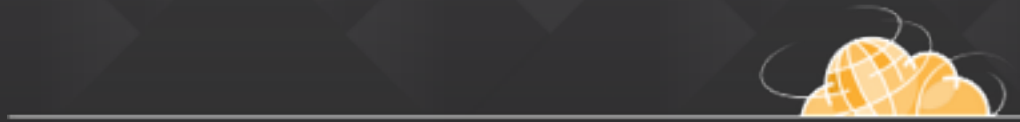


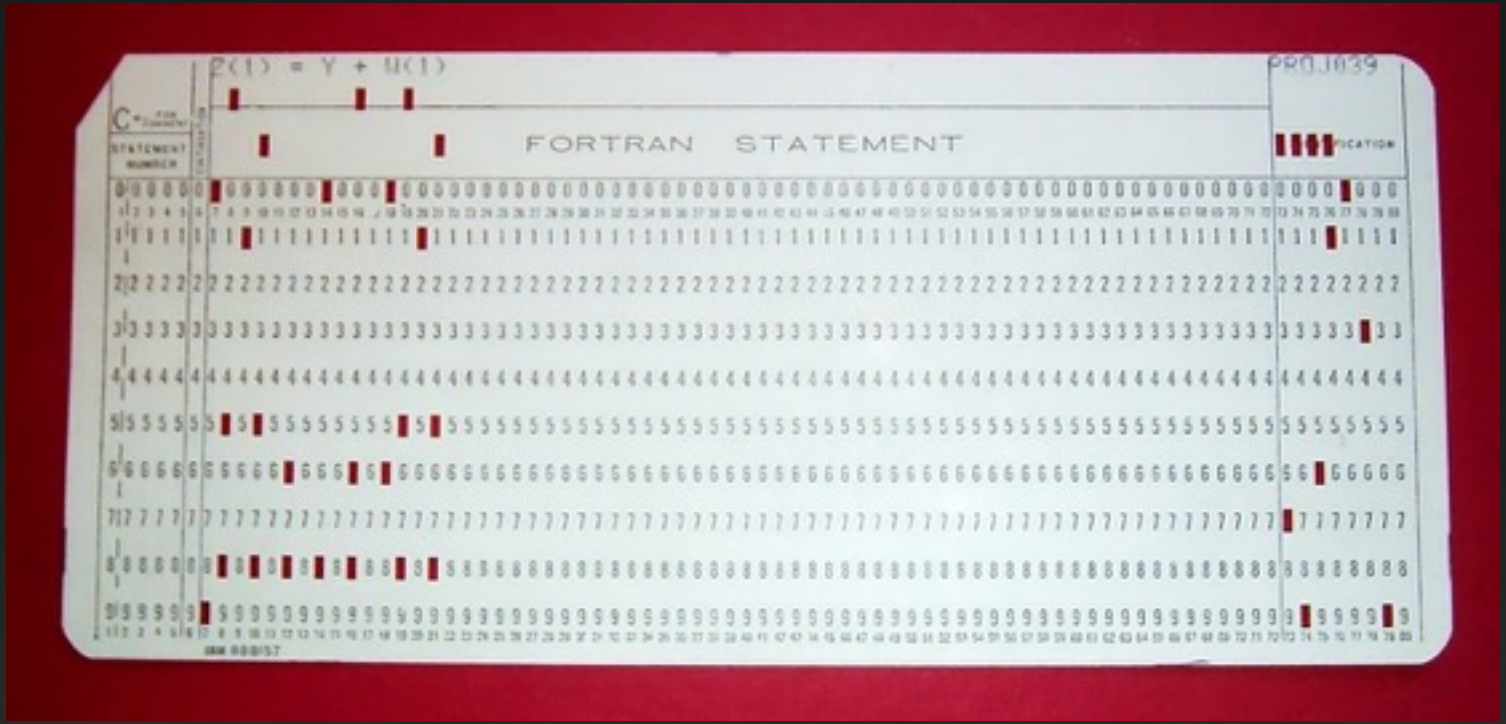
Serverless Design Patterns
with AWS Lambda

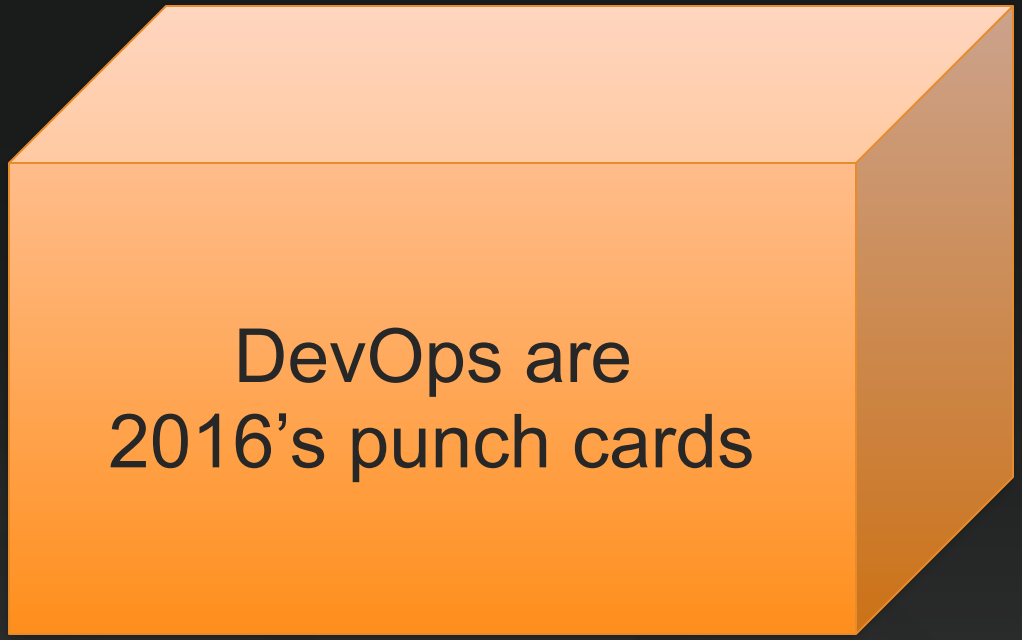
Dr. Tim Wagner
General Manager, AWS Lambda and Amazon API Gateway

