Making the Switch to “Serverless” Full-Stack Development

David Aktary
Founder, AktaryTech
Agenda

• Introduction
• “Serverless” Overview
• What it is
• Benefits
• Platforms & Tools
Agenda

• Serverless in Practice
• Architecture
• Greenfield
• Migrations
• Impact on Workflow
• Pitfalls and Gotchas
• Workshop
Caveats

- Rapidly changing
- AWS more mature
- Not exhaustive
- Node.js-focused
- Developer-centric
Introduction

• BS Computer Science, Michigan
• MBA, Duke
• 8 Years at IBM
• 3 Years VP Tech @ Nestle Digital Agency
• Founder of AktaryTech
  • Custom Dev Shop
  • Full-Stack JS Specialists
  • LA & Austin
“Serverless” Overview
What it is

The term “serverless”
• BaaS: pre-built, 3rd-party application service
• FaaS: custom application

Similarities:
• Pay-as-you-execute
• Managed server
• Auto-scaling
What it is

Function (“Action”)
- Containerized custom-written application code
- Should include bundled dependencies & binaries
- Memory & execution time limits

Triggers (“Events”)
- Launches Function execution
- Can be another Function

Resources
- External PaaS/BaaS/FaaS services
What it is

N-Tier

- Traditional
  - Client (browser) → Pet Store Server → Database

- Message-driven
  - Ad Server → Click Message Channel → Click Processor → Database

Serverless

- Client (browser) → API Gateway → Authentication Service
- Product Database
- Purchase Function
- Search Function
- Purchase Database

Images from https://martinfowler.com/articles/serverless.html#ACoupleOfExamples
Benefits
Benefits

No servers to provision or manage

Scales with usage

Never pay for idle

Availability and fault tolerance built in

Source: https://www.slideshare.net/AmazonWebServices/building-a-development-workflow-for-serverless-applications-march-2017-aws-online-tech-talks
Benefits: Cost

Bustle Case study*:

• “With AWS Lambda, the engineering team now puts **zero thought into scaling** applications. There is an extremely low cost for any engineer to deploy production-ready code that will scale. **No one has to deal with infrastructure management**, so every engineer can focus on building out new features and innovating.”

• “With no operational maintenance of servers, **the team can remain small, with half the people normally required** to build and operate sites of Bustle’s scale.”

• “Bustle has also experienced approximately **84% cost savings** by moving to a serverless architecture.”

Calculator (beta): [http://serverlesscalc.com](http://serverlesscalc.com)

*Source: https://aws.amazon.com/solutions/case-studies/bustle/*
Benefits: Cost

Lightweight and low-traffic website
• 10,000 hits/day
• perhaps 200 ms of execution time per hit at 256MB
• → 432,000 requests per month and 2160 GB-sec of compute per month
• → about $0.31/mo
... less than 1/10th the cost of even a t2.nano, the smallest EC2 instance!

Periodic Scheduled Job
• A scheduled job that runs every hour with 1GB RAM for 2 minutes
• → 86,400 GB-sec of compute per month and 720 requests per month
• → $1.44/mo
... still less than 1/3rd of a t2.nano!

Source: https://www.trek10.com/blog/lambda-cost/
Benefits: Cost

Lambda Breakeven Analysis for an m4.large Instance
m4.large, 2 vCPU, 8 GB RAM
costs $0.12/hr or ~$86/month in N. Virginia region.

<table>
<thead>
<tr>
<th>Function Execution Memory &amp; Time</th>
<th>Requests per Hour Required for Lambda Cost to Equal EC2 Cost</th>
<th>Requests per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 ms @ 128 MB</td>
<td>295,000</td>
<td>81.9</td>
</tr>
<tr>
<td>200 ms @ 512 MB</td>
<td>64,000</td>
<td>17.8</td>
</tr>
<tr>
<td>200 ms @ 1 GB</td>
<td>34,000</td>
<td>9.4</td>
</tr>
<tr>
<td>1 sec @ 1 GB</td>
<td>7,100</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Source: https://www.trek10.com/blog/lambda-cost/
Benefits: 12-Factor Architecture

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td><strong>Codebase</strong></td>
</tr>
<tr>
<td>II</td>
<td><strong>Dependencies</strong></td>
</tr>
<tr>
<td>III</td>
<td><strong>Configuration</strong></td>
</tr>
<tr>
<td>IV</td>
<td><strong>Backing services</strong></td>
</tr>
<tr>
<td>V</td>
<td><strong>Build, release, run</strong></td>
</tr>
<tr>
<td>VI</td>
<td><strong>Processes</strong></td>
</tr>
<tr>
<td>VII</td>
<td><strong>Port binding</strong></td>
</tr>
<tr>
<td>VIII</td>
<td><strong>Concurrency</strong></td>
</tr>
<tr>
<td>IX</td>
<td><strong>Disposability</strong></td>
</tr>
<tr>
<td>X</td>
<td><strong>Dev/prod parity</strong></td>
</tr>
<tr>
<td>XI</td>
<td><strong>Logs</strong></td>
</tr>
<tr>
<td>XII</td>
<td><strong>Admin processes</strong></td>
</tr>
</tbody>
</table>

