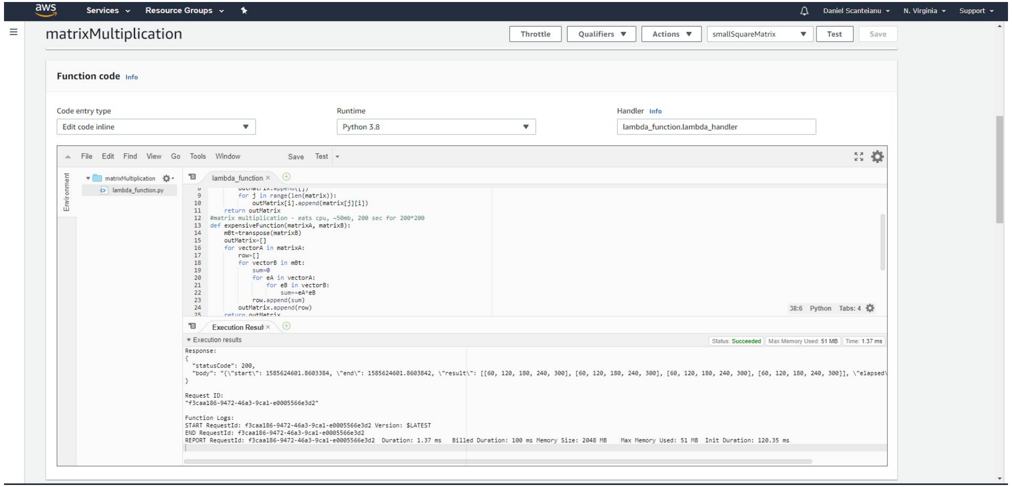
Black Box Benchmarking FAAS Platforms

Daniel Scanteianu OS/2 Project

What is FAAS/Serverless?

- Function As A Service
- Serverless
  - Cloud provider handles server
  - You write some code
  - Cloud provider deploys
  - Function gets triggered
  - Cloud provider handles scaling up and scaling down
  - You get billed per invocation/execution time
- Uses containers under the hood
  - "Preheated" runtimes accept function run requests



matrixMultiplication

Function code Info

Code entry type: Edit code inline

Runtime: Python 3.8

Handler Info: lambda_function.lambda_handler

```
        outMatrix.append([])
        for j in range(len(matrix)):
            outMatrix[i].append(matrix[j][i])
    return outMatrix
#matrix multiplication - eats cpu, ~50mb, 200 sec for 200*200
def expensiveFunction(matrixA, matrixB):
    mBt=transpose(matrixB)
    outMatrix=[]
    for vectorA in matrixA:
        row=[]
        for vectorB in mBt:
            sum=0
            for eA in vectorA:
                for eB in vectorB:
                    sum+=eA*eB
            row.append(sum)
        outMatrix.append(row)
    return outMatrix
```