April 11, 2016

EMERGING TECHNOLOGIE
FOR THE ENTERPRISE CONFERENC

amazon web services

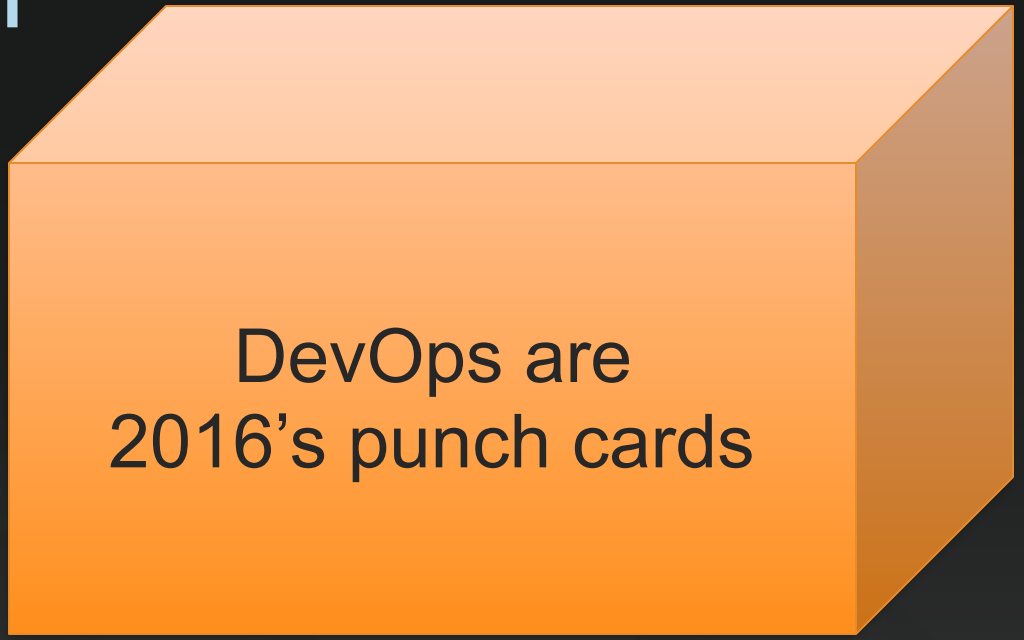




Which operating system patch level?

How many servers?

What kind of servers?



Which deployment tool?

Do we have the security patch?

Are we running the right version?

Scalability, reliability, and fault tolerance are hard.

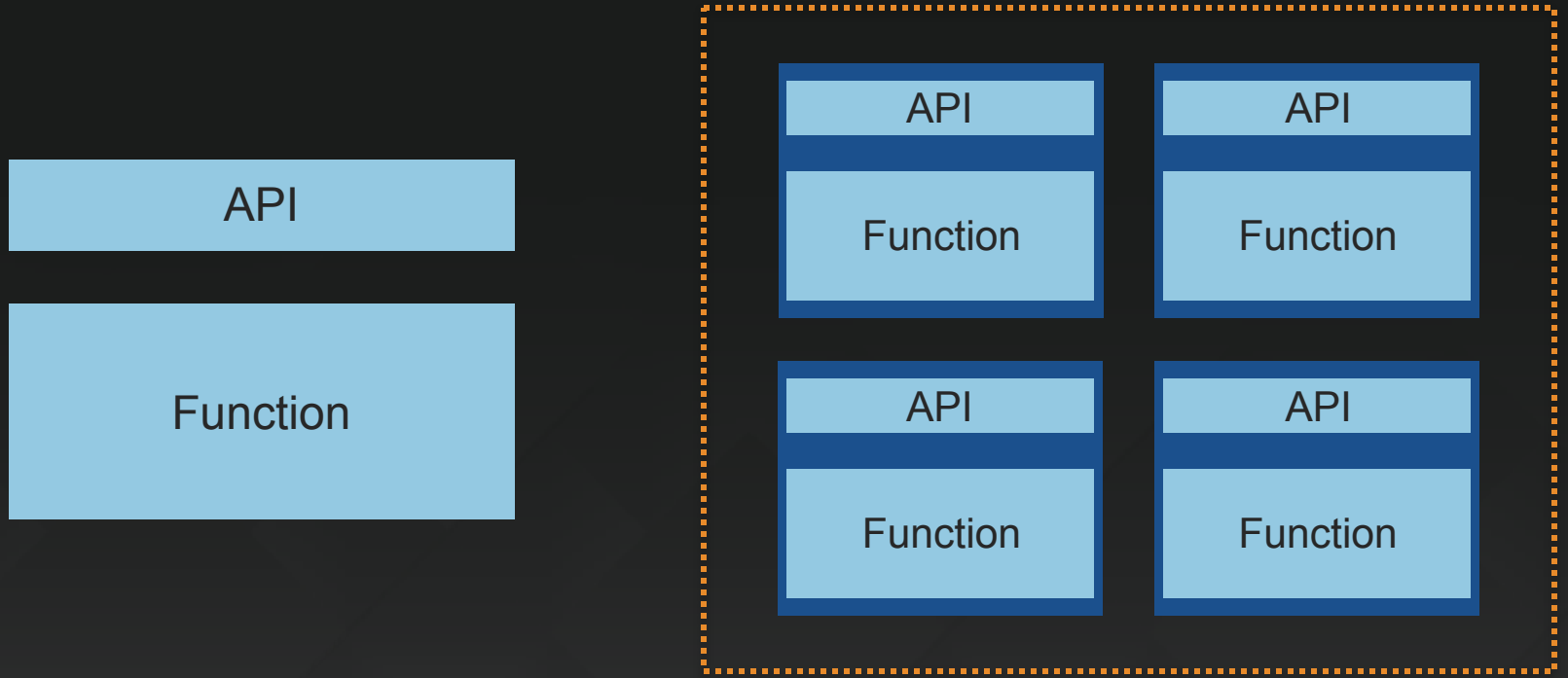
What if you could achieve them without working for it?

