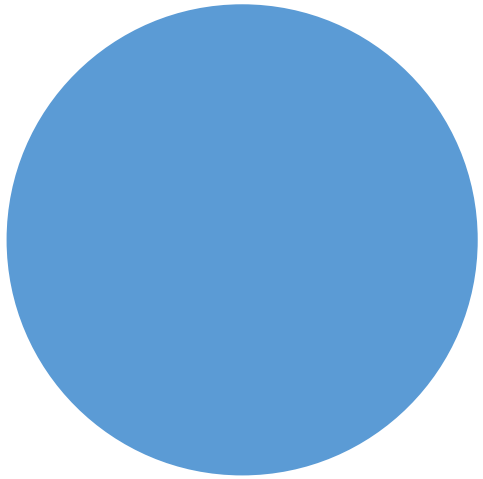
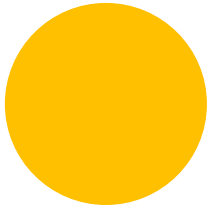
Serverless Programming: All Hype? Are there benefits?

About Me!

- I’m Pavan Agrawal
- Currently a Student
- Working as a full-stack developer in a local business.
- Follow me on Twitter: @Pavan_Agrawal_1

Disclaimer: For this talk we are going to focus on the style of serverless that involves deploying pure code to cloud providers.

This will not cover the newer style of serverless that involves a Docker container as a service.

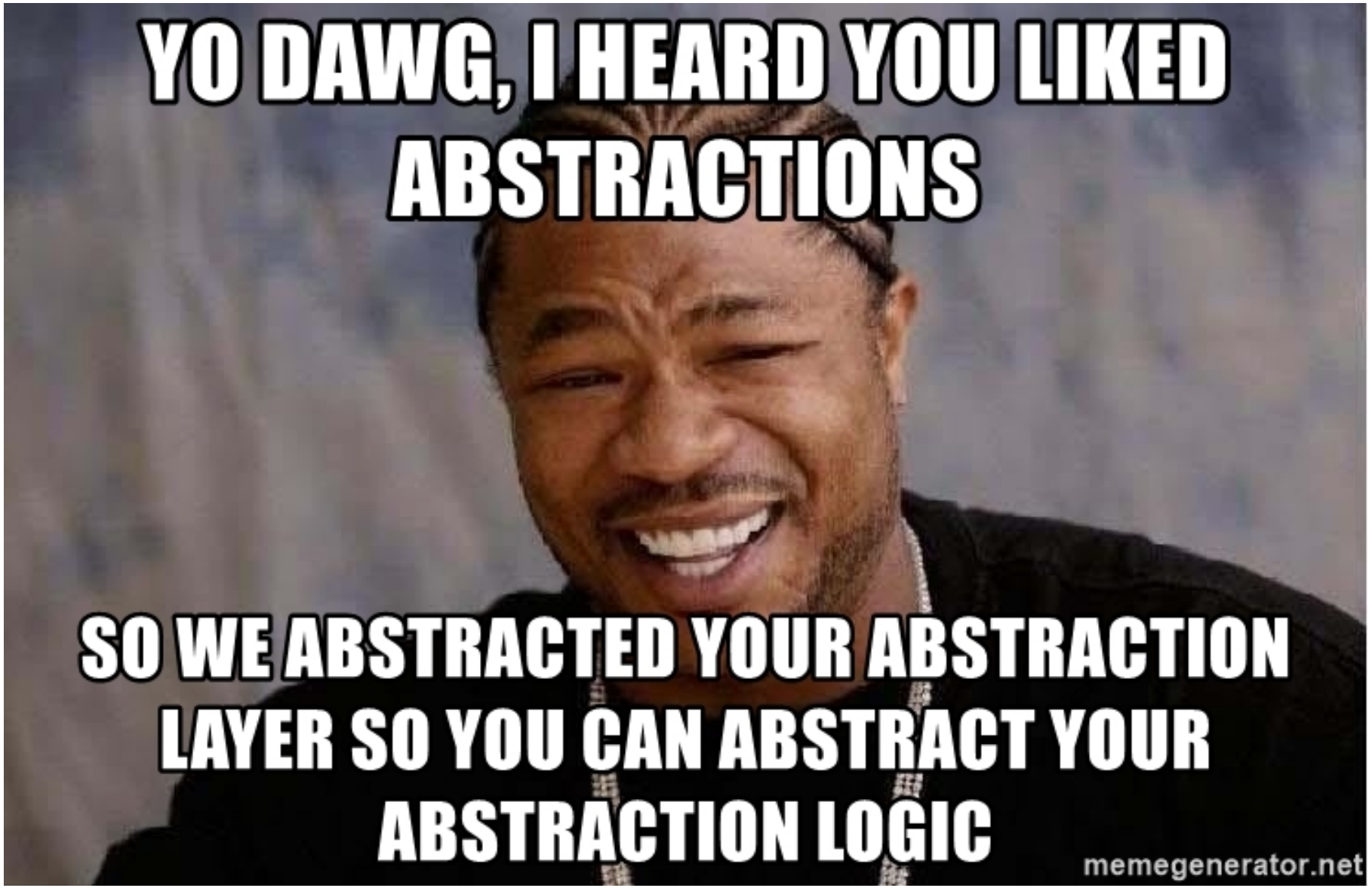
What is Serverless Programming?

