Serverless Computing
Redefining the Cloud
Technology Triggers
1 Edge
Highly Recommended

http://a16z.com/2016/12/16/the-end-of-cloud-computing/

Return to the Edge and the End of Cloud Computing

SPEAKER
Peter Levine, Andreessen Horowitz
Amazon Kinesis Firehose
Load massive volumes of streaming data into S3, Redshift, Elasticsearch,…

• **Zero administration:** Capture and deliver streaming data into Amazon S3, Amazon Redshift, and other destinations **without writing an application or managing infrastructure.**

• **Direct-to-data store integration:** Batch, **compress**, and **encrypt** streaming data for delivery into data destinations **in as little as 60 secs** using simple configurations.

• **Elastic:** Scales to match data throughput w/o intervention

• **Serverless ETL using AWS Lambda** - Firehose can invoke your Lambda function to transform incoming source data.
Amazon Kinesis Analytics

- **Apply SQL on streams**: Easily connect to a Kinesis Stream or Firehose Delivery Stream and apply SQL skills.

- **Build real-time applications**: Perform continual processing on streaming data with sub-second processing latencies using ANSI SQL.

- **Automatics Scalability**: Serverless, elastically scales to match data throughput.

Kinesis Analytics can send processed data to analytics tools so you can create alerts and respond in real-time.
What is serverless?

Build and run applications without thinking about servers
Let’s take a look at the evolution of computing

Physical servers in data centers
Let’s take a look at the evolution of computing
Let's take a look at the evolution of computing.

Physical servers in data centers

Virtual servers in data centers

Virtual servers in the cloud
Each progressive step was better

- Higher utilization
- Faster provisioning speed
- Improved uptime
- Disaster recovery
- Hardware independence
Each progressive step was better

- Higher utilization
- Faster provisioning speed
- Improved uptime
- Disaster recovery
- Hardware independence

- Trade CAPEX for OPEX
- More scale
- Elastic resources
- Faster speed and agility
- Reduced maintenance
- Better availability and fault tolerance
But there are still limitations

- Still need to administer virtual servers
- Still need to manage capacity and utilization
- Still need to size workloads
- Still need to manage availability, fault tolerance
- Still expensive to run intermittent jobs

- Trade CAPEX for OPEX
- More scale
- Elastic resources
- Faster speed and agility
- Reduced maintenance
- Better availability and fault tolerance
Evolving to serverless

Physical servers in data centers

Virtual servers in data centers

Virtual servers in the cloud

SERVERLESS
No server is easier to manage than no server

All of these responsibilities go away

Provisioning and Utilization
EC2 Compute Instance Types

- General purpose: T2, M4, M3
- Compute optimized: C5, C4, C3
- Storage and IO optimized: I3, D2
- Memory optimized: X1, R4, R3
- GPU and FPGA accelerated: F1, P2, G2
<table>
<thead>
<tr>
<th>Model</th>
<th>vCPU</th>
<th>CPU Credits / hour</th>
<th>Mem (GiB)</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2.nano</td>
<td>1</td>
<td>3</td>
<td>0.5</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.micro</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.small</td>
<td>1</td>
<td>12</td>
<td>2</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.medium</td>
<td>2</td>
<td>24</td>
<td>4</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.large</td>
<td>2</td>
<td>36</td>
<td>8</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.xlarge</td>
<td>4</td>
<td>54</td>
<td>16</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>t2.2xlarge</td>
<td>8</td>
<td>81</td>
<td>32</td>
<td>EBS-Only</td>
</tr>
<tr>
<td>Model</td>
<td>vCPU</td>
<td>Mem (GiB)</td>
<td>SSD Storage (GB)</td>
<td>Dedicated EBS Bandwidth (Mbps)</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>m4.large</td>
<td>2</td>
<td>8</td>
<td>EBS-only</td>
<td>450</td>
</tr>
<tr>
<td>m4.xlarge</td>
<td>4</td>
<td>16</td>
<td>EBS-only</td>
<td>750</td>
</tr>
<tr>
<td>m4.2xlarge</td>
<td>8</td>
<td>32</td>
<td>EBS-only</td>
<td>1,000</td>
</tr>
<tr>
<td>m4.4xlarge</td>
<td>16</td>
<td>64</td>
<td>EBS-only</td>
<td>2,000</td>
</tr>
<tr>
<td>m4.10xlarge</td>
<td>40</td>
<td>160</td>
<td>EBS-only</td>
<td>4,000</td>
</tr>
<tr>
<td>m4.16xlarge</td>
<td>64</td>
<td>256</td>
<td>EBS-only</td>
<td>10,000</td>
</tr>
<tr>
<td>Model</td>
<td>vCPU</td>
<td>Mem (GiB)</td>
<td>SSD Storage (GB)</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>-----------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>m3.medium</td>
<td>1</td>
<td>3.75</td>
<td>1 x 4</td>
<td></td>
</tr>
<tr>
<td>m3.large</td>
<td>2</td>
<td>7.5</td>
<td>1 x 32</td>
<td></td>
</tr>
<tr>
<td>m3.xlarge</td>
<td>4</td>
<td>15</td>
<td>2 x 40</td>
<td></td>
</tr>
<tr>
<td>m3.2xlarge</td>
<td>8</td>
<td>30</td>
<td>2 x 80</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>vCPU</td>
<td>Mem (GiB)</td>
<td>Storage</td>
<td>Dedicated EBS Bandwidth (Mbps)</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>c4.large</td>
<td>2</td>
<td>3.75</td>
<td>EBS-Only</td>
<td>500</td>
</tr>
<tr>
<td>c4.xlarge</td>
<td>4</td>
<td>7.5</td>
<td>EBS-Only</td>
<td>750</td>
</tr>
<tr>
<td>c4.2xlarge</td>
<td>8</td>
<td>15</td>
<td>EBS-Only</td>
<td>1,000</td>
</tr>
<tr>
<td>c4.4xlarge</td>
<td>16</td>
<td>30</td>
<td>EBS-Only</td>
<td>2,000</td>
</tr>
<tr>
<td>c4.8xlarge</td>
<td>36</td>
<td>60</td>
<td>EBS-Only</td>
<td>4,000</td>
</tr>
</tbody>
</table>
We Love Ourselves Some Compute