A: Serverless

Q: What is the right abstraction level for the cloud?

Too curated
Fat clients
Wrong unit: Deployed (monolithic) applications

[Micro] Application Design for the Cloud



The Serverless Compute Manifesto

- Functions are the unit of deployment and scaling.
- No machines, VMs, or containers 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.

AWS Lambda and Amazon API Gateway

Microservices where you *don't* have to think about:

- Servers
- Being over/under capacity
- Deployments
- Scaling and fault tolerance
- OS or language updates
- Metrics and logging
- Paying for idle time

…but where you can:

- Quickly code in the cloud
- Bring your own code… even native libraries
- Create backends, event handlers, and data processing systems
- Easily call enterprise APIs
- Run code in parallel
  Democratized scale!

So… how do you build apps?

What They Don’t Look Like

- Not monolithic
  - Expressed as a collection of independent functions
  - …with APIs where you need abstraction boundaries
- Skip the boring parts
  - AWS provides the webserver and API hosting
    (plus, we need to see the requests)
  - AWS provides basic metrics and logging
    (plus, it’s harder than it sounds)
- Skip the hard parts (distributed algorithms)
  - Fault-tolerance and scaling don’t show up in the app code

What They *Do* Look Like

- Individual functions
  - Often with HTTP endpoints
  - Sometimes hooked up to events or other services
    - Often asynchronous
  - That can also run on a schedule
  - *Not necessarily simple*
    - Application scaling / parallelism should *always* be simple
    - Algorithms should be as complicated as they need to be
      - Serverless scales *down* to a single line of Python….
      - …and scales *up* to a 50+ MB C++ algorithm

But… how do I scale, really?

How to Run a 1 Hour Task

- Old school: Nice big server running for an hour
  - What happens if the machine dies or there's an intermittent error?
  - Takes an hour. (Unless the machine dies, then it could take 2.)
  - What if it really only took 15 minutes? 75% waste…ugh.
  - Popular metaphor: *Pets versus cattle*
- New school: 60, 1-minute jobs
  - Who cares if a machine dies? That job just restarts.
  - Who cares what they run on? Perf comes from *parallelism*, not *hw*.
  - Done in a minute (worst case, 2).
  - No wasted time or money. Guaranteed.
  - Metaphor: *Drive-through window*

What Programming Model???

- Choice of language
  - Java method
  - Python function
  - Node.js function
  - Bring your own (yep, we support that)
- Choice of library (anything you want)
- JSON as schema / inter-service format
- HTTPS on the wire

Let’s see it…

Events: JSON + Functions

Amazon S3

Amazon DynamoDB

Amazon Kinesis

AWS CloudFormation

Amazon SWF

AWS CloudTrail

Amazon SES

Amazon Cognito

Amazon API Gateway

Amazon SNS

Amazon CloudWatch Logs / Events

AWS IoT

amazon web services

Example: Build an audit system in 5 minutes

1. Turn on AWS CloudTrails (AWS API logging).
2. Hook up Amazon CloudWatch events to your AWS Lambda audit function.
3. Send yourself a text message if your filter detects something suspicious.

