lambda
- …

Meta-patterns

1. Service pushes async event to Lambda (S3, SNS, SQS)
2. Lambda grabs event from service (DynamoDB, Kinesis)
3. Synchronous exchange (Alexa, Lex)
4. Batch transform (Kinesis Firehose)
5. Microservice (API + Lambda + your choice of DB)
6. Customization via functions (AWS Config, SES rules)
7. Data-driven fanout (S3-Lambda, Lambda-lambda)
8. Choreography (Step Functions + Lambda)
9. Lambda functions in devices (Greengrass, Snowball Edge)

Patterns: Best practices

- Don’t reinvent the wheel – use managed services when possible
  - …and stay current ☺
- Keep events inside AWS services for as long as possible:
  - Example: S3 → Lambda → client is preferable to S3 → client → Lambda
  - Verify limits end-to-end
- Prefer idempotent, stateless functions, and when you can’t…
  - Use Step Functions if you need stateful control (retries, long-running)
- Building a SaaS or ISV platform? Use the techniques AWS uses:
  1. Architect it serverlessly using Lambda and API Gateway
  2. Offer your ecosystem customization via functions
  3. Enable customers and partners to build reactive systems by emitting events

2. Use APIs Effectively

Start simple!

AWS SAM

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
 GetFunction:
   Type: AWS::Serverless::Function
   Properties:
     Handler: index.get
     Runtime: nodejs6.10
     CodeUri: s3://bucket/api_backend.zip
     Policies: AmazonDynamoDBReadOnlyAccess
     Environment:
       Variables:
         TABLE_NAME: !Ref Table
     Events:
       GetResource:
         Type: Api
         Properties:
           Path: /resource/{resourceId}
           Method: get
```

Extend with Swagger

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  Api:
    Type: AWS::Serverless::Api
    Properties:
      StageName: prod
      DefinitionUri: swagger.yml
```

```
---
swagger: "2.0"
info:
  version: "2018-06-06T01:36:07Z"
  title: "PetStore"
host: "0ftv5igkjd.execute-api.us-east-
1.amazonaws.com"
basePath: "/test"
schemes:
- "https"
paths:
  /:
    get:
      consumes:
      - "application/json"
      produces:
      - "text/html"
      responses:
        200:
          description: "200 response"
          headers:
            Content-Type:
              type: "string"
```