Elastic GPUs On EC2

- Lightsail
- T2
- M4
- D2
- R4
- X1
- I3
- C5
- G2
- P2
- F1

- Simple VPS
- Burstable
- General Purpose
- Dense storage
- Memory intensive
- Large memory
- Compute intensive
- High I/O
- Graphics intensive
- General Purpose GPU
- FPGAs
No server is easier to manage than no server

All of these responsibilities go away

Provisioning and Utilization
Availability and Fault Tolerance
No server is easier to manage than no server

All of these responsibilities go away

Provisioning and Utilization
Availability and Fault Tolerance
Scaling
No server is easier to manage than no server

All of these responsibilities go away

Provisioning and Utilization
Availability and Fault Tolerance
Scaling
Operations and Management
Serverless is a form of event-driven computing
Deliver on demand, never pay for idle
Serverless changes how you deliver

- Speeds up time to market
- Dedicated time to innovation
- Increases developer productivity
- Eliminates operational complexity
# Building blocks for serverless

<table>
<thead>
<tr>
<th>Compute</th>
<th>Storage</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>Amazon S3</td>
<td>Amazon DynamoDB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API Proxy</th>
<th>Messaging and Queues</th>
<th>Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon API Gateway</td>
<td>Amazon SQS</td>
<td>Amazon Kinesis</td>
</tr>
<tr>
<td></td>
<td>Amazon SNS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orchestration and State Management</th>
<th>Monitoring and Debugging</th>
<th>Edge Compute</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Step Functions</td>
<td>AWS X-Ray</td>
<td>AWS Greengrass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lambda@Edge</td>
</tr>
</tbody>
</table>
Occupy the Cloud: Distributed Computing for the 99%

Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht
University of California, Berkeley

Abstract

Distributed computing remains inaccessible to a large number of users, in spite of many open source platforms and extensive commercial offerings. While distributed computation frameworks have moved beyond a simple map-reduce model, many users are still left to struggle with complex cluster management and configuration tools, even for running simple embarrassingly parallel jobs. We argue that stateless functions represent a viable platform for these users, eliminating cluster management overhead, fulfilling the promise of elasticity. Further, learning graduate students have never written a cluster computing job.

In this paper we argue that a serverless execution model with stateless functions can enable radically-simpler, fundamentally elastic, and more user-friendly distributed data processing systems. In this model, we have one simple primitive: users submit stateless functions that are executed in a remote container and inputs, outputs for the function are accessed from shared remote storage. By removing the notion of servers from end users, we can avoid the significant developer and man-
Serverless today
Common Use Cases

**Web Applications**
- Static websites
- Complex web apps
- Packages for Flask and Express

**Backends**
- Apps & services
- Mobile
- IoT

**Big Data**
- Real time
- MapReduce
- Batch

**Chatbots**
- Powering chatbot logic

**Amazon Alexa**
- Powering voice-enabled apps
- Alexa Skills Kit

**IT Automation**
- Policy engines
- Extending AWS services
- Infrastructure management
Common Use Cases

Web Applications
- Static websites
- Complex web apps
- Packages for Flask and Express