Frameworks Bad; Services Good

- Amazon S3: object storage
- Amazon DynamoDB: NoSQL database
- Amazon SNS: messaging and notifications
- Amazon Kinesis: real-time streaming
- Amazon SES: email sending and receiving
- Amazon CloudWatch Events: event hub
- …and the rest of AWS
- …and anything else on public Internet (aka “no lockin”)
- …plus your own private APIs (aka “VPC” / enterprise)

Drill-down: Resource Sizing in AWS Lambda

- AWS Lambda offers 23 “power levels”
- Higher levels offer more memory *and* more CPU power
  - 128 MB, lowest CPU power
  - 1.5 GB, highest CPU power
- Higher power levels for CPU-bound and bursty tasks
- Compute price scales with the power level
- Billed duration ranges from 100ms to 5 minutes

Drill-down: Versioning

- Immutable versions of your functions
- One mutable working arena per function
- Aliases: server-side updates
- Blue/green deployments? Not needed, but…
- Traffic shaping? Sure! (and easy)
- API Gateway
  - Immutable deployments
  - API stages
  - Swagger import (“API as code”)
    - Goal: Open source microservice representation for both APIs *and* code

But what *is* it?

- Linux containers as an *implementation*, not a programming or deployment *abstraction*
  - Process and network isolation, cgroups, seccomp, …
  - Off-the-shelf language runtimes
  - We *minimize* innovation here
- The world's biggest bin-packing algorithm
  - High speed, highly distributed work routing and placement
  - We *maximize* innovation here
- Predictive capacity management
  - Purpose-built, massively scaled Language-Runtime-aaS
- API Gateway: Swagger interpreter as reverse proxy

Serverless Design Patterns (or, What can you do with it?)

Serverless Web Apps

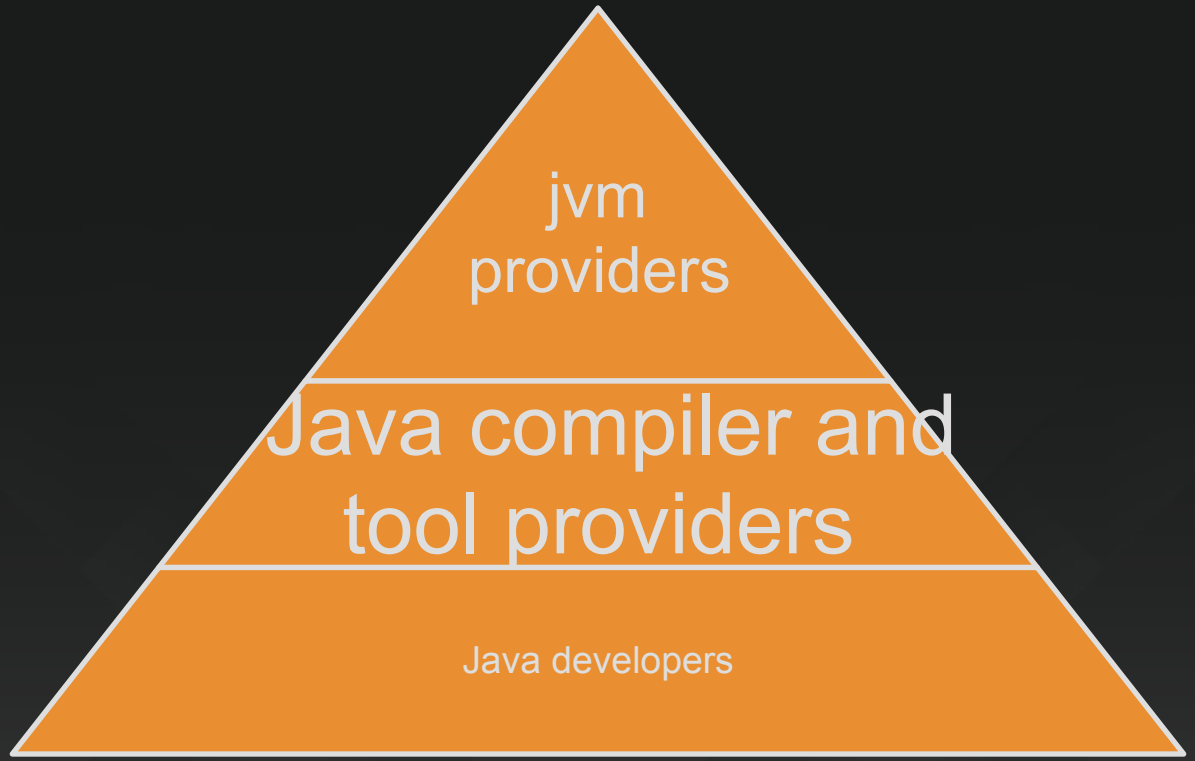
1. Amazon S3 for serving static content
2. AWS Lambda for dynamic content
3. Amazon API Gateway for https access
4. Amazon DynamoDB for NoSQL data storage

