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.

Scope of Investigation

Use empty test functions to measure platform startup performance.

128MB Memory Allocation
Single Function Execution

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

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)

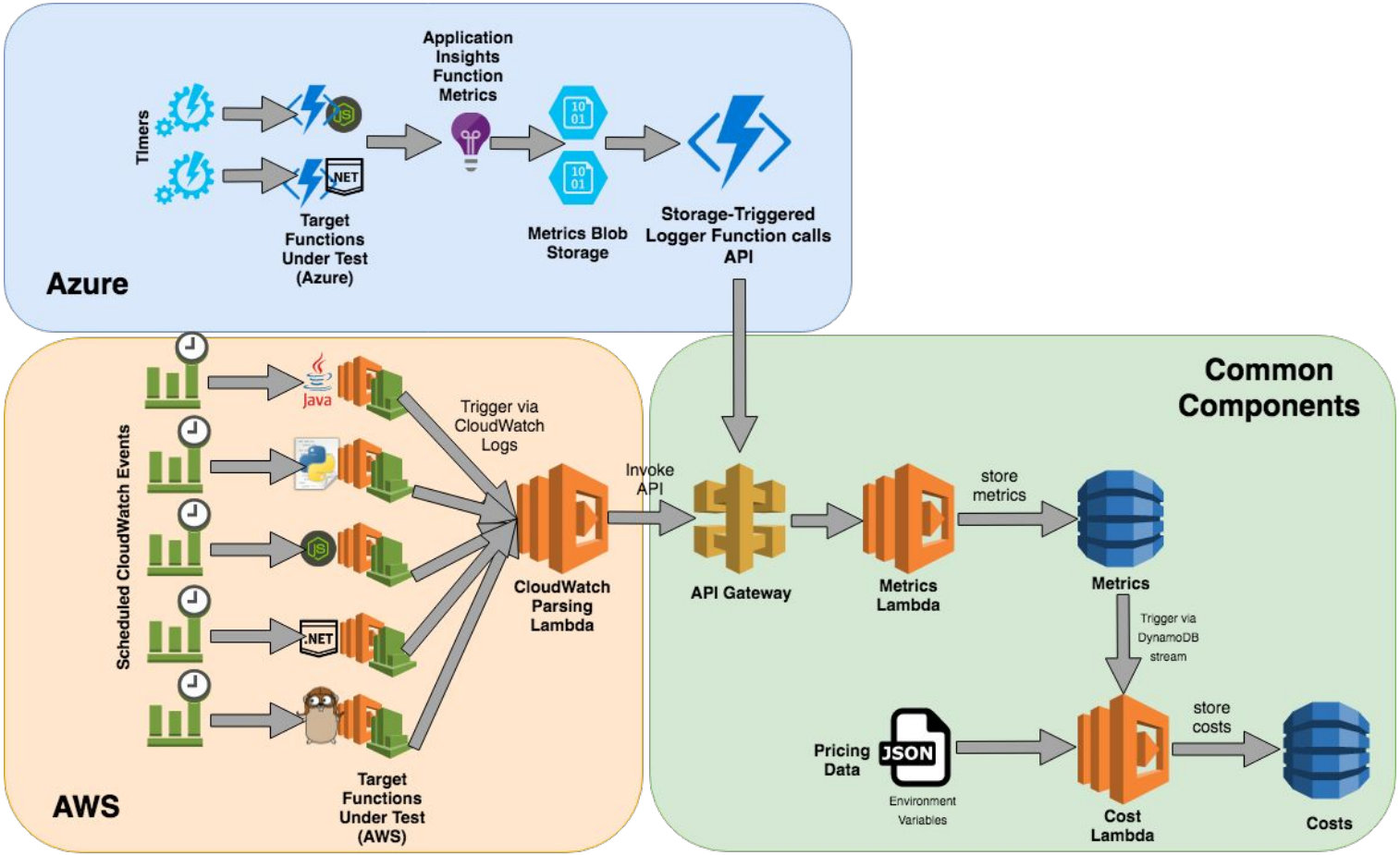
Invocation Cost

Total Cost

* 1 million function invocations

Serverless Architecture

Serverless Performance Framework



Results Summary

AWS Lambda - Warm Start

2.69ms

.NET Core 2 (Average Performance)

2.70ms	10.84ms	3.77ms	4.20ms
Python	GoLang	Java	NodeJS

AWS Lambda - Cold Start

2,643ms

.NET Core 2 (Average Performance)

4.84ms	6.63ms	412.89ms	31.9ms
Python	GoLang	Java	NodeJS

Azure Functions - Warm Start

0.78ms

.NET C# Script (Average Performance)

1.61ms

NodeJS

Azure Functions - Cold Start

17.08ms

.NET C# Script (Average Performance)

