Building serverless web applications

Benjamin Smith - Serverless, AWS
@Benjamin_l_s

May 2020
Why are we here today?
It’s easier than ever to build web applications:

Applications that are:

• Massively scalable
• Highly available and fault tolerant
• Easily securable
• Cost aligned with usage

Without:

• Managing a single server or container orchestration system
• Dealing with operating system management
• Focusing on system related metrics to scale
The end goal

Customer value

Reliability

Resilience

Scalability

And do it faster
Exploring a **traditional** web application technology stack
Exploring a traditional web application technology stack
Brief history of the front end

Rise of the front-end Professional: 2005
Maturity of front-end engineering: 2009
Componentization and the rise of VDOM: 2013
Serverless?: 2020
AWS today - two serverless options to host static sites

- AWS Amplify Console
- Roll your own with Amazon S3

AWS Amplify

Amazon Simple Storage Service (S3)
Cloud storage for the Internet
Object-based storage
11 9s of durability
Virtually limitless storage
Objects up to 5 TB in size
Good for things like the following:
  • Static assets (html, js, images)
  • Backups
  • Logs
  • Customer records
  • Media files
Roll your own with Amazon S3 and Amazon CloudFront
Hosting for static web applications
Built on top of S3
Integrated with Git
Built in CI/CD workflows
  • Supports branches
Integrated with Amazon CloudFront
Support for auth controls
Custom domain support
Rewrites and redirects built in
Integration with popular frontend frameworks such as Vue, Angular, and React, as well as static site generators like Jekyll & Hugo
Exploring a traditional web application technology stack
Exploring a **traditional** web application technology stack
What is serverless?

- No infrastructure provisioning, no management
- Automatic scaling
- Pay for value
- Highly available and secure
AWS Lambda
Lambda Handles Load Balancing
Auto Scaling
Failures
Security Isolation
OS Management
Managing Utilization
(and many other things) for you
Serverless applications
Serverless applications

Function

Node.js
Python
Java
C#
Go
Ruby
Runtime API
Serverless applications

Event source

Function

- Changes in data state
- Requests to endpoints
- Changes in resource state

Node.js
Python
Java
C#
Go
Ruby
Runtime API
Serverless applications

Event source

Changes in data state
Requests to endpoints
Changes in resource state

Function

Node.js
Python
Java
C#
Go
Ruby
Runtime API

Services (anything)
Lambda execution model

Asynchronous (event)

Amazon API Gateway

/order

Lambda function

Synchronous (push)

Amazon SNS

Amazon S3

reqs

Lambda function

Stream (Poll-based)

Amazon DynamoDB

Amazon S3

changes

AWS Lambda service

Lambda function

Lambda function
Fine-grained pricing

- Buy compute time in 100ms increments
- No hourly, daily, or monthly minimums
- No per-device fees
- Never pay for idle

Free Tier
1M requests and 400,000 GBs of compute.
Every month, every customer.
Tweak your function’s compute power

Lambda exposes only a memory control, with the % of CPU core and network capacity allocated to a function proportionally.

Is your code CPU, Network or memory-bound? If so, it could be cheaper to choose more memory.
Exploring a **traditional** web application technology stack
APIs are the front door to your backend services
Amazon API Gateway

Create a unified API frontend for multiple microservices

DDoS protection and throttling for your backend

Authenticate and authorize requests to a backend

Throttle, meter, and monetize API usage by third-party developers
Exploring a **traditional** web application technology stack.
Exploring a traditional web application technology stack

Mobile/Web apps → Internet → AWS Amplify → Amazon API Gateway → AWS Lambda → Database
Databases