A Scalable Backend for Mobile Apps or IoT

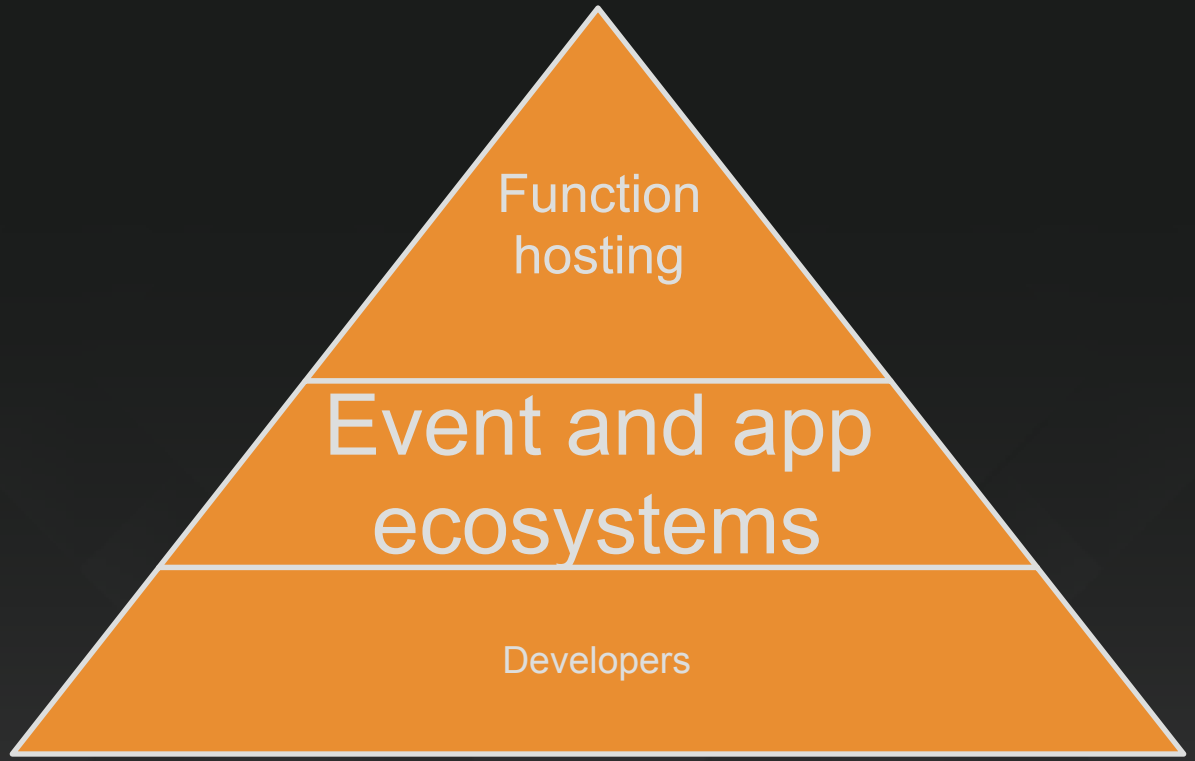
1. Pick one:
    a. Mobile apps: AWS Mobile SDK + Amazon Cognito (authorization)
    b. IoT devices: AWS IoT
2. AWS Lambda's "Mobile Backend" blueprint
3. Amazon DynamoDB for data storage