For starters, it’s not actually serverless.

Abstraction over the platform

Abstraction over the scaling

Abstraction over all runtime and state


YO DAWG, I HEARD YOU LIKED ABSTRACTIONS
SO WE ABSTRACTED YOUR ABSTRACTION LAYER SO YOU CAN ABSTRACT YOUR ABSTRACTION LOGIC
memegenerator.net

For starters, it’s not actually serverless.

Abstraction over the platform

Abstraction over the scaling

Abstraction over all runtime and state

```python
import math
def isPrime(num):
    for i in range(int(math.sqrt(num))):
        if (num % i == 0): return False
    return True
```

All you need to do to use serverless is just provide code.

Most cloud providers provide serverless runtime environments.

- AWS Lambda
- Azure Functions
- GCP Functions

There are multiple frameworks that aim to reduce vendor lock in

- Serverless (sls)
- Zappa


Suber


Suber

Serverless usually comes with billing benefits

- Most providers bill for serverless deployments with a consumption based model: you only pay for how much you use.
- No more paying for when your service is not in use

Performance and Scaling Benefits

- You no longer need to consider scaling, most serverless platforms invisibly scale the instances running your code automatically.
  - Advantage with this? As instances scale, you don’t pay for the time the instances stay idle, you’re only paying for usage…

Serverless programming also has security benefits

More abstractions = less responsibility for the developers*

The underlying stack and its security are updated by engineers at cloud providers, shifting the responsibility of security over

0-day found in Python V 3.XX?

Hopefully, the cloud provider you've chosen to host with will update ASAP

With every abstraction, there are downsides

- Less control over the stack that runs
  - Need a newer version of NodeJS that the Serverless platform doesn't support?
    - You can't*
  - Have a patch that speeds up the Python runtime by 2x, but won't be merged to master any time soon?
    - Nope, can't use it, you're stuck with what the cloud provider has preconfigured

Performance and Scaling

Scaling algorithms are almost always proprietary to cloud providers.

Most of the time serverless platforms are limited by what hardware they can run on.

- Example: AWS
  - RAM: 3008MB
  - Temporary Disk Space: 512MB
  - CPU: 1 Core

Cold Starts

Biggest Issue (IMO)

Depends on size of bundle uploaded

Can also depend on language chosen. JS seems to have the lowest cold starts, with Python and other interpreted languages being slower.

Demo!