424.97ms

NodeJS

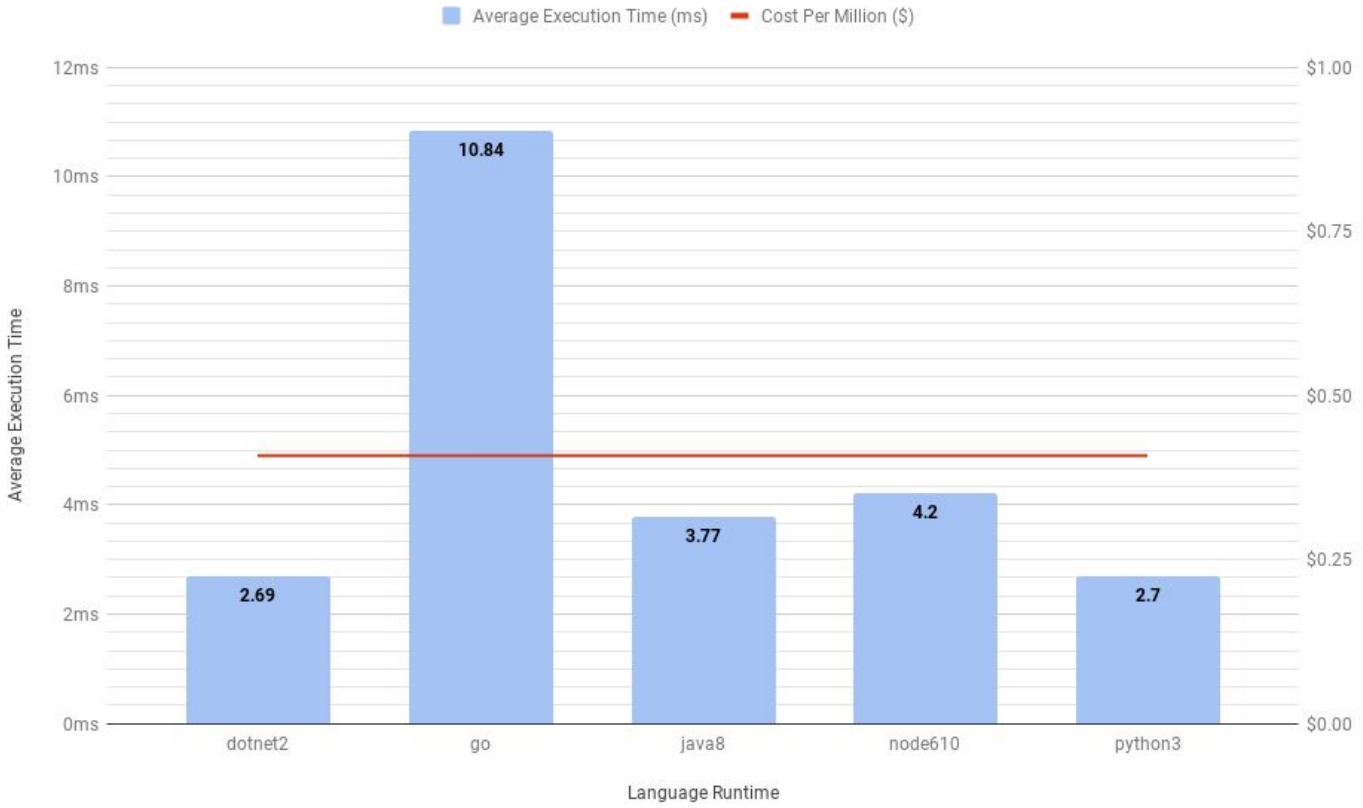
Results Analysis - AWS Lambda

AWS Lambda

Warm Start

9,550 Function Invocations Per Runtime

Nov 2018

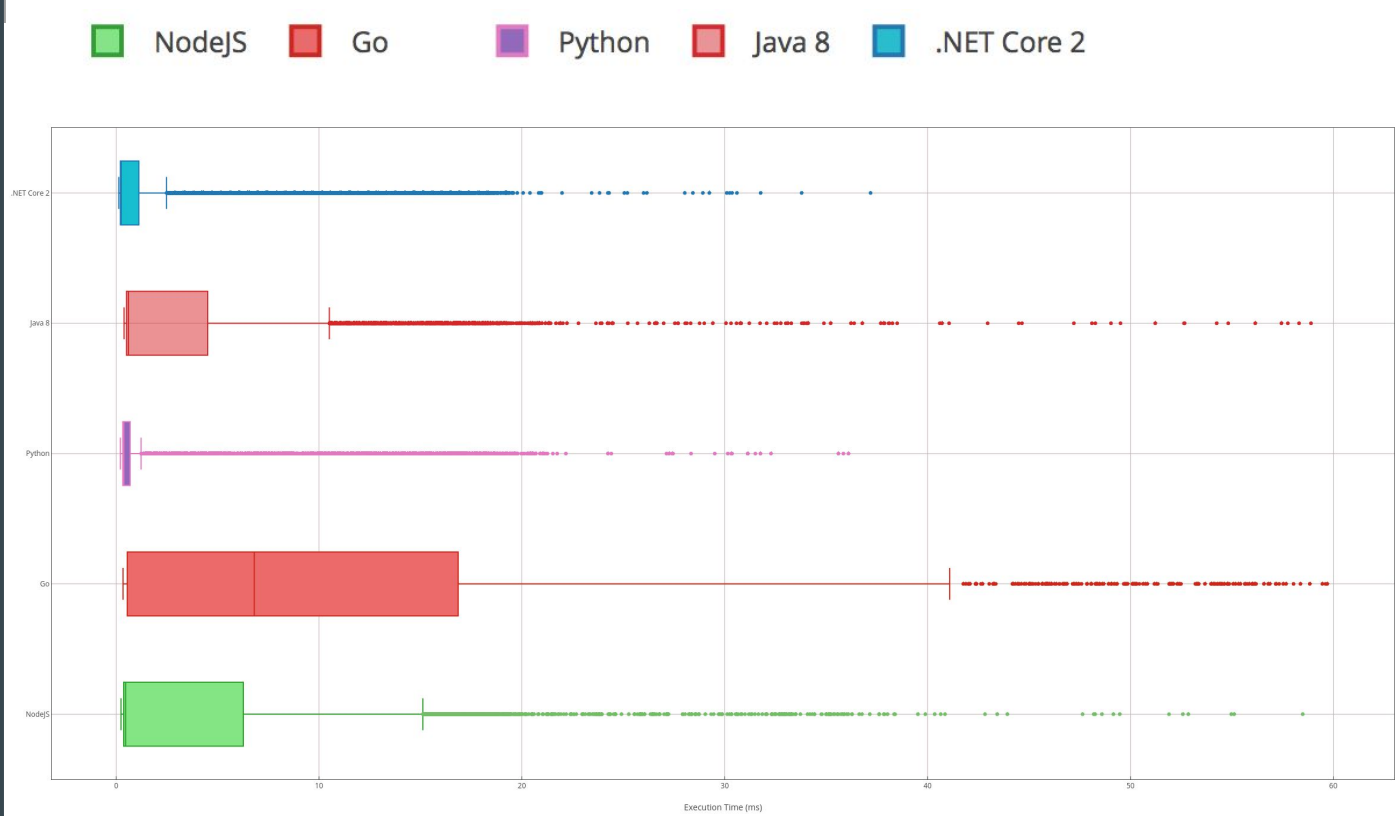


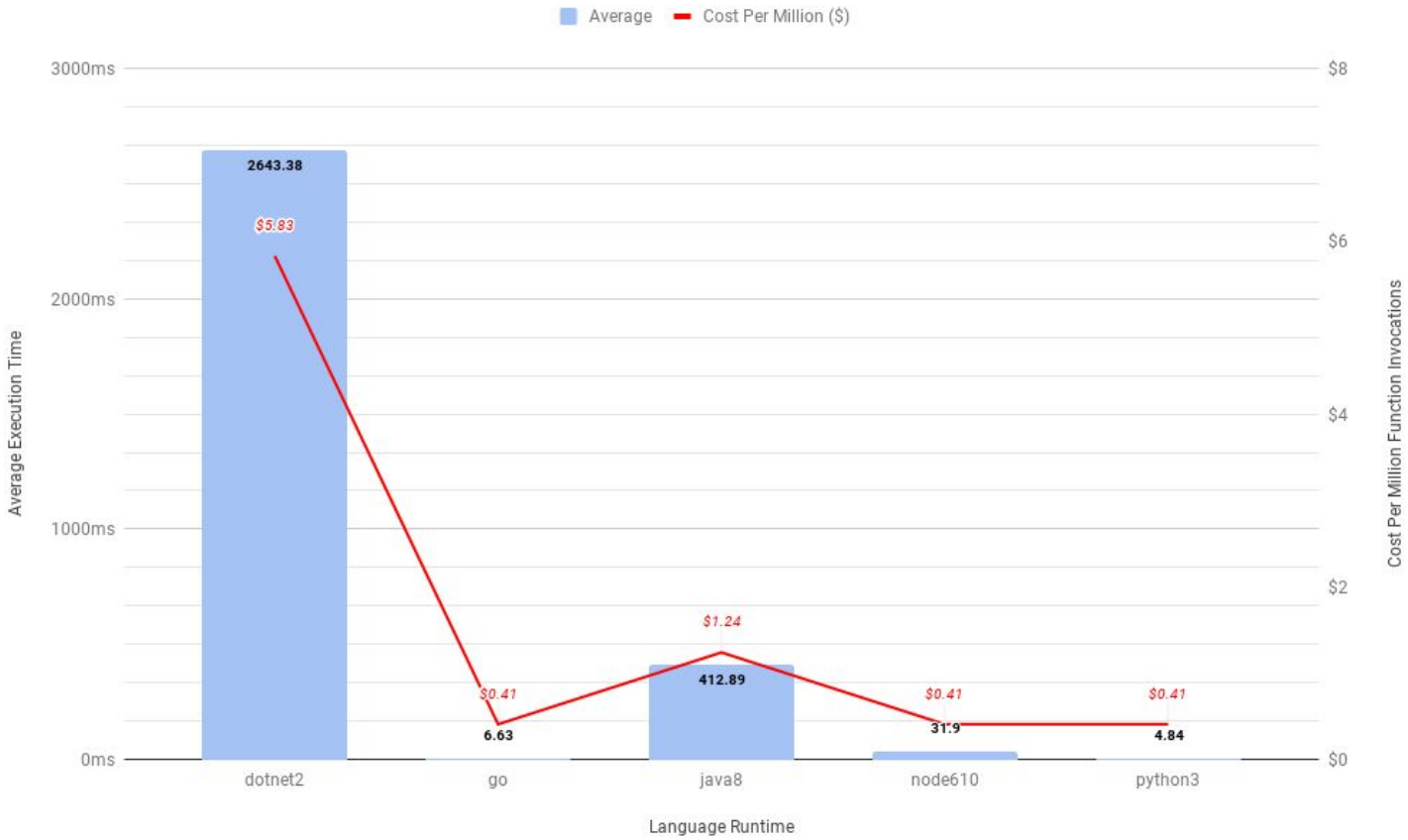
AWS Lambda

Warm Start

9,550 Function Invocations Per Runtime

Nov 2018





AWS Lambda

Cold-Start