amazon web services

Principle of Complexity Concentration

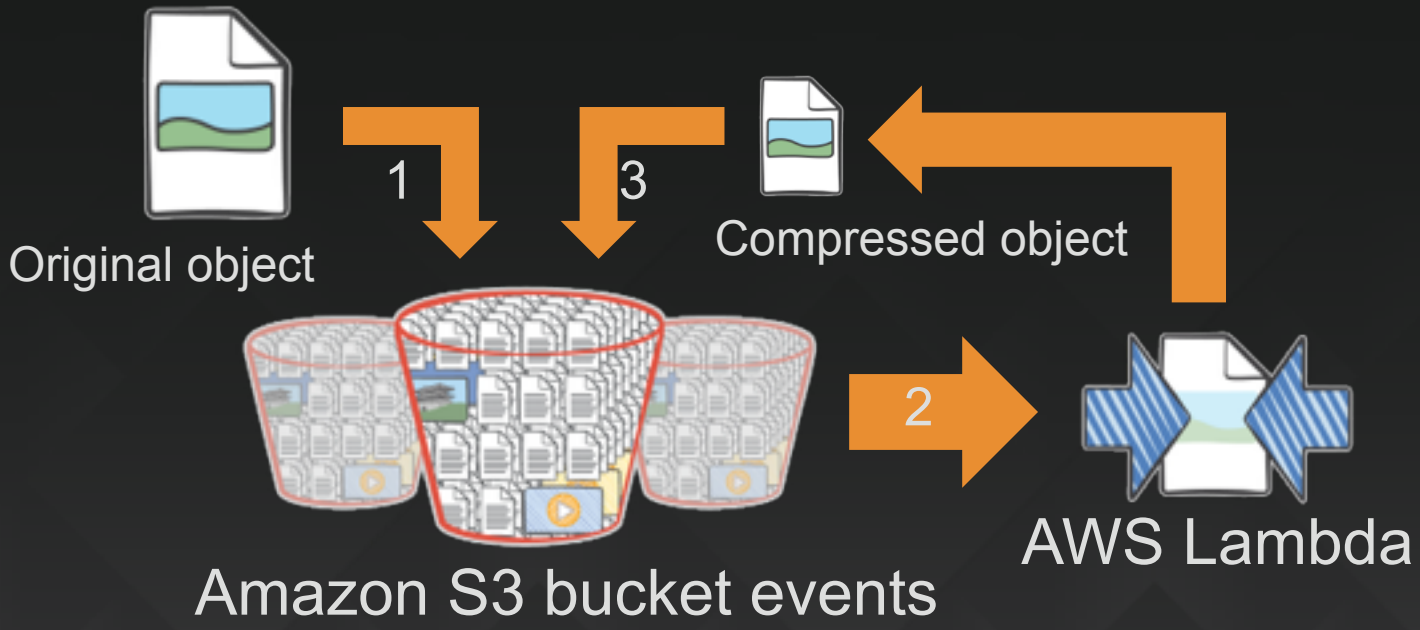


Principle of Complexity Concentration

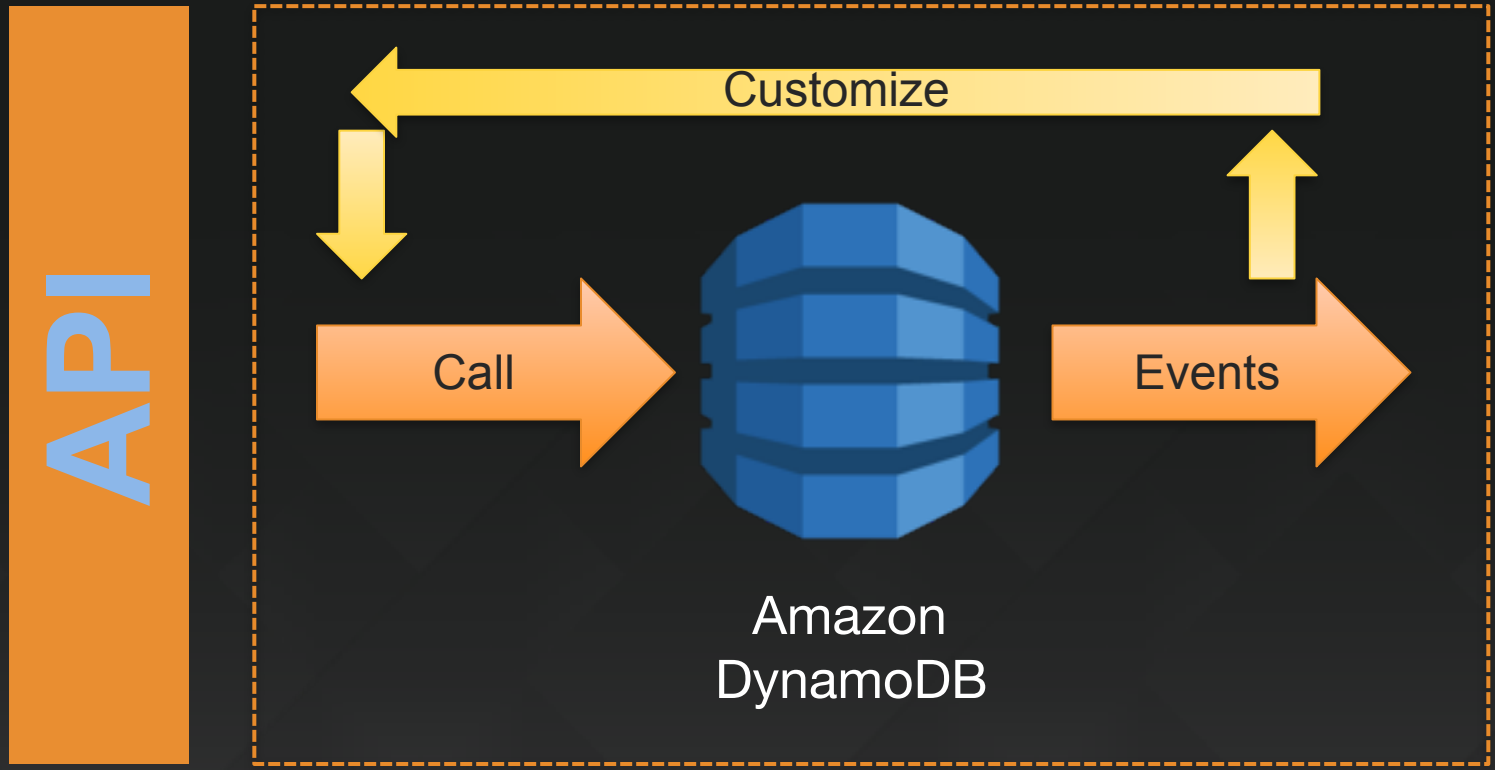


Amazon S3 Bucket Triggers

- Pick the S3 Lambda blueprint
- Select your bucket as the event source