@AWS
# Common data categories and use cases

<table>
<thead>
<tr>
<th>Data Category</th>
<th>Description</th>
<th>Use Cases</th>
<th>AWS Service(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>Referential integrity, ACID transactions, schema-on-write</td>
<td>Lift and shift, ERP, CRM, finance</td>
<td>Aurora, RDS</td>
</tr>
<tr>
<td>Key-value</td>
<td>High throughput, low-latency reads and writes, endless scale</td>
<td>Real-time bidding, shopping cart, social, product catalog, customer preferences</td>
<td>DynamoDB</td>
</tr>
<tr>
<td>Document</td>
<td>Store documents and quickly access querying on any attribute</td>
<td>Content management, personalization, mobile</td>
<td>DocumentDB</td>
</tr>
<tr>
<td>In-memory</td>
<td>Query by key with microsecond latency</td>
<td>Leaderboards, real-time analytics, caching</td>
<td>ElastiCache</td>
</tr>
<tr>
<td>Graph</td>
<td>Quickly and easily create and navigate relationships between data</td>
<td>Fraud detection, social networking, recommendation engine</td>
<td>Neptune</td>
</tr>
<tr>
<td>Time-series</td>
<td>Collect, store, and process data sequenced by time</td>
<td>IoT applications, event tracking</td>
<td>Timestream</td>
</tr>
<tr>
<td>Ledger</td>
<td>Complete, immutable, and verifiable history of all changes to application data</td>
<td>Systems of record, supply chain, health care, registrations, financial</td>
<td>QLDB</td>
</tr>
</tbody>
</table>
Relational

Referential integrity, ACID transactions, schema-on-write

Key-value

High throughput, low-latency reads and writes, endless scale

Common Use Cases

Lift and shift, ERP, CRM, finance

Real-time bidding, shopping cart, social, product catalog, customer preferences

Most common use-cases

Aurora, RDS

DynamoDB
# Amazon DynamoDB

Fast and flexible NoSQL database service for any scale

<table>
<thead>
<tr>
<th>Performance at scale</th>
<th>Serverless</th>
<th>Enterprise ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>Millions of requests / second</td>
<td>Maintenance free</td>
<td>ACID transactions</td>
</tr>
<tr>
<td>Delivers microsecond latency</td>
<td>Auto scaling</td>
<td>Encryption at rest</td>
</tr>
<tr>
<td>Automated global replication</td>
<td>On-demand capacity mode</td>
<td>On-demand backup and restore</td>
</tr>
</tbody>
</table>

© 2020, Amazon Web Services, Inc. or its Affiliates.
Exploring a traditional web application technology stack
Another way to draw it
Exploring a **serverless** web application technology stack

**AWS Cloud**

- **Access**
  - Client/browser
  - Amazon API Gateway
- **Compute**
  - AWS Lambda
- **Storage**
  - Amazon DynamoDB

Front end application makes requests to a resource generated by API Gateway
Exploring a serverless web application technology stack

Amazon API Gateway invokes Lambda function
Exploring a serverless web application technology stack

- Client/browser
- Amazon API Gateway
- AWS Lambda
- Amazon DynamoDB

AWS Lambda processes request, according to business logic
Exploring a serverless web application technology stack

Access

AWS Cloud

Compute

Amazon API Gateway

AWS Lambda

Storage

Amazon DynamoDB

Data persisted to Amazon DynamoDB

© 2020, Amazon Web Services, Inc. or its Affiliates.
Exploring a **serverless** web application technology stack

All serverless services scale automatically as needed to handle client requests.
Exploring a **serverless** web application technology stack

**Access**
- AWS Cloud
- Amazon API Gateway
- AWS Amplify Console

**Compute**
- AWS Lambda

**Storage**
- Amazon DynamoDB

Frontend hosted serverlessly and scales to meet load
Exploring a serverless web application technology stack

Access
- AWS Cloud
- Auth0
- AWS Amplify Console

Compute
- Amazon API Gateway
- AWS Lambda

Storage
- Amazon DynamoDB

Authenticate API requests with Amazon Cognito or Auth0
Exploring a serverless web application technology stack

Multiple Lambda functions to handle different requests

© 2020, Amazon Web Services, Inc. or its Affiliates.
Exploring a serverless web application technology stack

Access
- AWS Cloud
- Auth0
- Amazon API Gateway
- AWS Amplify Console

Compute
- [POST] /SignUrl
- Custom Authorizer
- [GET] /Activity
- getSIGNEDUrl
- S3
- getActivitiesForUser

Storage
- S3Bucket
- FreshTracks
- Amazon DynamoDB

Client/browser

S3 for static file storage
But there’s more…
The microservices “iceberg”

Common question: “Should every service of mine talk to another using an API?”