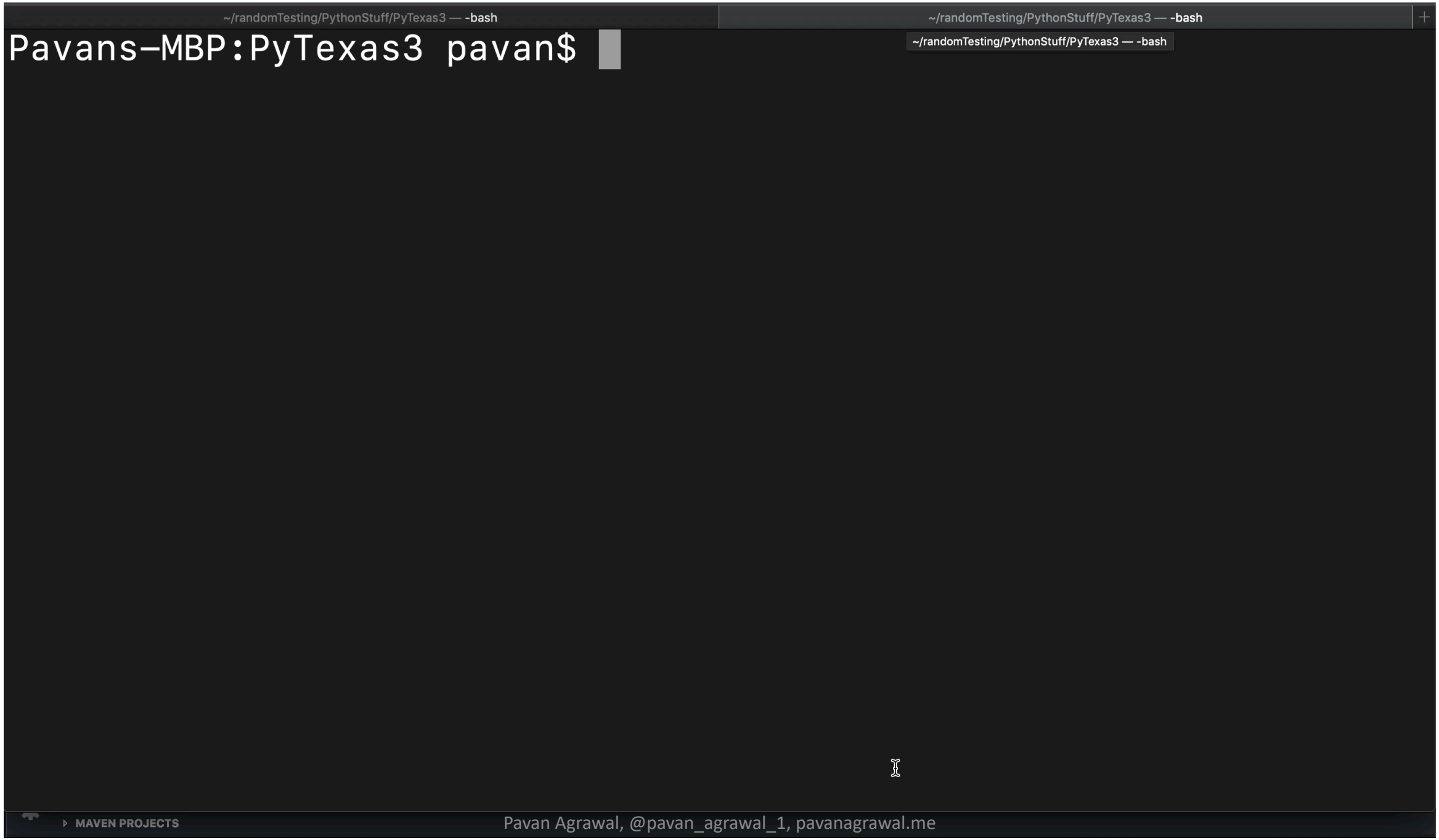
Hash Cracking

Brute Force

Python -> Flask for serving the demo

Zappa Deployment

AWS Lambda Hosting


~/randomTesting/PythonStuff/PyTexas3 — -bash
~/randomTesting/PythonStuff/PyTexas3 — -bash
~/randomTesting/PythonStuff/PyTexas3 — -bash
Pavans-MBP:PyTexas3 pavan$
MAVEN PROJECTS

~/randomTesting/PythonStuff/PyTexas3 — -bash ~/randomTesting/PythonStuff/PyTexas3 — python app.py

```
(venv) Pavans-MBP:PyTexas3 pavan$ time python compute.py --addres
s http://localhost:5000/ --concurrency 1 --count 1
senthttp://localhost:5000//?hash=a3bee0cc490ea95caa6bf436679d1a3e
?&start=0&end=1000000
processedhttp://localhost:5000//?hash=a3bee0cc490ea95caa6bf436679
d1a3e?&start=0&end=1000000
['P! ']

real    0m0.962s
user    0m0.145s
sys     0m0.063s
(venv) Pavans-MBP:PyTexas3 pavan$ time python compute.py --addres
s http://localhost:5000/ --concurrency 1 --count 1
```

MAVEN PROJECTS

~/randomTesting/PythonStuff/PyTexas3 — -bash | ~/randomTesting/PythonStuff/PyTexas3 — python app.py

```
processedhttp://localhost:5000//?hash=a203308917292ce11feeaf5c58b
39f02?&start=0&end=1000000
senthttp://localhost:5000//?hash=0a118279c8ad27c5554bd752fc1b4d83
?&start=0&end=1000000
processedhttp://localhost:5000//?hash=0a118279c8ad27c5554bd752fc1
b4d83?&start=0&end=1000000
senthttp://localhost:5000//?hash=8d91651e4d7f4bef5dbe5f1fb1876edd
?&start=0&end=1000000
processedhttp://localhost:5000//?hash=8d91651e4d7f4bef5dbe5f1fb18
76edd?&start=0&end=1000000
['0]!', 'z1&', '>fp', 'y 4', '1*\r', '}O\x0c', 'BDK', 'j]B', 'Rj%
', 'vTb']

real    0m5.922s
user    0m0.206s
sys     0m0.069s
(venv) Pavans-MBP:PyTexas3 pavan$
```