Backends
- Apps & services
- Mobile
- IoT

Web Applications and Backends
Common Use Cases

Web Applications
- Static websites
- Complex web apps
- Packages for Flask and Express

Backends
- Apps & services
- Mobile
- IoT

Task
- Extract Image Metadata
- Image Type Check
- Not Supported Image Type

Choice
- Parallel

Lambda + Step Functions
Image Recognition and Processing Backend

Diagram:
- Amazon S3
- AWS Lambda
- AWS Step Functions
- Amazon Rekognition
- Amazon DynamoDB
Serverless IoT with AWS Lambda at iRobot
How iRobot leverages AWS
Serverless is Distributed by Nature

- Component graph becomes call graph
- Distributed systems thinking is required from the start
- Event-based architecture
Common Use Cases

Web Applications
- Static websites
- Complex web apps
- Packages for Flask and Express

Data Processing
- Real time
- MapReduce
- Batch

Chatbots
- Powering chatbot logic

Backends
- Apps & services
- Mobile
- IoT

Amazon Alexa
- Powering voice-enabled apps
- Alexa Skills Kit

IT Automation
- Policy engines
- Extending AWS services
- Infrastructure management
Common Use Cases

Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express

https://github.com/awslabs/lambda-refarch-mapreduce
PyWren: a massive data framework for Lambda

- Open source MapReduce framework using Lambda
- 25 TFLOPS performance
- 60 GB/sec read and 50 GB/sec write to S3

https://github.com/pywren/pywren
http://pywren.io/
http://ericjonas.com/
Now run denser workloads with Lambda

Default concurrency

= 

600 concurrent functions
Common Use Cases

Web Applications
- Static websites
- Complex web apps
- Packages for Flask and Express

Backends
- Apps & services
- Mobile
- IoT

Data Processing
- Real time
- MapReduce
- Batch

Chatbots
- Powering chatbot logic

Lambda + S3
- Example: Image Thumbnail Creation
- Lambda is triggered

Amazon Alexa
- Powering voice-enabled apps
- Alexa Skills Kit
- Lambda runs image resizing code to generate web, mobile, and tablet sizes

IT Automation
- Policy engines
- Extending AWS services
- Infrastructure management
Common Use Cases

Data Processing

• Real time
• MapReduce
• Batch

Lambda + Kinesis + DynamoDB

Example: Analysis of Streaming Social Media Data

KINESIS
Social media stream is loaded into Kinesis in real-time

Lambda is triggered

DYNAMODB
Lambda runs code that generates hashtag trend data and stores it in DynamoDB

Social media trend data immediately available for business users to query
Common Use Cases

Data Processing

- Real time
- MapReduce
- Batch

Lambda + DynamoDB + Redshift

Example: Retail Data Warehouse ETL

1. Online order is placed
2. DYNAMODB: Order data is stored in operational database
3. Lambda is triggered
4. Lambda runs data transformation code and loads results into data warehouse
5. REDSHIFT
6. Analytics generated from data
15-20 minutes of processing → now in seconds
2x order of magnitude for cost savings

https://www.youtube.com/watch?v=TXmkj2a0fRE
Common Use Cases

Web Applications
- Static websites
- Complex web apps
- Packages for Flask and Express

Backends
- Apps & services
- Mobile
- IoT

Data Processing
- Real time
- MapReduce
- Batch

Chatbots
- Kevin says, "Break a leg!"
- Powering chatbot logic

Amazon Alexa
- Powering voice-enabled apps
- Alexa Skills Kit

Slack
- Message upload (via Slack API)
- Message retrieval through scheduled polling

IT Automation
- Policy engines
- Extending AWS services
- Infrastructure management
- Team (channel users)
Common Use Cases

**Web Applications**
- Static websites
- Complex web apps
- Packages for Flask and Express

**Backends**
- Apps & services
- Mobile
- IoT

**Amazon Alexa**
- Powering voice-enabled apps
- Alexa Skills Kit

**IT Automation**
- Policy engines
- Extending AWS services
- Infrastructure management
Serverless is a core component of modern apps
Customers innovating with serverless

Enterprises are achieving massive scale with Lambda

- **Thomson Reuters** processes 4,000 requests per second

- **FINRA** processes half a trillion validations of stock trades daily

- **Hearst** reduced the time to ingest and process data for its analytics pipeline by 97%

- **Vevo** can handle spikes of 80x normal traffic

- **Expedia** triggers 1.2 billion Lambda requests each month
Capabilities of a serverless platform

- Cloud Logic Layer
- Orchestration and State Management
- Responsive Data Sources
- Application Modeling Framework
- Developer Ecosystem
- Integrations Library
- Security and Access Control
- Reliability and Performance
- Global Scale
Dead Letter Queues

- Automatically capture events after exhausting retries
- Build even more reliable event processing applications
- Target Amazon SQS queues or Amazon SNS topics
- Available in all regions
How do you debug distributed applications made of multiple functions or services?