Maybe not!: Most microservices are internal only for a given product supporting their customer facing features. They may only need to pass messages to each other that are simple events and not need a full fledged interactive API.
Focusing below the water line

Public interface

Internal services
Microservice Patterns
WHAT IT IS
Simple, flexible, fully managed publish/subscribe messaging and mobile push notification service for high throughput, highly reliable message delivery

USE CASE
- Notify multiple subscribed applications,
- Replicate data across regions,
- Invoke multiple steps in workloads,
- Parallel processing,
- Trigger serverless actions

COOL CAPABILITIES
Highly reliable delivery of any volume of messages to any number of recipients across multiple protocols
Amazon SNS + Lambda

1. Data published to a topic

2. Lambda function(s) invoked

Messages are published to a Topic

Topics can have multiple subscribers (fanout)

Messages can be filtered and only sent to certain subscribers

Asynchronous
WHAT IT IS
Simple, flexible, fully managed message queuing service for reliably and continuously exchanging any volume of messages from anywhere

USE CASE
Build decoupled, highly scalable microservices, distributed systems, and serverless applications in the cloud

COOL CAPABILITIES
Nearly infinite scalability without pre-provisioning capacity

SQS is #1 event-trigger for Lambda
Amazon SQS + Lambda

1. Message inserted into a queue

2. Lambda function invoked

3. SQS removes message from queue on successful response from function

Processed in batches

Standard queue = at least once delivery

FIFO queue = ordered and exactly once

Visibility timeout allows for handling of failures during processing

Asynchronous
Introducing Events
Events may be observable, not directed

Directed Commands

Please create an invoice.

Ok, here it is

Observable Events

Customer X just ordered a widget

I'll add that to the sales report

I'll send an invoice
Amazon EventBridge

A serverless event bus service for SaaS and AWS services

- Fully managed, pay-as-you-go
- Native integration with SaaS providers
- 17 target services
- Easily build event-driven architectures
WHAT IT IS
Simple, flexible, fully managed, pay-as-you-go event bus service to ingest and process data from AWS services, your own applications, and SaaS applications.

USE CASE
Take action on SaaS messages, run workflows, apply intelligence, audit and analyze, and synchronize data.

COOL CAPABILITIES
17 Targets including Lambda, SQS, SNS and Kinesis

Schema Registry for developers to SEARCH/FIND/TRACK schemas that are used for applications
Amazon EventBridge + Lambda

1. Events published to an event bus

2. Rules & filters route to destination(s)

3. Lambda function(s) invoked

Events are published to an event bus

Set up rules to filter metadata and payload, and route events to targets

Asynchronous
Exploring a serverless web application technology stack

Access

AWS Cloud

Auth0

Amazon API Gateway

AWS Amplify Console

Compute

Custom Authorizer

[POST] /SignUrl

getSignedUrl

S3

[GET] /Activity

getActivity

[GET] /Activities

getActivitiesForUser

Storage

Upload .gpx File with signed URL

FreshTracks S3Bucket

Amazon DynamoDB

Client/browser

Exploring a serverless web application technology stack.

Upload .gpx File with signed URL
Exploring a serverless web application technology stack

Client/browser

Amazon API Gateway

AWS Amplify Console

AWS Cloud

Access

Compute

Storage

Messaging services for data exchange

[GET] /Activities

getActivitiesForUser

[GET] /Activity

getActivity

[POST] /SignUrl

Custom Authorizer

Upload .gpx File with signed URL

FreshTracks S3Bucket

Amazon DynamoDB

Amazon EventBridge

AWS IoT Core

© 2020, Amazon Web Services, Inc. or its Affiliates.
Orchestration
Serverless Orchestration
WHAT IT IS
Simple, flexible, fully managed, pay-as-you-go event bus service to ingest and process data from AWS services, your own applications, and SaaS applications.

USE CASE
serverless Application orchestration with parallel processing and error handling, Automate tasks with built in retries and human approval steps, Data processing with resilient long running workloads.