Execution results

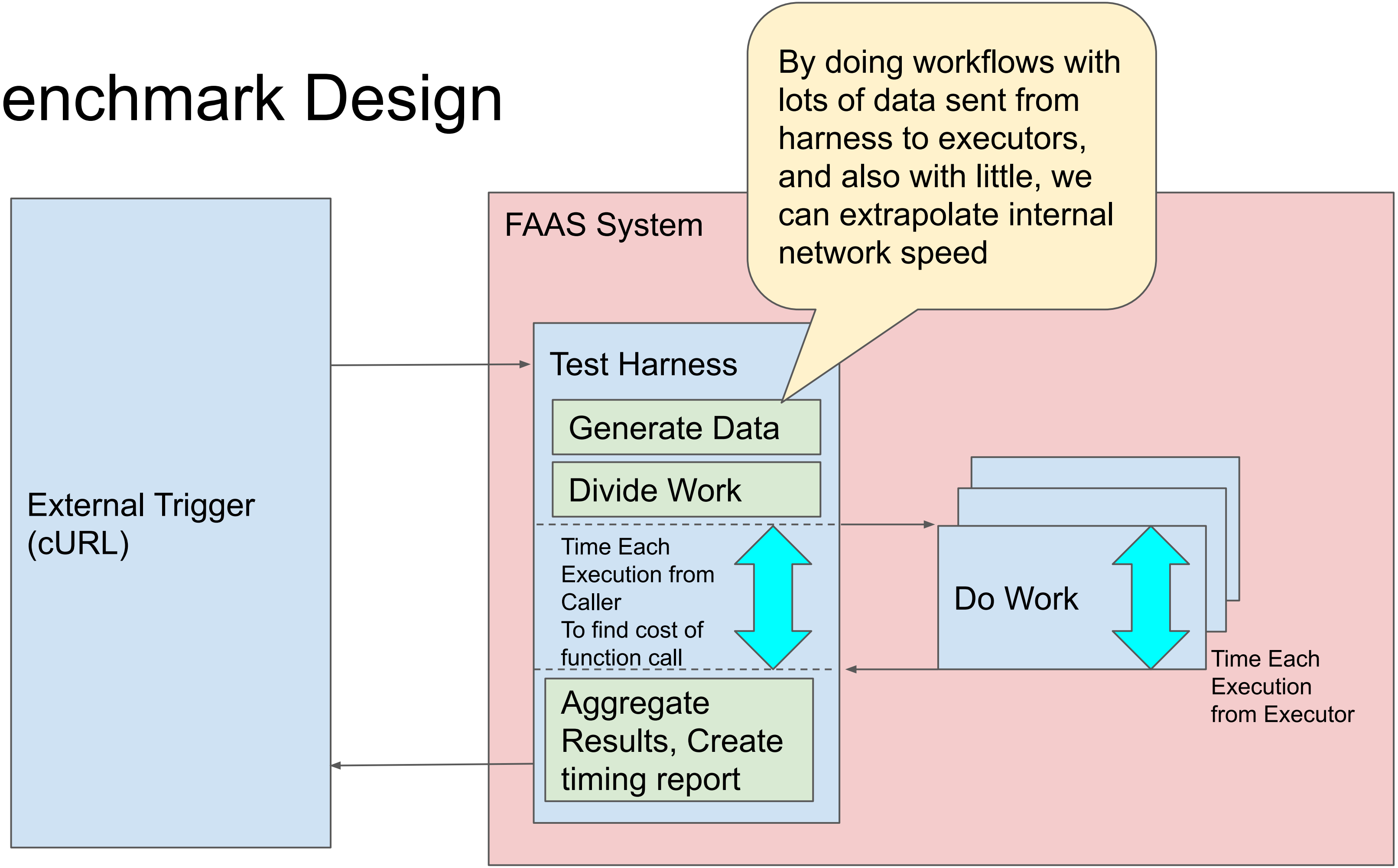
Status: Succeeded | Max Memory Used: 51 MB | Time: 1.37 ms

```
Response:
{
  "statusCode": 200,
  "body": "{\"start\": 1585624601.8603384, \"end\": 1585624601.8603842, \"result\": [[60, 120, 180, 240, 300], [60, 120, 180, 240, 300], [60, 120, 180, 240, 300], [60, 120, 180, 240, 300]], \"elapsed\
}

Request ID:
"f3caa186-9472-46a3-9ca1-e0005566e3d2"

Function Logs:
START RequestId: f3caa186-9472-46a3-9ca1-e0005566e3d2 Version: $LATEST
END RequestId: f3caa186-9472-46a3-9ca1-e0005566e3d2
REPORT RequestId: f3caa186-9472-46a3-9ca1-e0005566e3d2  Duration: 1.37 ms   Billed Duration: 100 ms Memory Size: 2048 MB    Max Memory Used: 51 MB  Init Duration: 120.35 ms
```

Benchmark Design



By doing workflows with lots of data sent from harness to executors, and also with little, we can extrapolate internal network speed

External Trigger (cURL)