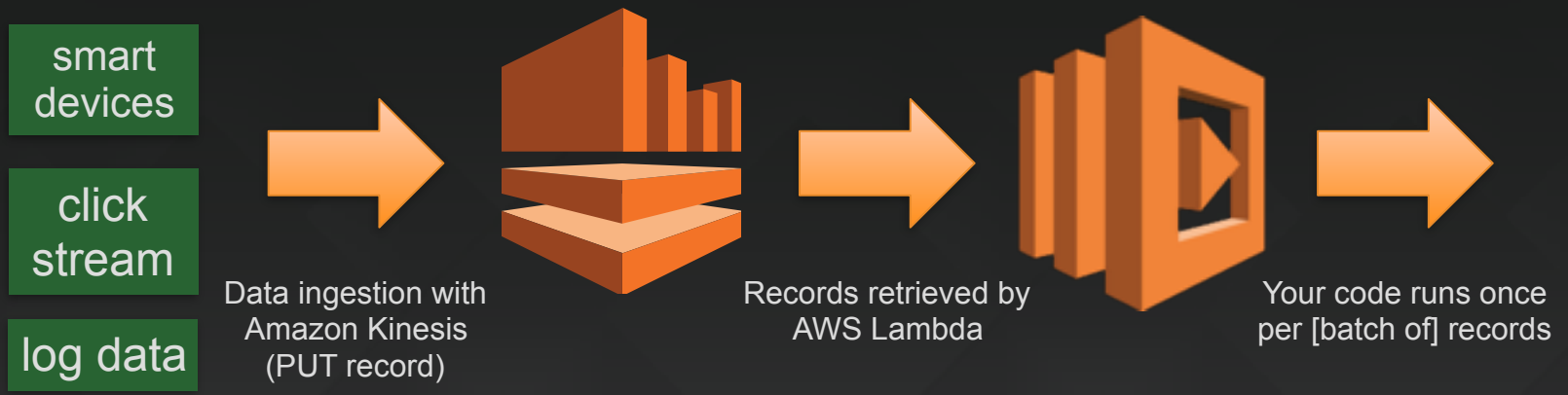


Aside: Functions for Behavioral Abstraction



Real-time Analytics Processing

1. Amazon Kinesis for high-speed data ingestion
2. Select AWS Lambda's "Kinesis" blueprint
3. Store aggregated results in Amazon Redshift, Amazon S3 Amazon DynamoDB