COOL CAPABILITIES
Express Workflows for high throughput workloads, Nested workflows and Dynamic Parallelism.
AWS Step Functions: Visual Workflows

Define in JSON

Visualize in the Console

Monitor Executions

```
...:
"Comment": "An AML example using a choice state.",
"StartAt": "FirstState",
"States": {
  "FirstState": {
    "Type": "Task",
    "Next": "ChoiceState"
  }
}
...;
```
Exploring a serverless web application technology stack

Access
- AWS Cloud
- Auth0
- Amazon API Gateway
- AWS Amplify Console

Compute
- Custom Authorizer
- AWS Amplify Console
- getSignedUrl S3
- getActivity
- getActivitiesForUser

Storage
- FreshTracks S3Bucket
- Amazon DynamoDB

Messaging
- Amazon EventBridge
- AWS IoT Core

Client/browser
- [POST] /SignUrl
- [GET] /Activity
- [GET] /Activities
- [Message] Workflow Complete

Upload .gpx File with signed URL

© 2020, Amazon Web Services, Inc. or its Affiliates.
Exploring a **serverless** web application technology stack.
Upload .gpx File with signed URL

Custom Authorizer

Upload .gpx File with signed URL

Client/browser

Amazon API Gateway

AWS Amplify Console

Access

Compute

Storage

Messaging

Orchestration

[GET] /Activities

getActivitiesForUser

[GET] /Activity

getActivity

[POST] /SignUrl

getSignedUrl S3

[GET] /Activity

getActivity

FreshTracks S3Bucket

Amazon EventBridge

Amazon DynamoDB

[Message] Workflow Complete

AWS Step Functions Express workflow

Process GPX File

Save meta to DB

Publish to IOT

AWS IoT Core

© 2020, Amazon Web Services, Inc. or its Affiliates.
Demo: https://myFreshTracks.com

A new sample application that demonstrates how to connect to multiple SaaS providers, eliminate the data sprawl that occurs when using multiple SaaS integrations and manage workloads between them.

Activities
- Total Distance: 20.28 miles
- Total Elevation: 1950 M
- Average Speed: 16.39 M/h
- Moving Time: 2:07:48 M

Activities
<table>
<thead>
<tr>
<th>Date</th>
<th>Creator</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-01-31</td>
<td>John Doe</td>
<td>549792</td>
</tr>
<tr>
<td>2020-02-07</td>
<td>Jane Smith</td>
<td>638796</td>
</tr>
</tbody>
</table>

To automate these SaaS integration partners, use AWS services like AWS Step Functions, Amazon EventBridge, and AWS Lambda.

© 2020, Amazon Web Services, Inc. or its Affiliates.
How do you expand beyond your own systems?
Over 24 Amazon EventBridge SaaS Integrations…
Getting Started
Start with a framework

AWS:
- AWS SAM
- AWS Amplify
- AWS Chalice

Third-party:
- APEX
- architect
- Claudia.js
- Serverless Framework
- Zappa
Serverless Application Model
SAM Templates

- Can mix in other **CloudFormation resources**
  - i.e. Amazon S3, Amazon Kinesis, AWS Step Functions
- Supports use of **Parameters, Mappings, Outputs**, **Intrinsic Functions**
  - ie: Ref, Sub, Join, Select, Split
- YAML or JSON
SAM Templates

AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetProductsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.getProducts
      Runtime: nodejs10.x
      CodeUri: src/
      Policies:
        - DynamoDBReadPolicy:
          TableName: !Ref ProductTable
    Events:
      GetResource:
        Type: Api
        Properties:
          Path: /products/{productId}
          Method: get
  ProductTable:
    Type: AWS::Serverless::SimpleTable

In just 20 lines:
  • Lambda function
  • IAM role
  • API Gateway
  • DynamoDB table
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetProductsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.getProducts
      Runtime: nodejs10.x
      CodeUri: src/
      Policies:
        - DynamoDBReadPolicy:
            TableName: !Ref ProductTable
  Events:
    GetResource:
      Type: Api
      Properties:
        Path: /products/{productId}
        Method: get
  ProductTable:
    Type: AWS::Serverless::SimpleTable
Resources

http://MyfreshTracks.com
Github: bls20AWS
Twitter: @Benjamin_l_s
Q&A
Benjamin Smith
Thank you!