FAAS System

Test Harness

Generate Data

Divide Work

Time Each Execution from Caller
To find cost of function call

Aggregate Results, Create timing report

Do Work

Time Each Execution from Executor

Where we left off

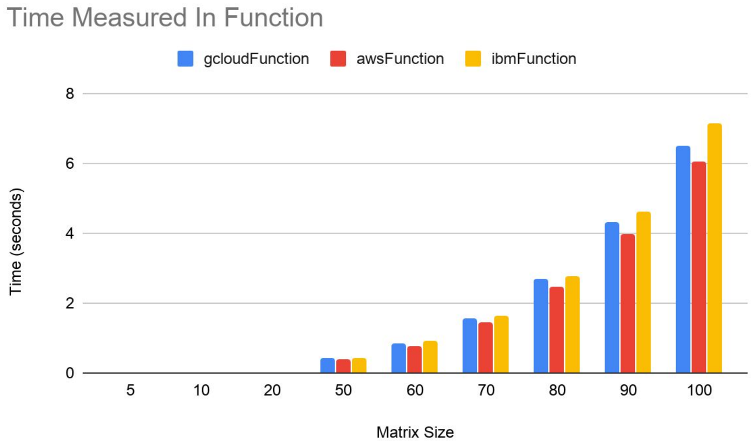
- Goal: find out how different FAAS systems perform under different load patterns
    - Matrix Multiplication Test
        - Meant to find out who had a faster cpu
        - Gathered other metrics
    - AWS vs Google - we found AWS was faster in the function, but Google spent less time between the function and the harness
        - Add IBM - based on Microsoft results from other study, IBM is a more production-ready choice
        - Naive matrix multiplication has been pointed out to not be the most valid test

Other Work

- Reliability (failure rate) when running lots of requests:
  - AWS 0.954%, GCF 2.777%, IBM 2.775, MS 95.976%
  - https://www.ise.tu-berlin.de/fileadmin/fg308/publications/2020/Online_Preprint___SAC_2020__Benchmarking_Elasticity_of_FaaS_Platforms_as_a_Foundation_for_Objective_driven_Design_of_Serverless_Applications.pdf
- Startup delays for google cloud vs aws - AWS<Google<IBM
  - https://oaciss.uoregon.edu/icpp18/views/includes/files/pos115s2-file1.pdf
- Evaluation of AWS, Google, MSFT on cpu intensive tasks (video processing, ml) shows that for the most part outperforms others, especially toward high end
  - http://jckim.me/assets/paper/FunctionBench%20-%20A%20Suite%20of%20Workloads%20for%20Serverless%20Cloud%20Function%20Service.pdf
- Elasticity - ability to serve requests at same speed as load increases https://www.cs.colostate.edu/~shrideep/papers/ServerlessComputing-IC2E-2018.pdf (also other tests)
- Total time vs execution time (USENIX)! In aws step functions
  - https://www.usenix.org/system/files/conference/atc18/atc18-akkus.pdf

Test: Matrix Multiplication

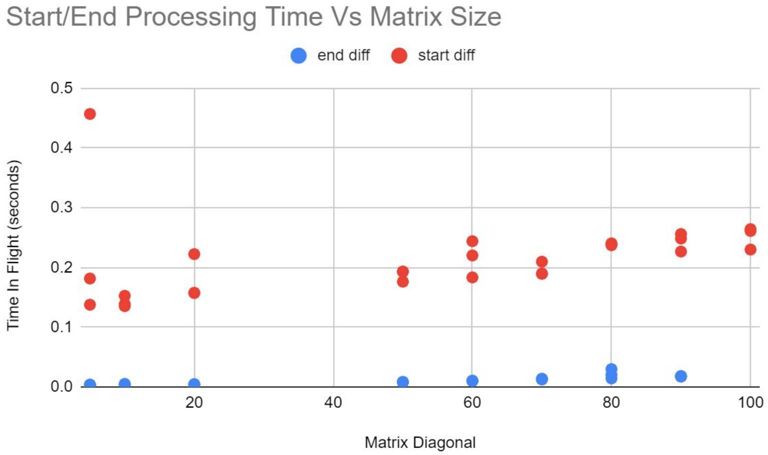
- Naive Matrix Multiplication Algorithm
  - 2gb instances, O(n^2)*n^2 cells in matrix
- Headline results
  - AWS has a faster CPU than Google or IBM
    - Statistically significant difference
    - ~18% faster - explainable by slightly newer CPUs
  - Presumably very similar underlying systems
    - Enterprise grade vms running containers which host functions, x86 based, linux
- Other observations were more interesting