500 Function Invocations Per Runtime

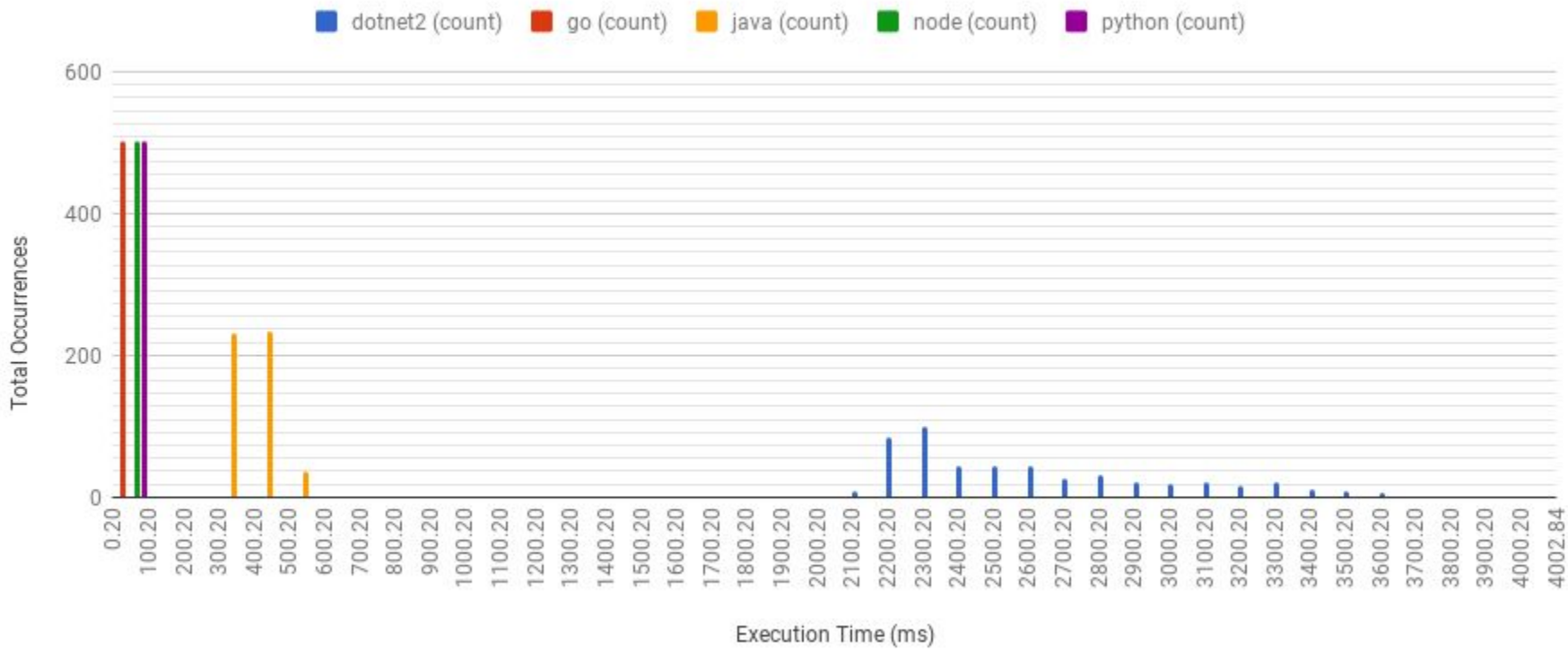
Nov 2018

AWS Lambda

Cold-Start Histogram

Nov 2018

500 Function Invocations Per Runtime

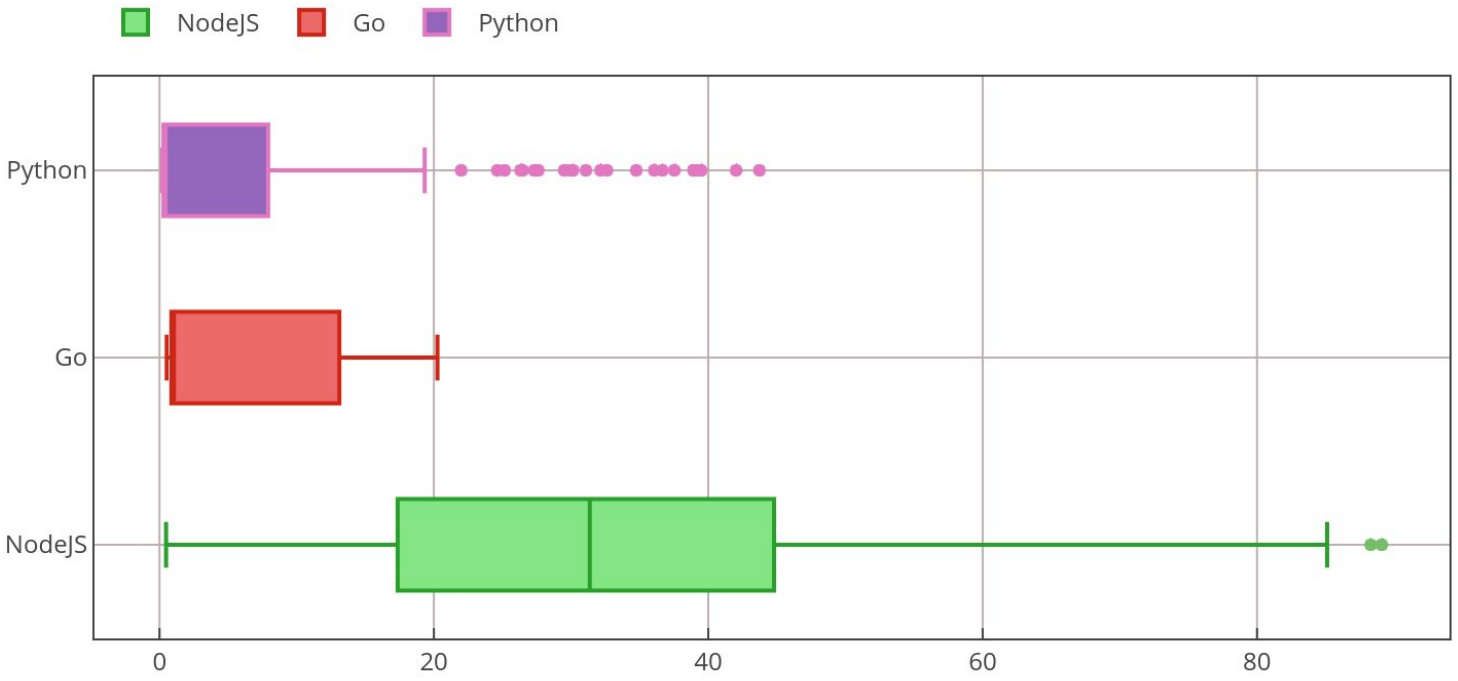


AWS Lambda

Cold Start

Top 3 Performers

500 Function Invocations Per Runtime



Nov 2018

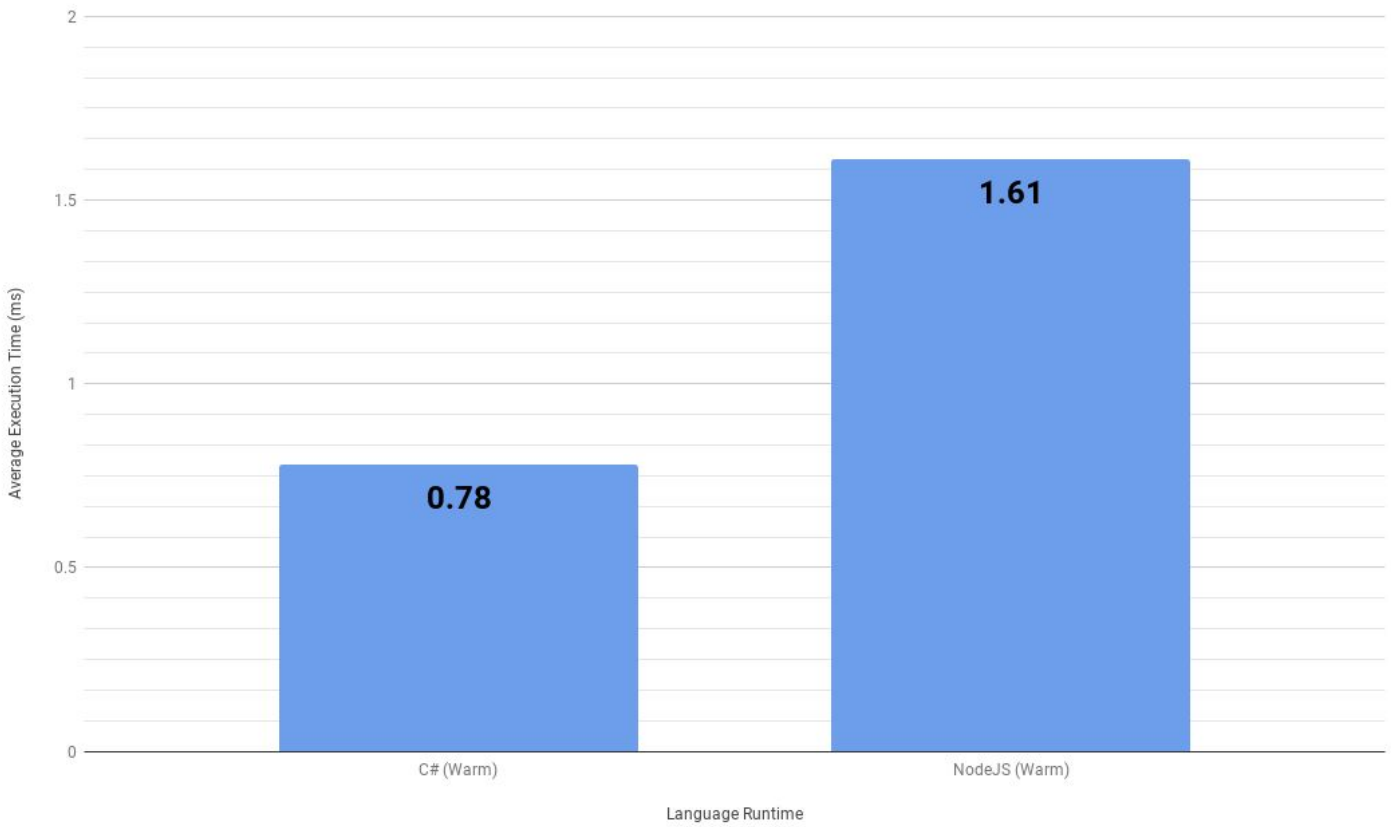