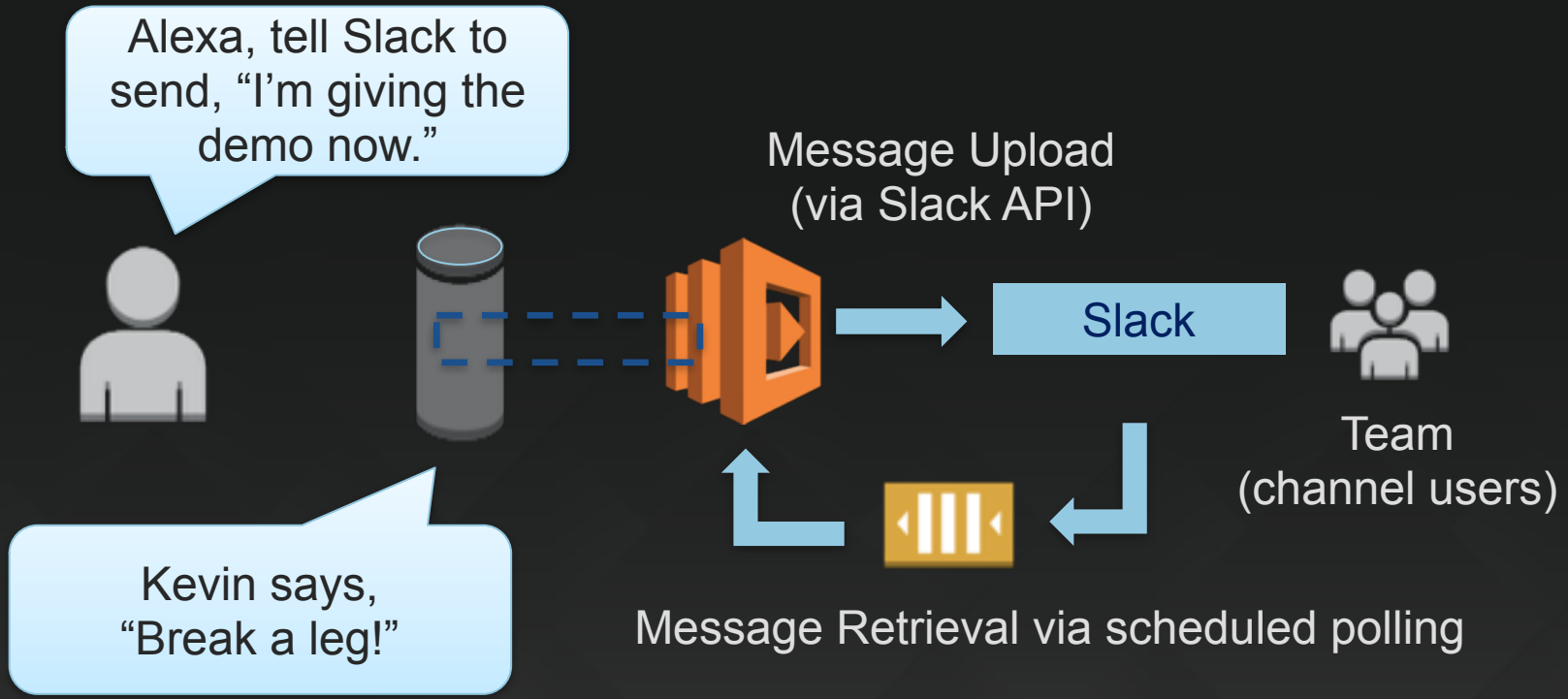


amazon web services

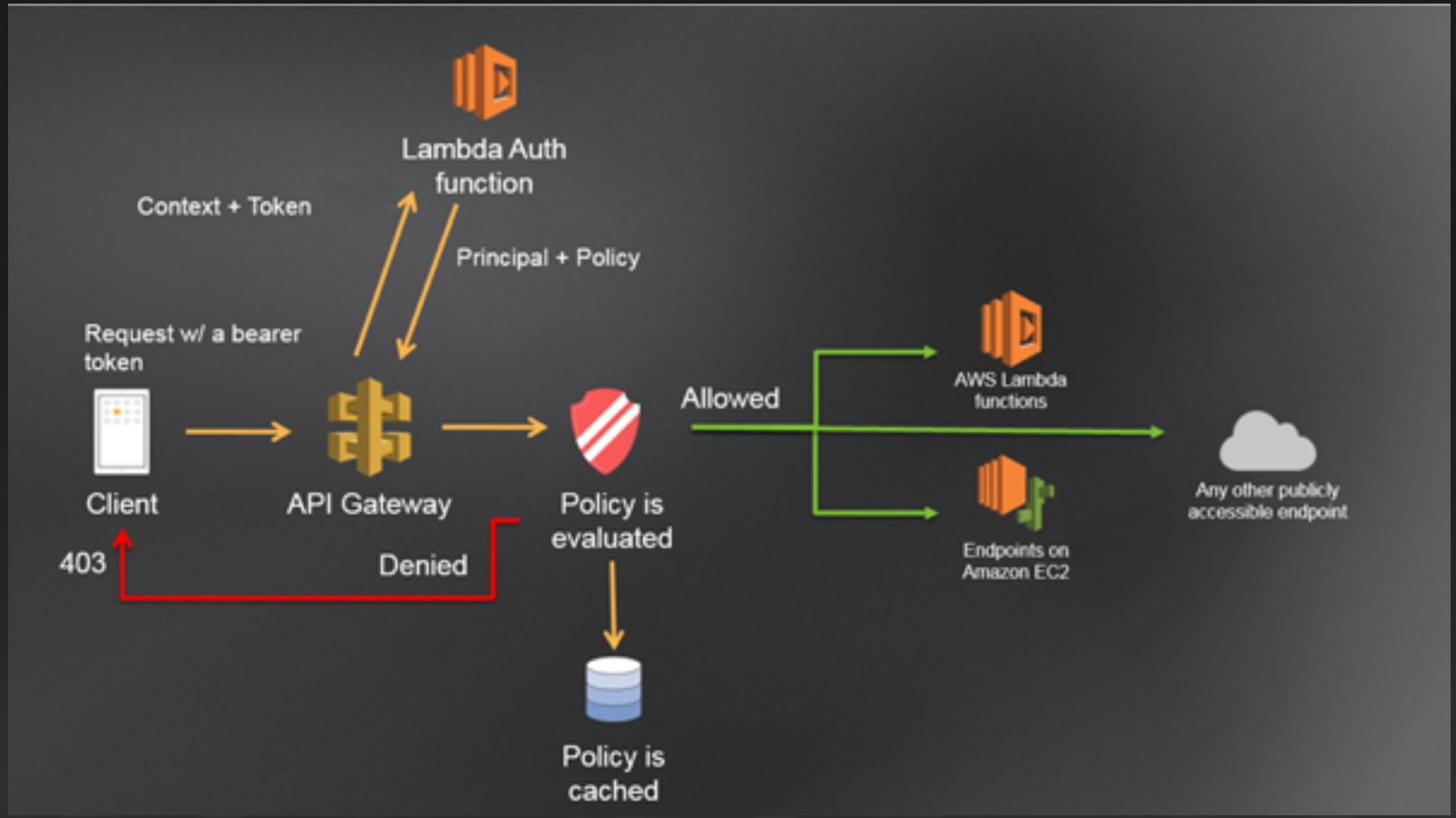
How Work Arrives

- Synchronous calls (clients, APIs, cross-calls)
- Asynchronous calls (example: Amazon S3)
- Polling other services (example: Amazon Kinesis)
- Schedules (“cron”)
- If another service is involved, we call it an “event”
  - Trivial programming model (IJJ – It’s Just JSON)
  - Operational complexity stays in the cloud, where it belongs:
    - S3 and AWS Lambda handle impedance matching behind the scenes

New App Ecosystems:
Alexa Apps + Slack = Serverless Bots!



SaaS and Enterprise Integration



Customer Success Stories

Pay-per Request

- Buy compute time in 100 ms increments
- Low request charge
- No hourly, daily, or monthly minimums
- No per-device fees

Free Tier

1 million requests and 400,000 GBs of compute every month, every customer

Never pay for idle!

AWS Lambda, API Gateway, and AWS IoT Regions

Available regions

Join the serverless revolution!

amazon web services

0. Product manager or business analyst? Check out aws.amazon.com/lambda for scenarios and customer stories.

amazon web services

1. Developer? Go to the AWS Lambda console, create a function, and run it.

(The first million invokes are on us!)

amazon web services

2. Congrats, you're a Lambda function expert! Add an event source or an HTTP endpoint.

amazon web services

3. Build a mobile, voice, or IoT backend with a few lines of code.

amazon web services

4. Consign your devops tools to the dustbin of history.

amazon web services

Thank You

Follow AWS Lambda and
Amazon API Gateway!

aws.amazon.com/blogs/compute
aws.amazon.com/lambda
AWS Lambda forum

t: @timallenwagner

amazon web services