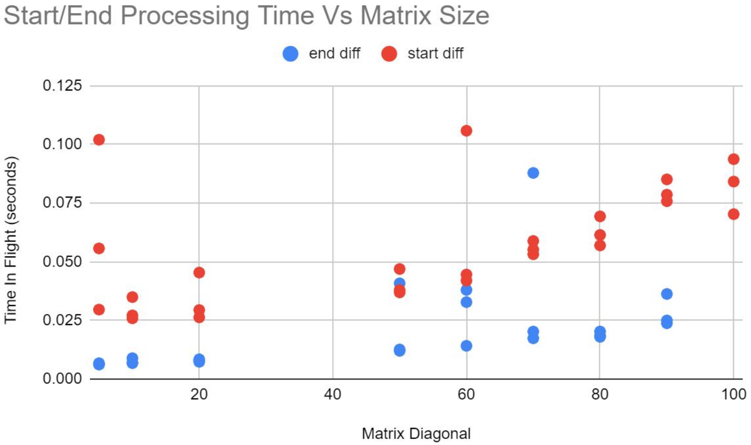


aws < gc	aws < ibm	gc < ibm
p=0.0001434475609	p=0.0003607757513	p=0.001313550811

Observations

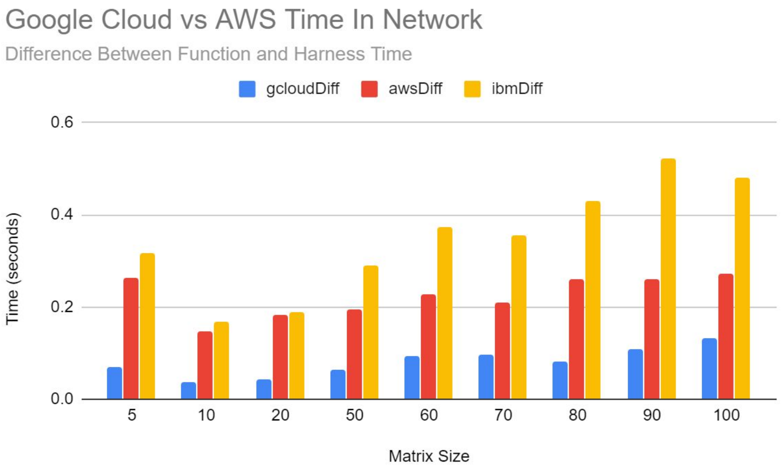
- Measured start/end harness vs function time
- Noticed aws end time was pretty low, but mild upward trend as the matrix size increased
- Noticed that there was a bigger spread in Google timings than in Amazon timings
- New experiment - which provider has fastest (and most consistent) network?





