Serverless Java Web-apps Running on AWS

because the world can always use another photo sharing site

What is “Serverless”?

An approach to application development that focuses on the application (its code and configuration) and not on the hardware/software needed to run that app or scale it to meet demand.

Yes, this looks a lot like shared hosting

The promise: easy scalability, modularization, offloading of ops

The reality: you give up control, and you still have to monitor your apps

The benefits outweigh the drawbacks for some applications

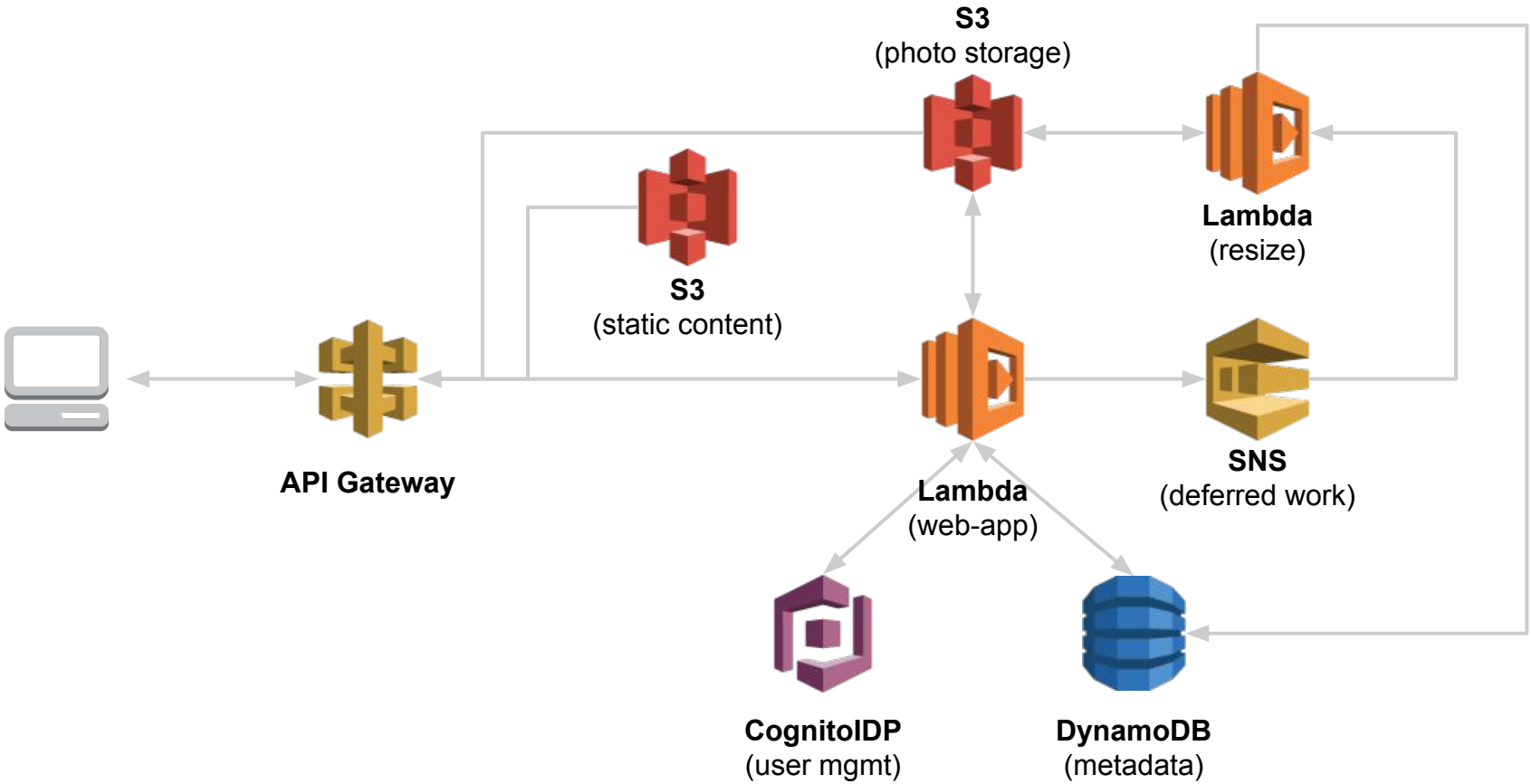
Demo

A photo sharing site: users can create accounts, upload photos, share those photos with other users

For every photo uploaded, we produce various scaled versions that the users can then attach to their own web pages

The system presented here is intended to highlight component features and how to use Java in this environment; it's not intended to show best practices for an AWS deployment

System Architecture



API Gateway

Exposes HTTPS endpoints to the world

- Each API defines a tree of endpoints/requests
- Each endpoint/request combination can interact with a different destination

Can act as front-end for custom HTTP service, AWS service (eg, S3), or Lambda

Applies transformation to request/response before passing it on

- Direct Integration: you define Velocity templates that perform transforms
- Proxy Integration: requests and responses follow a standard form

Beware: API Gateway uses JSON, but HTTP requests/responses don’t map 1:1

Swagger

http://swagger.io/

An API specification language expressed in JSON

The easiest way to handle production API Gateway configuration

- Don’t even try explicit CloudFormation resource configuration!