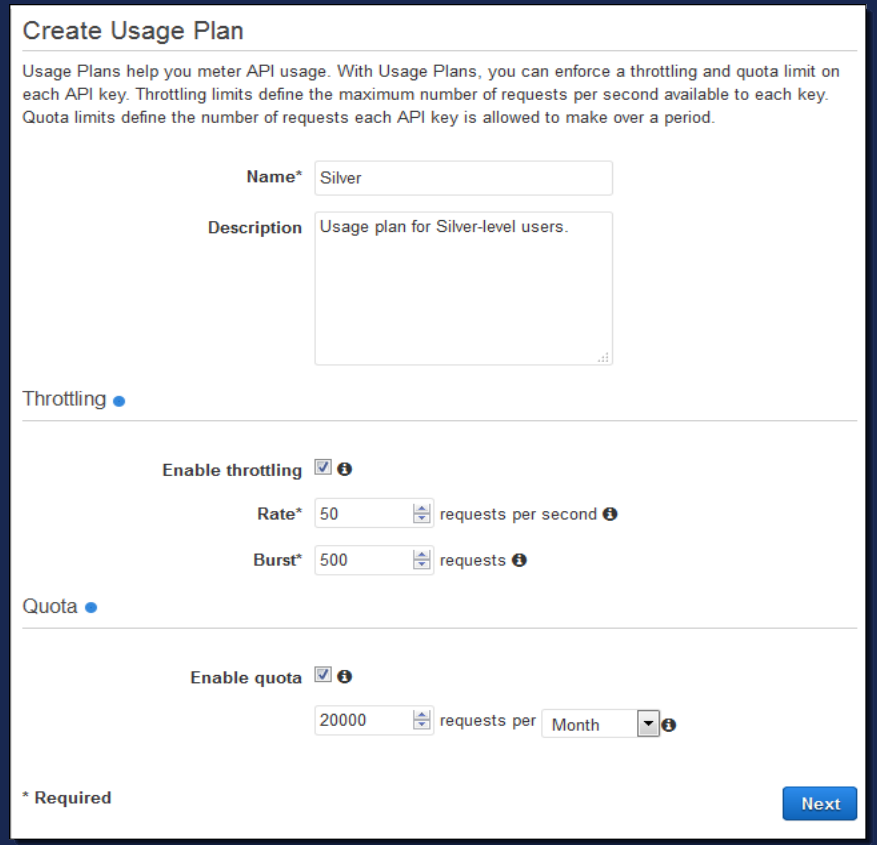
Choose the right API Endpoint Type

API Endpoint Type determines where clients connect to the published APIs

- **Edge optimized:** Designed for mobile clients, uses integrated CDN (CloudFront) to enable clients to connect to the closest POP
- **Regional:** Designed for EC2 clients, within the same region as the API
- **Private:** Designed for VPC only access, with no Internet access needed. (Uses PrivateLink VPC Endpoint)

Manage throttling and rate-limiting

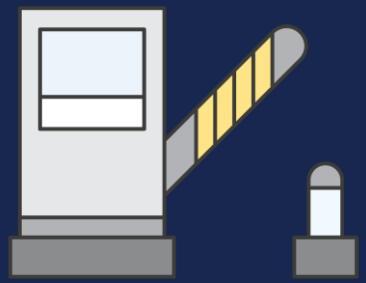
- Protect backends with method level rate-limiting *New!*
- Protect clients from aggressive use with Usage Plans
  - Limit on RPS
  - Limit on requests per day/week/month



ICYMI – Summary of recent API Gateway launches

- Fine-grained (method-level) throttling
- Overrides on request/response parameters and response status
- Higher limits on APIs and resource change rates
  - 600 APIs (regional or private endpoint)
  - 120 APIs (edge-optimized)
- Resource policies and cross-account access enabled
- AWS X-Ray Support (distributed tracing)
- OpenAPI 3.0 Support (import/export)

3. Security and AuthZ

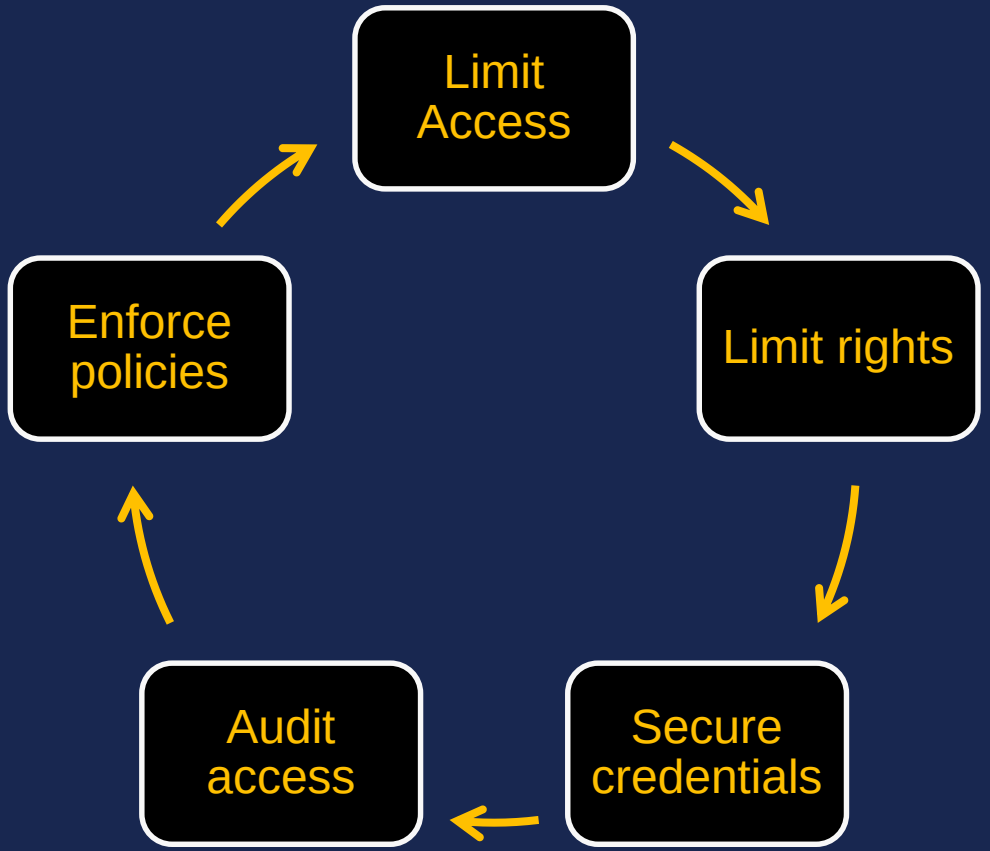


Serverless apps can be the *most* secure apps, if…

…you use the tools provided:

- Resource policies
- Custom authorizers
- Execution roles
- AWS Config policies
- AWS CloudTrail traces

Full circle of protection



Limit access: Resource policies and custom authorizers

Analogy: Who would I invite into my house?