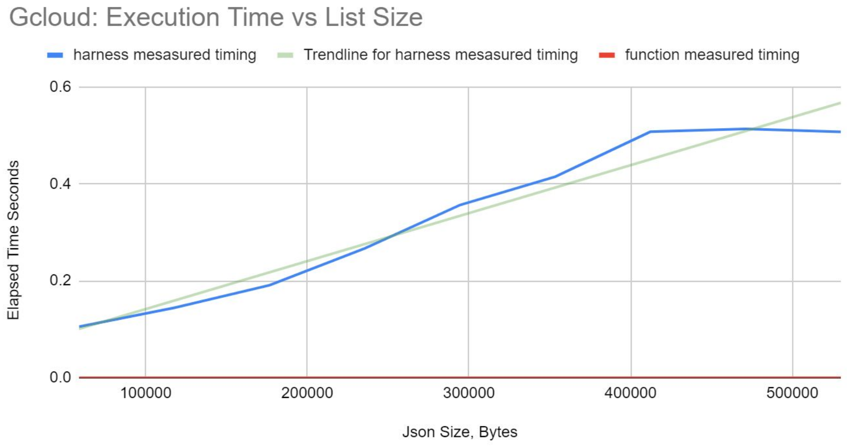
Test: Large Network

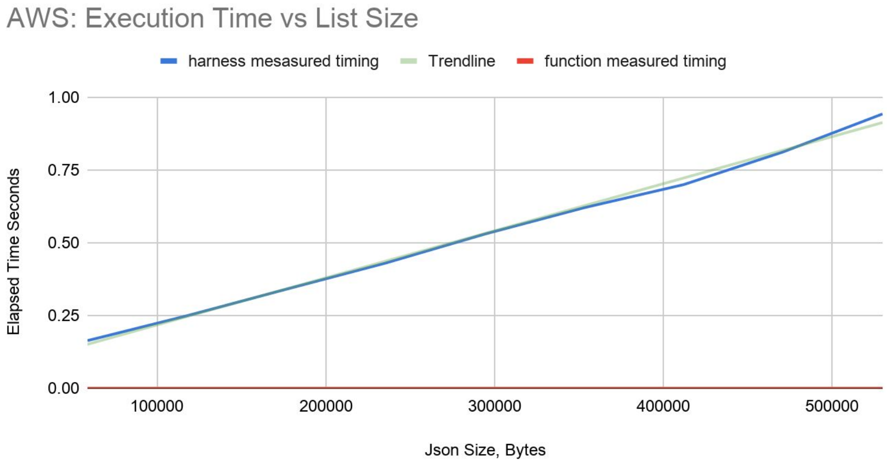
- Very simple test
  - Generate huge list
  - Send over the wire to function
  - Return length of list
  - Assuming JSON serialization, estimate network
- Isolates network component
- Desired output
  - Learn about the network
    - Can estimate size of packet sent over the wire
    - Find out both approximate network speed
      - Includes ser/de because it's a list encoded in json
      - "Real" cost of passing data to a function
    - Learn about network variability

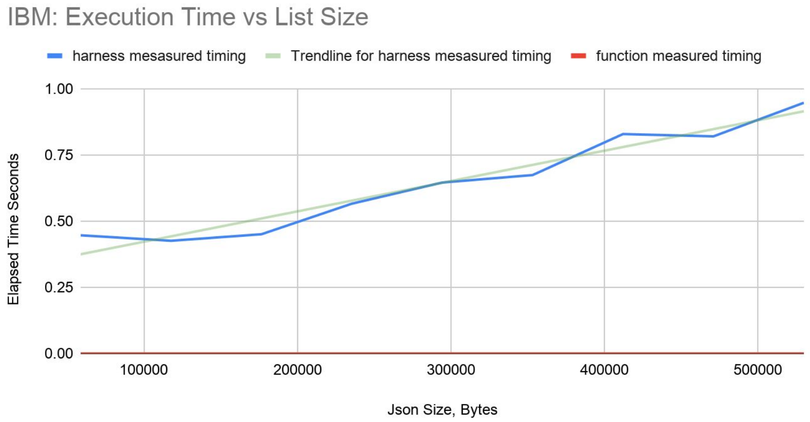


Test: Large Network

- AWS is surprisingly low in variability
- Last 3 Google data points might indicate that they let larger data have more bandwidth







Test: Large Network

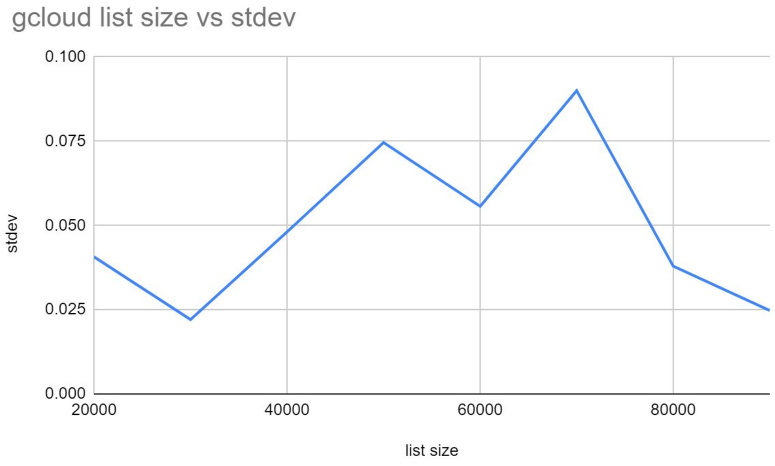
- Google is fastest
- AWS - 99.25% of variance in response timing due to packet size
  - Surprising - means network is super super consistent
  - Throttling or underused?
- Extremely unlikely difference in means is due to random variance