Start with manual configuration and export the Swagger template

Beware: configuration contains hardcoded paths and ARNs

Serving Static Content

API Gateway supports endpoints that retrieve from S3

- Endpoints can specify explicit path or all files in folder
- Can not specify individual HTML files (dots not allowed in endpoint)
- Useful hack: root resource redirect

Can enable CloudFront caching, for a fee

- Caching controlled on the deployment stage, can be overridden for specific endpoints
- Alternative: an explicit `Cache-Control` header

Better: use CloudFront to serve static assets directly

Lambda

- Independent “functions” — actually, programs
  - Java, JavaScript (node.js), C#, and Python
  - Java deployed as a “shaded” JAR, can include third-party libraries
- Runs in a container
  - Container is started when function first triggered; beware JVM startup time
  - Container remains running for some amount of time after call finishes
- Charged by the “Gigabyte-second” + #/requests
  - Everlasting free tier is 1MM requests, 400,000 GB-seconds per month

Lambda Calling Conventions

Function accepts and returns JSON object

- Lambda will transform to POJO if required, `Map<String,Object>` is generally easier

Functions may be synchronous or asynchronous

- Synchronous functions expected to return a response (eg: HTTP service)
- Asynchronous functions triggered by event (eg: file added to S3)

Uncaught exceptions transformed into JSON

- Contains `errorType`, `errorMessage`, `stackTrace`

Sample Lambda Request (proxy integration)


```
{
    "resource": "/api/{action}",
    "path": "/api/signin",
    "httpMethod": "POST",
    "headers": {
        "Accept": "application/json, text/plain, */*",
        …
    },
    "queryStringParameters": null,
    "pathParameters": {
        "action": "signin"
    },
    "stageVariables": null,
    "requestContext": {
        "accountId": "999999999999",
        …
    },
    "body": "{\"email\":\"test@mailinator.com\",\"password\":\"Test1234\"}",
    "isBase64Encoded": false
}
```

What This Means for a Web-app

No javax.servlet.http

Use “Front Controller” pattern

- Single entry point necessitates a dispatcher that examines request and calls service

Focus on request and response payloads

- Headers can be a challenge: you need to construct/decompose repeated headers

You’re responsible for managing session data

- Must be external to Lambda (eg: Redis, Amazon ElastiCache)

Using AWS SDK

Each service has its own client, which requires authentication

Lots of options in package `com.amazonaws.auth`

Don't be tempted to create your own `AWSCredentials`!

Instead, use `DefaultAWSCredentialsProviderChain`, which looks for credentials in standard locations:

- Environment (`AWS_ACCESS_KEY_ID`/`AWS_SECRET_ACCESS_KEY`)
- Java System Properties
- EC2/Lambda role

Default client constructors use default provider chain

Build and Deploy

Deployment unit is a “shaded” JAR containing all dependencies

- Don’t include the entire AWS SDK!

Can upload manually, better is to retrieve from S3

CloudFormation is almost but not quite capable of managing deployments

- Does not support all features of API Gateway -- need to use Swagger
- But it does finally support Cognito!

Beware: Swagger definitions use hardcoded resource references

Configuration

Simplest: environment variables

- Associated with Lambda function version
- Multiple layers of encryption (by default, stored encrypted but accessible via console)

Alternative: API Gateway stage variables

- Provided with each request
- Questionable use case: different databases for dev/test/prod
- Better use case: feature toggles

Logging

Each Lambda function has its own Cloudwatch log group

- Writes StdOut/StdErr; each line is a separate logging event
- Better: use Log4J with the LambdaAppender
- Alternative: use an appender that routes logs somewhere else (eg: Loggly, ElasticSearch)

API Gateway can also log to Cloudwatch

- Requires explicit role
- If role exists, API Gateway will create log streams even if logging turned off
- Lots of streams!

Testing

Most tests are integration tests

- If you try to mock-out cloud services, you will just be testing your own misconceptions about how they work
- Focus on end-to-end tests

Remember that distributed services introduce an element of time

Curl is your new best friend

Useful Development Habits

Start with a “Hello World” Lambda function to validate your calls

- This function should log its inputs, produce mock outputs

Smoketest API Gateway and Lambda allow from the AWS console

- This is particularly useful to work kinks out of parameters / headers

Domain Names

API Gateway is assigned a domain name by AWS

How to use your own domain:

- Elastic Load Balancer
- CloudFront
- Self-managed reverse proxy

Since API endpoints are served via HTTPS, you can’t simply use a CNAME

Operational Limits

API Gateway

- Max endpoints per API: 300
- Max requests/second (account-wide): 1,000

Lambda:

- Max request size: 6 MB
- Max runtime: 300 seconds
- Max concurrent executions: 100

Things to Know

API Gateway exposes its endpoints to the open Internet

- Can’t run in a VPC or control endpoint name
- Rely on security by obscurity using ELB/Cloudfront

API Gateway execution roles apply to entire service

- May be able to add conditions to trust relationships?

Do You Want to Do This?

It’s too soon to convert your customer-facing applications to Lambda

- JVM startup time will cause more pain than scalability solves
- Perhaps someday Amazon will offer reserved capacity

But worth considering for limited-scope out-of-band operations

- Respond to a Cloudwatch alarm by enabling a website banner warning of degraded performance
- Offload generation of emails or downloadable documents
- Log aggregation (eg: statistical breakdown of ELB access logs)

Not managing your own machines may be enough of a win

http://www.kdgregory.com

https://github.com/kdgregory/example-lambda-java