Results Analysis - Azure Functions

Azure Functions

Warm Start

500 Function Invocations Per Runtime

Nov 2018

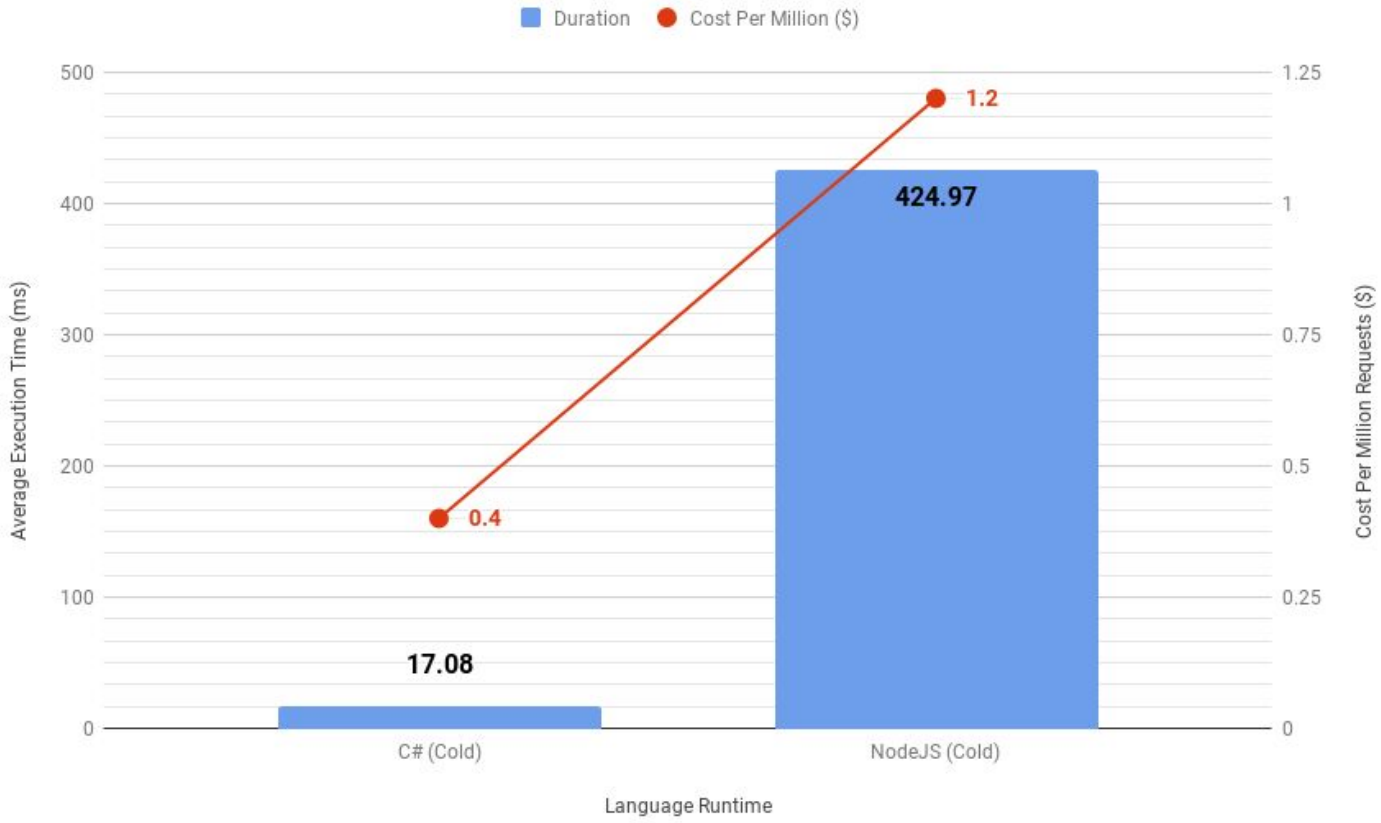


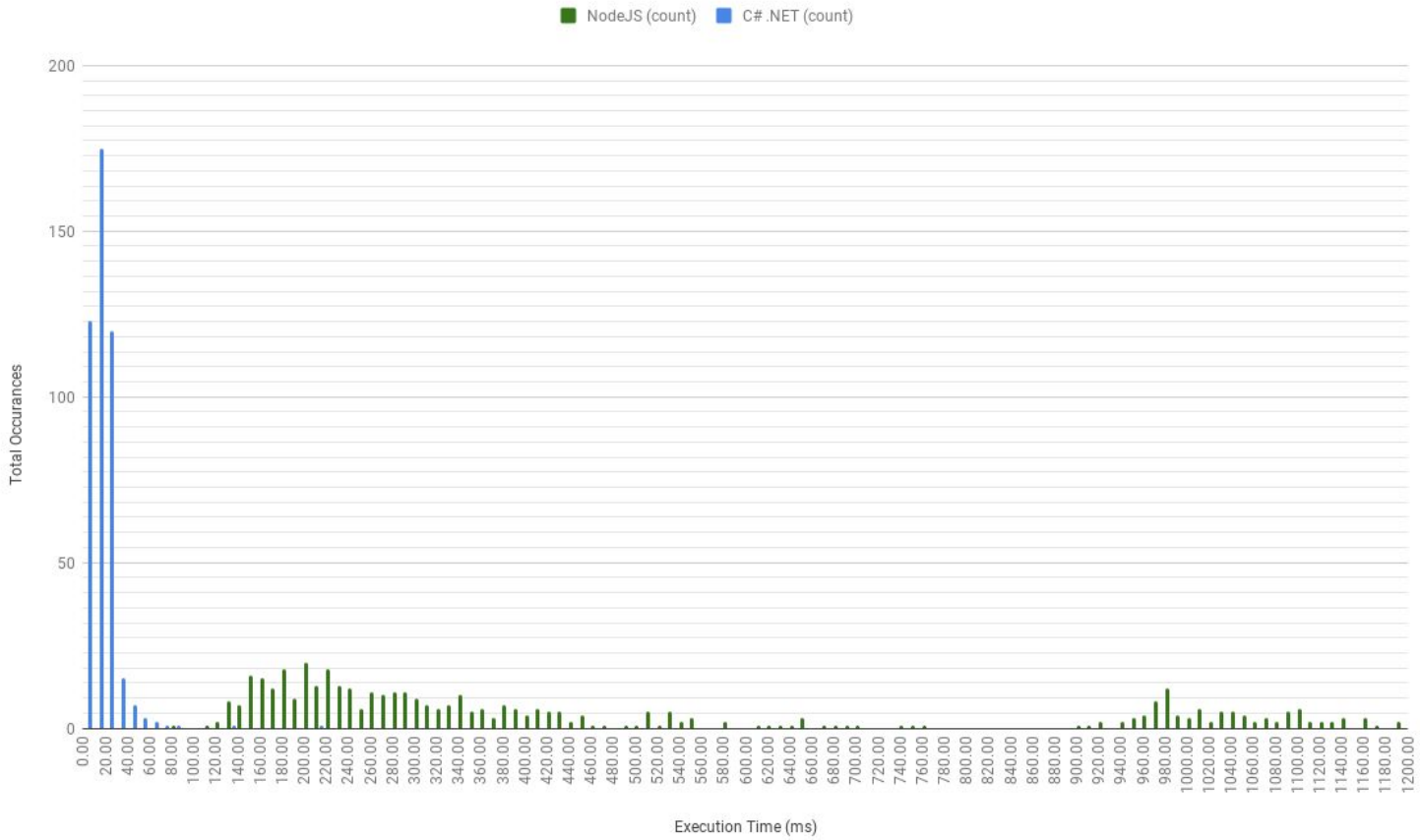
Azure Functions

Cold Start

500 Function Invocations Per Runtime

Nov 2018





Azure Performance

-Start Histogram

500 Function Invocations Per Runtime

Nov 2018