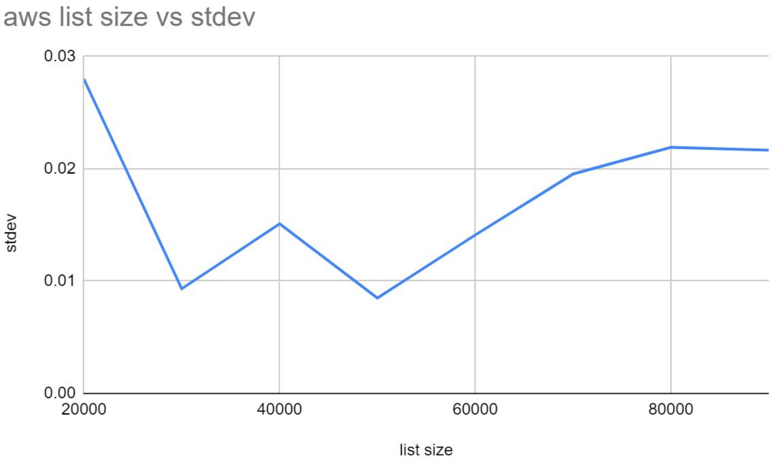
Provider	Network Speed (MBPS)	Startup Latency	R^2
AWS	4.910699466	0.05582174593	0.9925222419
Google	7.025047395	0.04267904891	0.8691661488
IBM	5.352188787	0.3073757185	0.7667496019

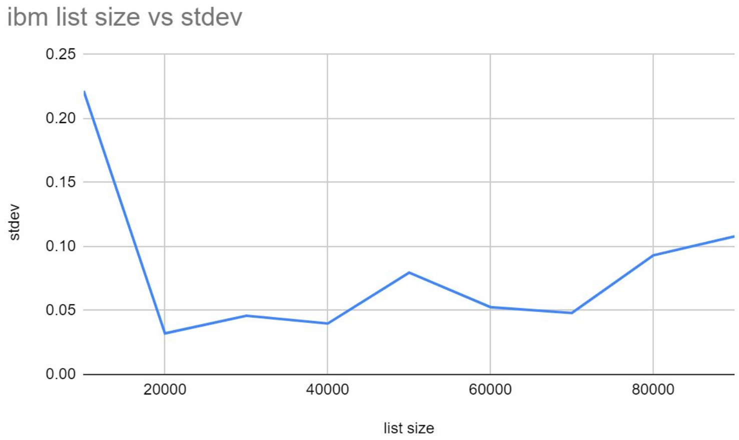
aws < gc	aws < ibm	gc > ibm
p=0.0	p<0.00001	p=0.0

Test: Large Network

- AWS looks like I expect
    - More data = more opportunity for network to vary
- Google very consistent for bigger requests
- Initial startup latency clouding measurement







Conclusions: Large Network

- Amazon is very predictable and consistent
    - Slowest network, but seems that the only factor affecting start up time increase was increase in data size
        - Boto3 library as opposed to raw http request
        - Internal throttling may be happening
- Google is faster than other two, and might have benefits for larger data sets
- Was able to get actual MBPS figure for all 3 providers
- All 3 tests included built in lambda ser/de time
    - No really good way to bypass this

Perspective on FAAS platforms

- Small, distinct learning curve for each individual provider
- AWS has a much larger selection of runtimes available than other providers
    - More production-ready system than other providers
        - Few failures
        - Responsive UI
        - Super consistent
- Google is powerful, but rougher around the edges
    - Glitchy UI was hard to work with
- Pick one ecosystem and switch with it
    - Migrating functions is bad
    - Migrating communication/IAMs/Databases is worse

Conclusions on FAAS as a tool

- Very easy to learn and spin up services
- All IBM experiments fit within free tier limits
  - Generous offering, especially for students and others with limited resources
- Required much less time spent in configuring a web server
  - Very easy to add rest endpoint exposed in Google or IBM directly to an otherwise static website
- Automatic scaling is a very attractive offer
  - Huge devotion of engineer time to scalability and scaling in industry, and very complex planning is involved
- Potential for vendor lock in
- Well suited for people building applications
  - Use the right language for the function, common interface
- Probably way more reliable than having/maintaining your own servers

Future

- Case for implementing managed FAAS in industry
  - Multiple teams who own microservices might benefit
- I might try to make some contribution to some open source FAAS platform over the summer
- Any Questions?