API Gateway: Resource policies vs. Custom Authorizers

- Which should I use?
  - If resource policies can express the restriction, use them. (They're free, fast, and built in.)
- How do these two features interact?
  - We evaluate in three phases; all 3 have to pass:
    - Pre-authN (time checks, source IP checks)
    - Post-authN (call principal checks)
    - Custom authorizer (if present)
  - *We stop as soon as permission is denied, so we only run your custom authorizer if it's necessary to make a decision.*

Limits rights: Execution and invocation roles

Analogy: What do I let my kids play with?

Keeping track: Auditing and Config Policies

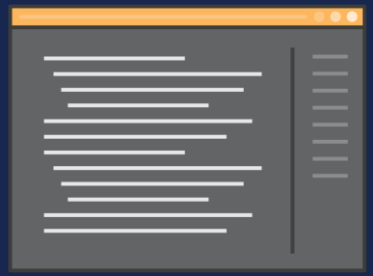
- Use AWS CloudTrail to track API and function management (creation, deletion, retrieval, etc.)
  - Can also audit function *invocation* for Lambda
- Run queries against CloudTrail data using Amazon Athena
  - Or build a simple analysis engine using Lambda functions
- Use AWS Config to continuously monitor and enforce policies
  - For example: Enforcing resource-level scoping on Lambda functions or custom authorizers for APIs

Calling Third Party Services Securely

AWS Secrets Manager

- Securely store, retrieve, and manage database credentials, API keys, and other secrets for AWS or third-party systems
- Rotate, audit, and version secrets
- Built-in support for MySQL, PostgreSQL, and Amazon Aurora on Amazon RDS
- Supports both default and custom KMS keys
- *Avoid cleartext secrets in Lambda env vars or code!*

4. Building, Testing, and Deploying Serverless Apps



AWS Code Suite

AWS CodeCommit or GitHub → AWS CodeBuild (build) → AWS CodeBuild (test) → CloudFormation (deploy) → AWS CodeDeploy

AWS CodePipeline

AWS Cloud9 Editor

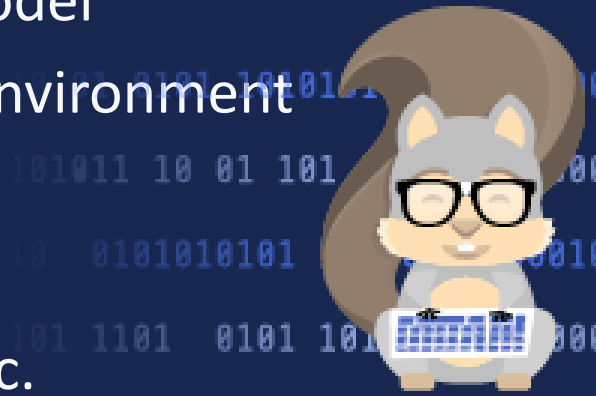
AWS CodeStar

Best Practices for Serverless Development

- CI/CD pipeline for apps (including repo publishing!)
- Test and debug locally
- AWS CloudFormation can deploy to multiple regions
    - Useful for multi-region APIs
- Deploy incrementally for safety
    - Weighted aliases in Lambda
    - Canary stages in API Gateway
    - AWS CodeDeploy can automate SAM app deploys (including rollback)

Local test, build, & debug with SAM Local

- SAM = Serverless Application Model
- Copy of the Lambda execution environment
- Powers local test/debug
- Models serverless applications
- Works in IDEs, command line, etc.
- Use “sam init” to create a new project
- Open source



<demo time>

5. It’s All about [Serverless] Apps

AWS Serverless Application Repository

- Discover, customize, and deploy serverless apps
- Publish and share apps three ways…
    - *Within* your AWS account
    - *Across* accounts
    - With everyone (*public* access)

Growing Set of App Producers

AWS Serverless Application Repository

Customize and create directly from the Lambda console, command line, or SDK

- Based on SAM – simple model of your serverless application
- Deploys using CloudFormation (use anywhere CloudFormation works!)
- Complements GitHub (source) with deployable (built) artifacts
- Easiest way to help developers get started, whether you're in an enterprise or working on an open source project
    - Will replace blueprints in the console