Platforms & Tools
Platforms & Tools

Hosted
- Amazon Lambda
- Azure Functions
- Google Cloud Functions
- IBM Bluemix OpenWhisk*
- Auth0 Webtask

On-Prem
- Apache OpenWhisk* (https://github.com/openwhisk)
- IronFunctions (https://github.com/iron-io/functions)
- Functions as a Service (https://github.com/alexellis/faas)
- Funcatron* (https://funcatron.org)

* Claim to minimize vendor lock-in
This list is not exhaustive. Visit https://github.com/anaibol/awesome-serverless for a more detailed list
Platforms & Tools - Rolling Your Own

Virtualization/Orchestration:
- Docker / Docker Swarm
- Vagrant
- Kubernetes
- Mesos

Messaging:
- Kafka
- RabbitMQ

Metrics/Monitoring: Prometheus

API Definition: Swagger
## Platforms & Tools - Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Language</th>
<th>Platform</th>
<th>API Gateway Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node.js</td>
<td>Python</td>
<td>Java</td>
</tr>
<tr>
<td>Serverless</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Claudia</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apex</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Lambda Framework</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zappa</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Gordon</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
SATURN 2017

Serverless in practice
Serverless in Practice - Architecture

Design Patterns

• Event-driven data processing
• Web applications
• Mobile applications
• Internet-of-Things applications
• Real-time stream processing

Source: [http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html](http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html)
Serverless in Practice - Architecture

- Event-driven data processing

Reference architecture: https://github.com/awslabs/lambda-refarch-fileprocessing
Source: http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html
Serverless in Practice - Architecture

- Web applications

Reference architecture: https://github.com/awslabs/lambda-refarch-webapp
Source: http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html
Serverless in Practice - Architecture

• Mobile applications

Reference architecture: https://github.com/awslabs/lambda-refarch-mobilebackend
Source: http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html
Serverless in Practice - Architecture

- Internet-of-Things applications

Reference architecture: [https://github.com/awslabs/lambda-refarch-iotbackend](https://github.com/awslabs/lambda-refarch-iotbackend)

Source: [http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html](http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html)
Serverless in Practice - Architecture

- Real-time stream processing

Reference architecture: https://github.com/awslabs/lambda-refarch-streamprocessing
Source: http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html
Serverless in Practice - Architecture

• Modeling (AWS)
  • Step Functions - model your application as a state machine
  • Cloud Formation - infrastructure as code
  • Serverless Application Model - simplified CF
• Reactive programming: Corenova (https://github.com/corenova/kos)
• Monitoring and Logging: IOpipe (iopipe.com)
  • Function performance metrics
  • Real-time alerts
  • Distributed stack-traces
  • Real-time dashboard
Serverless in Practice - Greenfield

Process
1. Choose language/platform/framework combo
2. Map out microservices
3. Identify Events and Resources
4. Install dependencies/frameworks**
5. Write Tests
6. Write Function(s)
7. Test
8. Deploy
9. Orchestrate*  
10. Rinse, repeat

* Using a tool like AWS Step Functions  
** Native modules must be compiled on and for the native environment (e.g. Amazon Linux)
Serverless in Practice - Migrations

Refactor points:
• Environment variables
• Timeouts (e.g. API Gateway < DB timeout)
• Native modules must be compiled on EC2
• Serve static assets through S3/CloudFront
• Remove session state for scalability (or store session state in DB)
• HTTP requests are not being sent over ports
• Export the Express configuration so the Lambda handler can consume it

Big-bang: https://github.com/awslabs/aws-serverless-express

Impact on Workflow
Impact on Workflow - Checklist

✓ Model your application and infrastructure resources
✓ Configure multiple environments
✓ Establish your testing/validation model
✓ Automate your delivery process

Source: https://www.slideshare.net/AmazonWebServices/building-a-development-workflow-for-serverless-applications-march-2017-aws-online-tech-talks
Impact on Workflow - Testing

Testing of Resources (emulating the platform)

• Localstack (AWS): https://github.com/atlassian/localstack
• Azure CLI: https://www.npmjs.com/package/azure-functions-cli (Win only)
• GCF Emulator: https://cloud.google.com/functions/docs/emulator (Alpha)
• Nock: https://github.com/node-nock/nock (HTTP interceptor for mocking)
• Serverless plugins: https://github.com/serverless/serverless

Best Practice:

• Separate context from business logic
• Emulate/mock resources where you can
• See framework-specific testing details
Impact on Workflow - CI/CD

• Write code
• Commit to source control
• CI server
  • checks out commit
  • runs tests
  • reports findings
• If we’re running a CI/CD setup, then…
• CD server
  • if tests pass, deploys updates
Impact on Workflow - CI/CD

- Write code
- Commit to source control
- CI server
  - checks out commit
  - runs tests
  - reports findings
- If we’re running a CI/CD setup then…
- CD server
  - if tests pass, deploys updates

Events? Routes? Resources?
Impact on Workflow - CI/CD

- AWS
  - CodePipeline
  - CodeCommit
  - CodeBuild
    - Example: https://github.com/awslabs/aws-serverless-samfarm
- GCF, Azure, OpenWhisk: Serverless Framework
- IBM BlueMix: https://www.ibm.com/devops/method/category/tools/ (CD)
- ServerlessCICD.com
SATURN 2017

Pitfalls and Gotchas
Pitfalls and Gotchas

- Stateless server
- Cold starts
- No access to command line, nor sudo/root
- Logging done via external tool
- Language & version lag/limits
- Vendor lock-in
- No clear solution for cross-cloud deployment
Pitfalls and Gotchas - Limitations

- Resource limits
- Concurrent execution*
- Deployment limits

- GCF: [https://cloud.google.com/functions/quotas](https://cloud.google.com/functions/quotas)
- Microsoft: depends on pricing plan

* Can be increased upon request
SATURN 2017
Workshop
Workshop - Overview

Set up “traditional” full-stack app
• Clone back-end
• Update DB URI (and fix `package.json`)
• Start server
• Clone front-end
• Update API URL

“Strangle” migration
• Set up framework
• “Lambdafy” back-end route / function
• Re-direct front-end call
“Traditional” Full-Stack App
Workshop - Set Up Back-End

https://github.com/gothinkster/node-express-realworld-example-app.git

Change database URI in app.js:

```javascript
if(isProduction){
    mongoose.connect(process.env.MONGODDB_URI);
} else {
    mongoose.connect('mongodb://localhost/conduit');
    mongoose.set('debug', true);
}
```

```
npm install

npm run dev
```
Workshop - Set Up Front-End

https://github.com/gothinkster/react-redux-realworld-example-app

Change API_ROOT to http://localhost:3000/api

```javascript
import _superagent from 'superagent';

const superagent = superagentPromise(_superagent, global.Promise);

const API_ROOT = 'http://localhost:3000/api';

const encode = encodeURIComponent;
const responseBody = res => res.body;
```

`npm install`

`npm run start`
“Traditional” Full-Stack App Complete!
Now to “Strangle” It!

Pizza
Cheese, sauce, and dough!

Popular Tags
- anchovies
- cheese
- crust
- mushrooms
- olives
- onions
- pepperoni
- peppers
- sauce

Read more...
"Strangle" Migration

React client

Express API

Tags API

Tags Function

SATURN DB

Webpack Dev Server

Node Express

API Gateway

Lambda Function

MongoDB

localhost

AWS

mLab

SATURN 2017
“Strangle” Migration - Back-End

Create new `<LAMBDA PROJECT>` directory, then…

```
npm install -g serverless
```

Create a new “service” based on the AWS Node template

```
serverless create -t aws-nodejs
```

Uncomment the event in `serverless.yml`, change path, add cors
"Strangle" Migration - Back-End

```javascript
'use strict';

module.exports.hello = (event, context, callback) => {
  var MongoClient = require('mongodb').MongoClient;

  var url = 'mongodb://saturn:saturn2017@ds123381.mlab.com:23381/saturn';

  MongoClient.connect(url, function(err, db) {
    db.collection('articles').distinct('tagList').then(function(tags){
      const response = {
        statusCode: 200,
        headers: {
          "Access-Control-Allow-Origin" : "*" // Required for CORS support to work
        },
        body: JSON.stringify({tags: tags}),
      };
      db.close();
      callback(null, response);
    }).catch(callback);
  });
};
```

- npm init -f
- npm install -S mongodb

In handler.js:
- Add in Mongo connection
- Add in tags DB search
- Create Lambda response

https://pastebin.com/s27L2bbM
“Strangle” Migration - Back-End

Configure AWS CLI


```
aws configure
```

```
serverless deploy
```

Save this!
“Strangle” Migration - Back-End

Great success!

```javascript
const Tags = {
  //getAll: () => requests.get('/tags')
  getAll: () => superagent.get('https://mu8vovubyi.execute-api.us-east-1.amazonaws.com/dev/tags')
    .use(tokenPlugin)
    .then(responseBody)
};
```

Now, just update the front-end to hit Lambda in src/agent.js
“Strangle” Migration Complete!
Thank you!  Questions?