How do you gain insights into how your functions are performing or behaving?
AWS X-Ray

- Analyze and debug distributed apps in production
- Visualize service call graph of your app
- Identify performance bottlenecks and errors
- Pinpoint service-specific issues
- Identify impact of issues on users of the app
- Trace function executions (preview)
```javascript
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
s3 = new AWS.S3({signatureVersion: 'v4'});

exports.handler = (event, context, callback) => {

    var params = {Bucket: 'tim-example-bucket', Key: 'MyKey', Body: 'Hello!'};

    s3.putObject(params, function(err, data) {});
};
```
### X-Ray example

<table>
<thead>
<tr>
<th>Method</th>
<th>Response</th>
<th>Duration</th>
<th>Age</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>202</td>
<td>2.0 sec</td>
<td>1.3 min (2017-04-14 00:42:54 UTC)</td>
<td>1-58f01b0e-53eef2bd463eefcfd7f311ce4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Res.</th>
<th>Duration</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s3example</strong> AWS::Lambda</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3example</td>
<td>202</td>
<td>87.0 ms</td>
<td>✔️</td>
<td>0.0s</td>
</tr>
<tr>
<td>Dwell Time</td>
<td>-</td>
<td>186 ms</td>
<td>✔️</td>
<td>0.0s</td>
</tr>
<tr>
<td>Attempt #1</td>
<td>200</td>
<td>1.8 sec</td>
<td>✔️</td>
<td>0.0s</td>
</tr>
<tr>
<td><strong>s3example</strong> AWS::Lambda::Function</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3example</td>
<td>-</td>
<td>863 ms</td>
<td>✔️</td>
<td>0.0s</td>
</tr>
<tr>
<td>Initialization</td>
<td>-</td>
<td>334 ms</td>
<td>✔️</td>
<td>0.0s</td>
</tr>
<tr>
<td>S3</td>
<td>404</td>
<td>762 ms</td>
<td>🔄</td>
<td>0.0s</td>
</tr>
</tbody>
</table>
X-Ray example

Method: --
Response: 202
Duration: 2.0 sec
Age: 1.3 min (2017-04-14 00:42:54 UTC)
ID: 1-58f01b0e-53eef2bd463eefcfd7f311ce4

<table>
<thead>
<tr>
<th>Name</th>
<th>Res.</th>
<th>Duration</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3example</td>
<td>202</td>
<td>87.0 ms</td>
<td>✔️</td>
</tr>
<tr>
<td>Dwell Time</td>
<td>-</td>
<td>186 ms</td>
<td>✔️</td>
</tr>
<tr>
<td>Attempt #1</td>
<td>200</td>
<td>1.8 sec</td>
<td>✔️</td>
</tr>
<tr>
<td>s3example</td>
<td>AWS::Lambda::Function</td>
<td>863 ms</td>
<td>✔️</td>
</tr>
<tr>
<td>Initialization</td>
<td>-</td>
<td>334 ms</td>
<td>✔️</td>
</tr>
<tr>
<td>S3</td>
<td>404</td>
<td>762 ms</td>
<td>⚠️</td>
</tr>
</tbody>
</table>

Remote fault caused by NoSuchBucket
The specified bucket does not exist. (Click for details)
```javascript
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
s3 = new AWS.S3({signatureVersion: 'v4'});

exports.handler = (event, context, callback) => {
    var params = {Bucket: 'tim-example-blucket', Key: 'MyKey', Body: 'Hello!'};
s3.putObject(params, function(err, data) {});
};
```
```javascript
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
s3 = new AWS.S3({signatureVersion: 'v4'});
exports.handler = (event, context, callback) => {
  var params = {
    Bucket: 'tim-example-bucket',
    Key: 'MyKey',
    Body: 'Hello!'  
  }

  s3.putObject(params, function(err, data) {});
```

X-Ray example
AWS Greengrass (in preview)

- Extends Lambda functions to devices
- Low latency, near-real time
AWS Snowball Edge

• Petabyte-scale hybrid device with onboard compute and storage

• Deploy AWS Lambda code to Snowball Edge
Lambda@Edge is available in all Amazon CloudFront edge locations

- Low-latency request/response customization
- Supports viewer and origin events
Takeaways

Serverless is a Fundamental Component of Modern Applications
• Many enterprise applications can go serverless
• Move to event driven computing

The ecosystem continues to grow
• Tooling, languages, and application capabilities
• But we still have a long ways to go…

Serverless and Edge are technology triggers with the potential to reshape distributed computing and the role of cloud computing