<demo time>

6. When Things Go Wrong: Error-Handling and Diagnostics

API Gateway CloudWatch Metrics

Default Metrics

Included for free

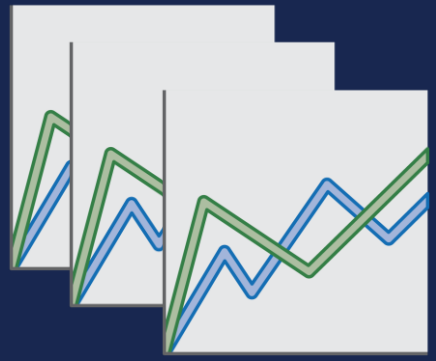
Granularity: per-API stage

Detailed Metrics

CloudWatch pricing

Granularity: per-method

Can be globally enabled



Lambda CloudWatch Metrics (free)

At both function and version level:

- Invocations, Errors, Throttles, IteratorAge, DeadLetterError

At both account and function level (when limit is set):

- ConcurrentExecutions

At the account level only:

- UnreservedConcurrentExecutions

API Gateway CloudWatch Logs

API Gateway Logging

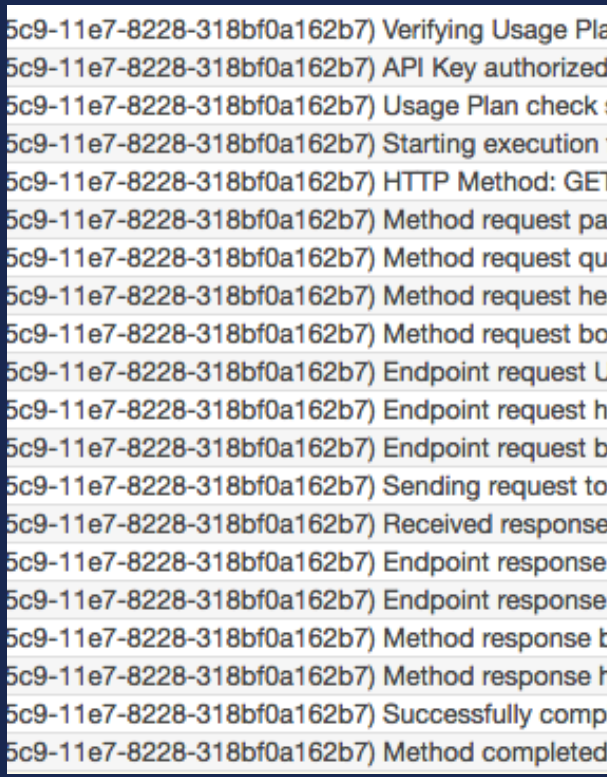
- Two levels of logging, error and info
- Optionally log method request/body content
- Set globally in stage, or override per method

API Gateway Access Logging

- Customizable format for machine parsable logs

Log Pivots

- Build metrics based on log filters
- Jump to logs that generated metrics



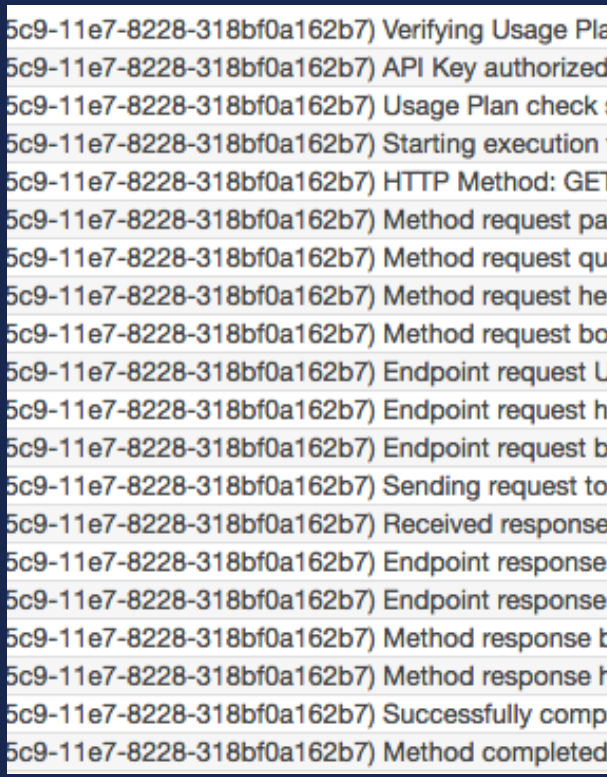
Lambda CloudWatch Logs

Basics:

- Automatically configured
- Basic info (requests, duration, memory consumption, etc.) handled automatically
- Application can add log entries easily

Cool Stuff:

- Grab a range on the console graph and jump to the logs for that time range
- Send logs to another Lambda function
- Use Elasticsearch & Kibana to aggregate/analyze



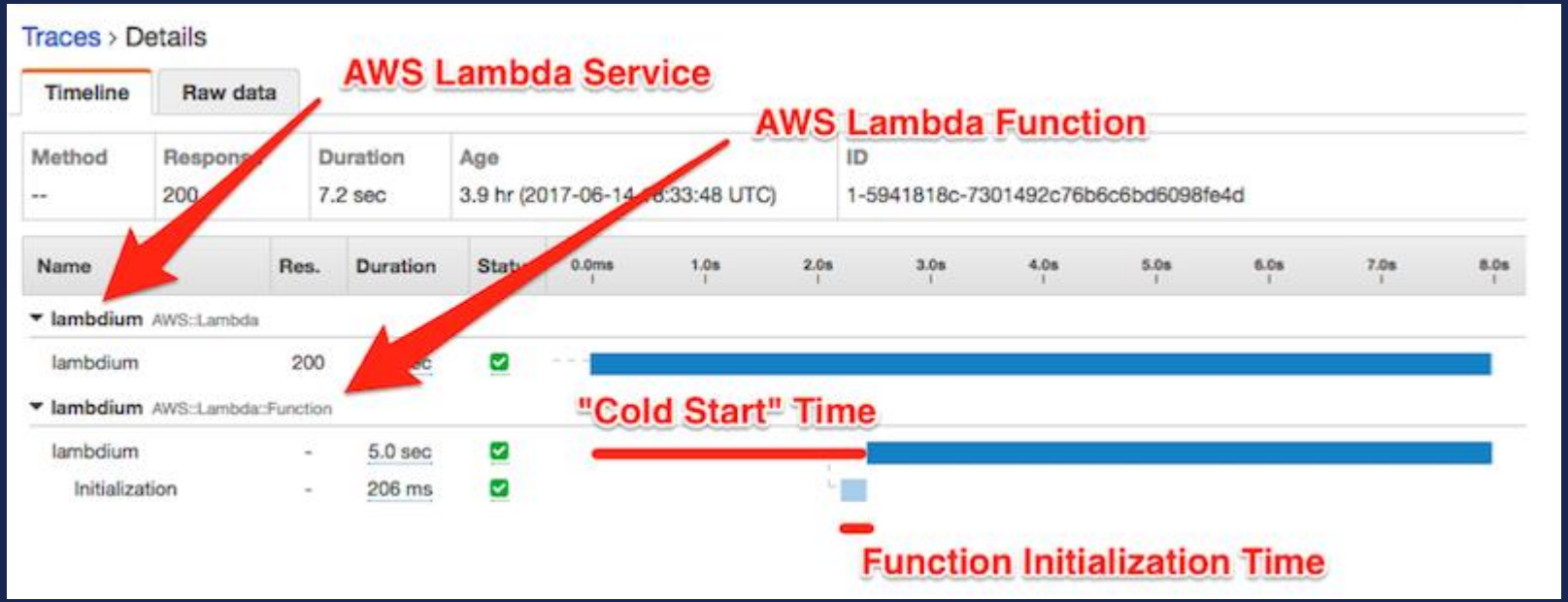
The Lambda Function Request Lifecycle

1. Download your code (as a SquashFS filesystem)
2. Start the language runtime
3. Load your code (e.g., class loading in Java)
4. Run “init” code (e.g., static initializers in Java)
5. Process the request

warm

cold

Same View in AWS X-Ray



Things You Can Track in X-Ray for Lambda

- Cold vs warm starts
- Async dwell time (time event spends in queue)
- Duration of calls to other AWS services (and with a little work to add trace points, third-party services)
- Errors and throttles from AWS service calls

