An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions

David Jackson & Gary Clynch
Institute of Technology, Tallaght

4th International Workshop on Serverless Computing (WoSC4)
20th December 2018
Objective:
To understand the impact of the choice of language runtime on the performance and cost of serverless function execution.
Use **empty** test functions to measure platform **startup** performance.

**Scope of Investigation**

**AWS Lambda**
- .NET Core 2 (C#)
- Java 8
- Python 3.6
- NodeJS 6.10
- Go

**Azure Functions**
- .NET C#
- NodeJS 6.11.2

AWS US-East-1 / Azure East US
- Empty Functions
- Cold-Start vs Warm-Start

November 2018

128MB Memory Allocation
Single Function Execution
Serverless Billing Model

- Individual execution cost per function invocation
- Execution duration billed at “GB-second” rate
- Cost rounded at 100ms increments (50ms @ edge)

Example: AWS Lambda, 128MB, 85ms Duration, 100ms “Billed” Duration

$0.21 + $0.20 = $0.41

Execution Time (0.1 s * 0.125 GB * $0.00001667 per function)
Invocation Cost
Total Cost

* 1 million function invocations
Results Summary
AWS Lambda - Warm Start

2.69 ms

.NET Core 2 (Average Performance)

2.70 ms
Python

10.84 ms
GoLang

3.77 ms
Java

4.20 ms
NodeJS
AWS Lambda - Cold Start

2,643ms

.NET Core 2 (Average Performance)

4.84ms Python
6.63ms GoLang
412.89ms Java
31.9ms NodeJS
Azure Functions - Warm Start

0.78ms
.NET C# Script (Average Performance)

1.61ms
NodeJS
Azure Functions - Cold Start

17.08 ms
.NET C# Script (Average Performance)

424.97 ms
NodeJS
Results Analysis - AWS Lambda
AWS Lambda

Warm Start

9,550 Function Invocations Per Runtime

Nov 2018
AWS Lambda

Warm Start

9,550 Function Invocations Per Runtime

Nov 2018
AWS Lambda Cold-Start
500 Function Invocations Per Runtime

Average Execution Time

Cost Per Million ($)

Nov 2018
AWS Lambda
Cold-Start Histogram

500 Function Invocations Per Runtime

Nov 2018
AWS Lambda

Cold Start

Top 3 Performers

500 Function Invocations Per Runtime

Nov 2018
Results Analysis - Azure Functions
Azure Functions
Warm Start

9,550 Function Invocations Per Runtime

Nov 2018
Azure Functions
Cold Start

500 Function Invocations Per Runtime

Nov 2018
Azure Performance
Cold-Start Histogram
500 Function Invocations Per Runtime
Nov 2018
Results Analysis - AWS vs Azure
AWS vs Azure

C# .NET Warm Start

Execution Time (ms)
Cost Analysis
## TPS Cost Calculations

**AWS Lambda**

<table>
<thead>
<tr>
<th>Language Runtime</th>
<th>Cost Per Day @ 100-TPS</th>
<th>Cost Per Day @ 30k-TPS</th>
<th>Cost Per Year @ 100-TPS</th>
<th>Cost Per Year @ 30k-TPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C# .NET</td>
<td>$50.34</td>
<td>$15,101</td>
<td>$18,373</td>
<td>$5,511,980</td>
</tr>
<tr>
<td>Golang</td>
<td>$3.53</td>
<td>$1,059</td>
<td>$1,288</td>
<td>$386,355</td>
</tr>
<tr>
<td>Java 8</td>
<td>$10.73</td>
<td>$3,219</td>
<td>$3,916</td>
<td>$1,174,913</td>
</tr>
<tr>
<td>NodeJS</td>
<td>$3.53</td>
<td>$1,059</td>
<td>$1,288</td>
<td>$386,355</td>
</tr>
<tr>
<td>Python</td>
<td>$3.53</td>
<td>$1,059</td>
<td>$1,288</td>
<td>$386,355</td>
</tr>
</tbody>
</table>

*Figures based on cold-start times to illustrate potential cost impact*
# TPS Cost Calculations

## Azure Functions

<table>
<thead>
<tr>
<th>Language Runtime</th>
<th>Cost Per Day @ 100-TPS</th>
<th>Cost Per Day @ 30k-TPS</th>
<th>Cost Per Year @ 100-TPS</th>
<th>Cost Per Year @ 30k-TPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET C#</td>
<td>$3.46</td>
<td>$1,036.80</td>
<td>$1,261</td>
<td>$378,432</td>
</tr>
<tr>
<td>NodeJS</td>
<td>$10.37</td>
<td>$3,110.40</td>
<td>$3,784</td>
<td>$1,135,296</td>
</tr>
</tbody>
</table>

* Figures based on cold-start times to illustrate potential cost impact
CostHat Model
(Leitner et al. 2016)

Serverless Performance Framework Architecture

30k Function Invocations based on 1,000 TPS
Conclusions & Future Work
Conclusion

Overall Performance

For optimum performance and cost-management of serverless applications, C# .NET is the top performer for Azure Functions. Python is clear overall choice on AWS Lambda.
The performance of NodeJS in Azure Functions in cold-start scenarios demands caution on its usage.

Similarly caution is advised with Java and especially C# .NET on AWS Lambda.
The pace of change in serverless computing is extremely high - in features offered, performance characteristics and cost models.

This constantly shifting environment requires regular review to ensure serverless applications are designed for optimum performance and cost benefit.
The composition of functions in serverless applications is a crucial design decision, which if done in an appropriately fine-grained manner, can lead to a more flexible but also more cost-effective solution in the long term.
Future Work

● Additional Serverless Platform Testing
  ○ Google Cloud Functions
  ○ IBM OpenWhisk
  ○ OpenLambda

● Real-Time Dashboard

● Additional Test Variables
  ○ Regions / Hardware
  ○ Memory Allocations

● Additional Test Scenarios
  ○ DynamoDB Access
  ○ API Access
  ○ Language Performance Benchmarking Tests
Questions?
References