Results Analysis - AWS vs Azure

AWS vs Azure

NodeJS

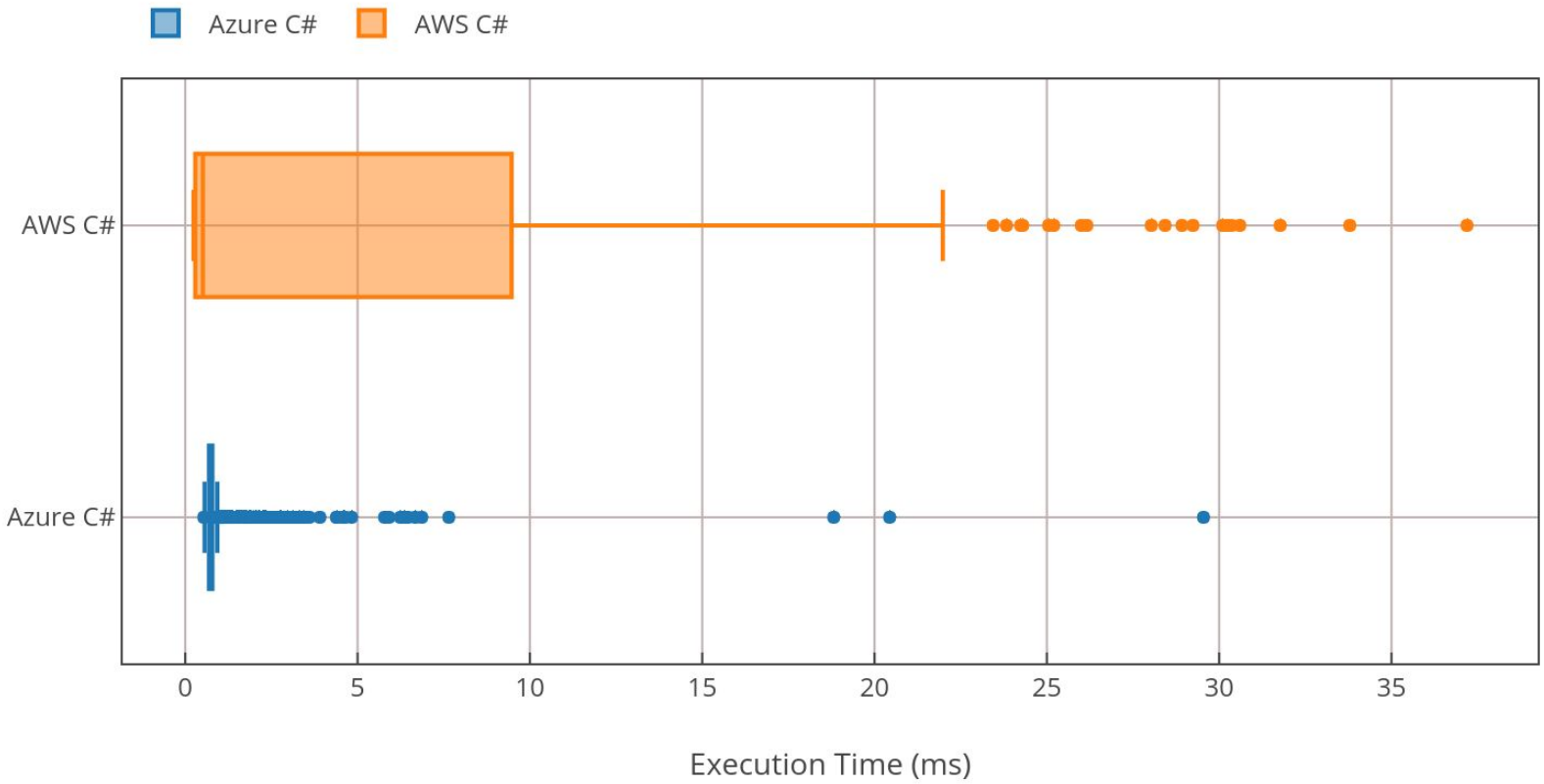
Cold Start

500 Function Invocations Per Runtime

Nov 2018

AWS vs Azure

C# .NET Warm Start



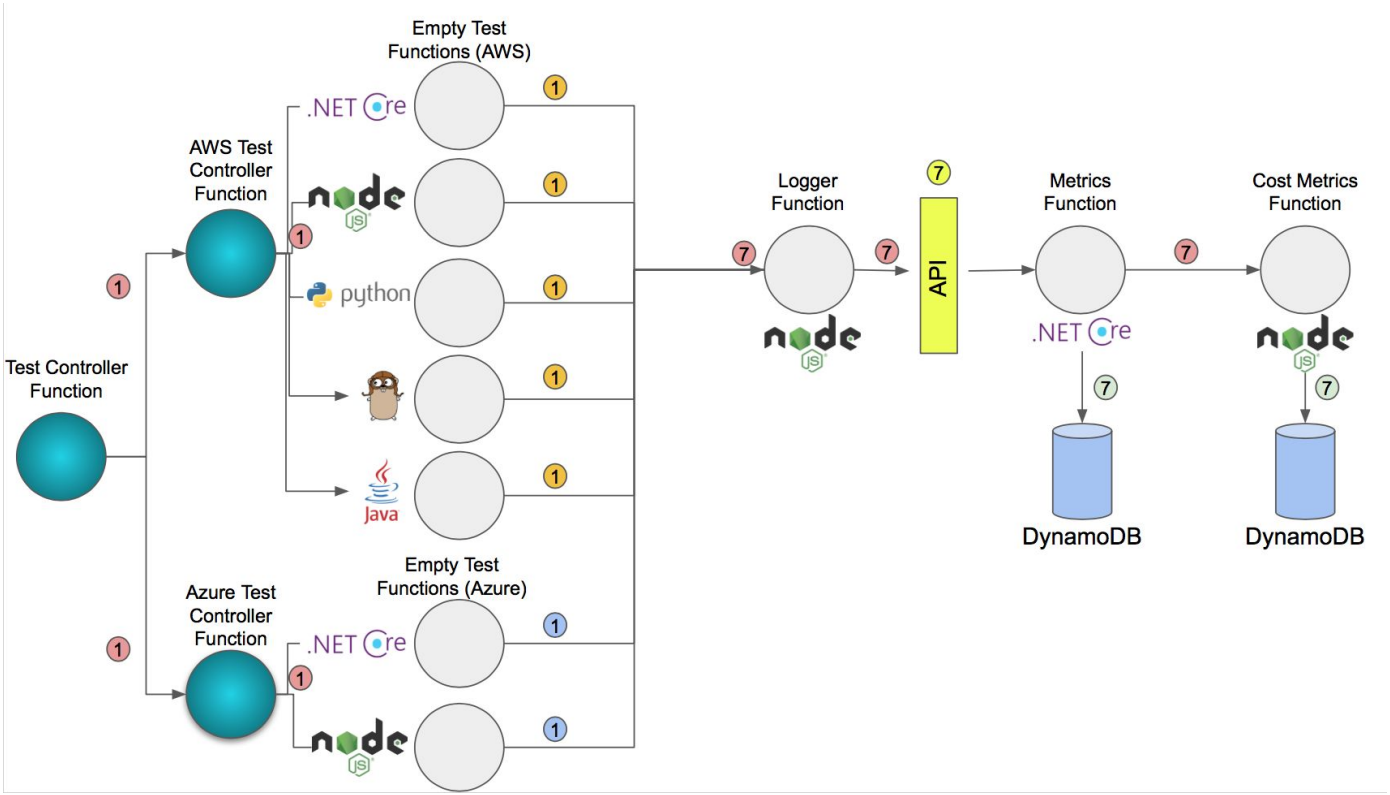
Cost Analysis

CostHat Model

(Leitner et al. 2016)

Serverless Performance Framework Architecture

30k Function Invocations based on 1,000 TPS



TPS Cost Calculations

AWS Lambda

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

TPS Cost Calculations

Azure Functions

Language Runtime	Cost Per Day @ 100-TPS	Cost Per Day @ 30k-TPS	Cost Per Year @ 100-TPS	Cost Per Year @ 30k-TPS
.NET C#	$3.46	$1,036.80	$1,261	$378,432
NodeJS	$10.37	$3,110.40	$3,784	$1,135,296

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.

Conclusion
Cold-Start Performance

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.

Conclusion
Warm-Start Costs

In warm-start testing, despite some languages being 200%-300% times slower than others, cost is not (immediately) affected.

Conclusion
Pace of Change

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.

Conclusion

Function Composition

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

- Leitner, P., Cito, J. & Stöckli, E. (2016), Modelling and managing deployment costs of microservice-based cloud applications, in ‘Proceedings of the 9th International Conference on Utility and Cloud Computing’, ACM, pp. 165–174.