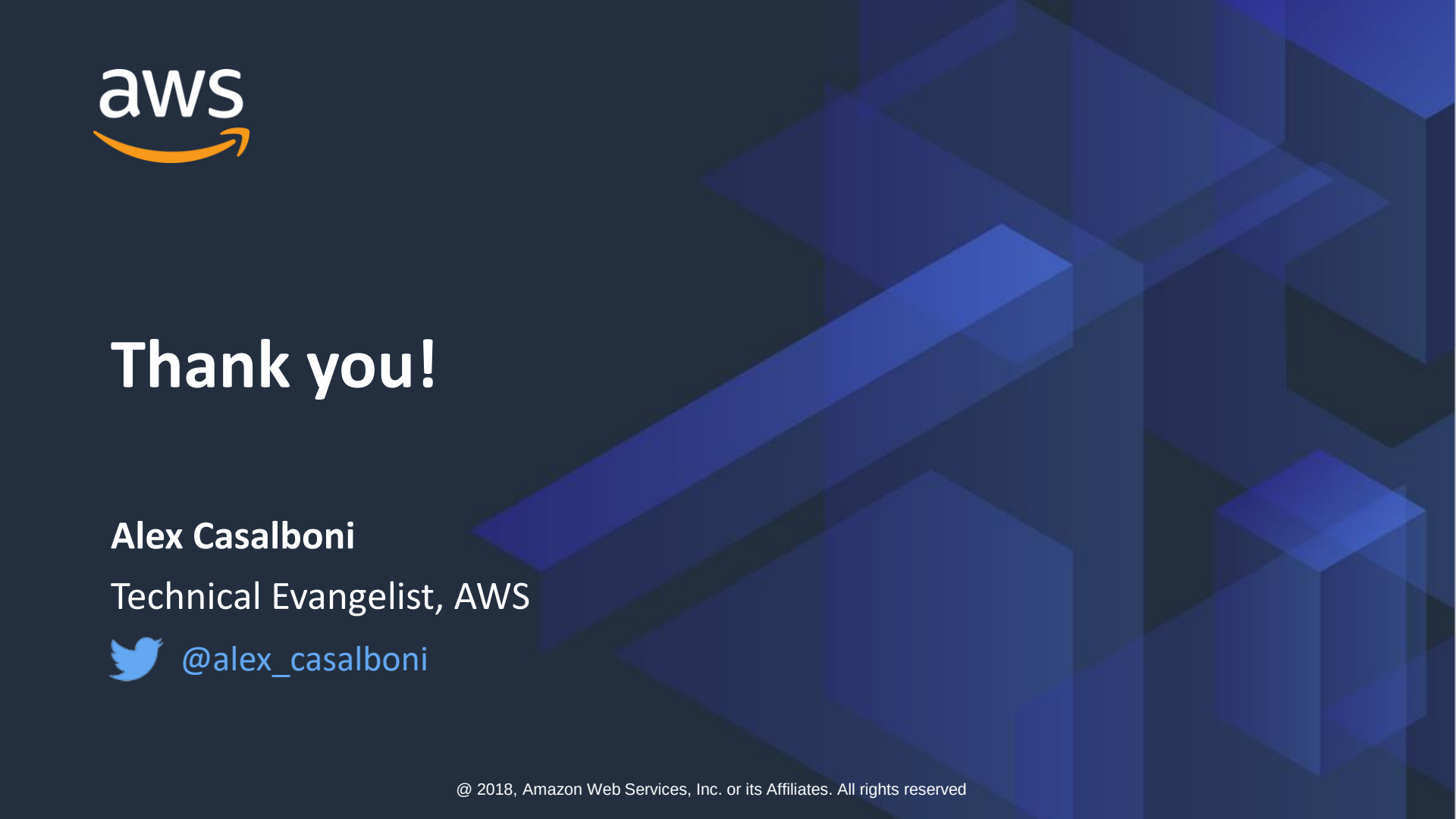
Lots of great third-party offerings can also help!

Making Cold Starts More Efficient

- Control the dependencies in your function's deployment package
  - Tree shake your Java code! (Hint: Spring is expensive ☹)
- Can combine ‘cold’ and ‘hot’ functions
- Optimize for your language, where applicable…
  - Node – Browserfy, Minify
  - AWS Java v2 SDK (preview)
- Only use VPC if you need access to a resource there!
- Delay loading where possible

Summary: 6 ways to improve your serverless mojo

1. Think in patterns: the serverless architecture playbook
2. Use APIs effectively
3. SecOps like you mean it: permissions & privileges
4. Software lifecycle: from coding to deployment
5. Think in apps
6. When things go wrong: Error-handling and diagnostics



Thank you!

Alex Casalboni

Technical Evangelist, AWS

@alex_casalboni