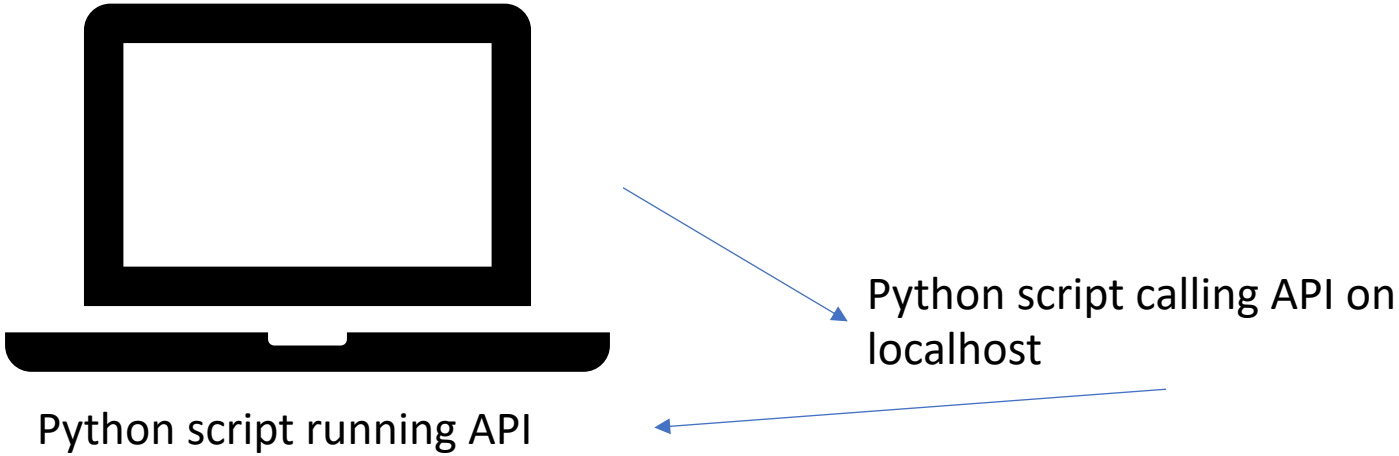
MAVEN PROJECTS

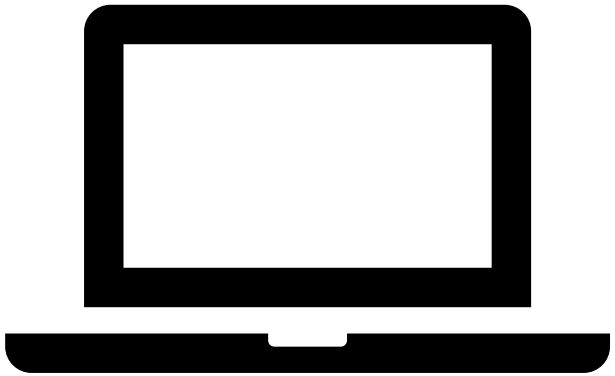
```
...compute.py --address https://s32w06gbia.execute-api.us-east-1.amazonaws.com/dev/ --concurrency 200 --count 1000    ~/randomTesting/PythonStuff/PyTexas3 — python app.py
  File "/Users/pavan/miniconda3/lib/python3.7/multiprocessing/context.py", line 277, in _Popen
    return Popen(process_obj)
  File "/Users/pavan/miniconda3/lib/python3.7/multiprocessing/popen_fork.py", line 20, in __init__
    self._launch(process_obj)
  File "/Users/pavan/miniconda3/lib/python3.7/multiprocessing/popen_fork.py", line 69, in _launch
    parent_r, child_w = os.pipe()
OSError: [Errno 24] Too many open files

real    0m1.063s
user    0m0.668s
sys     0m1.287s
(venv) Pavans-MBP:PyTexas3 pavan$ time python compute.py --address https://s32w06gbia.execute-api.us-east-1.amazonaws.com/dev/ --concurrency 200 --count 1000
```

MAVEN PROJECTS

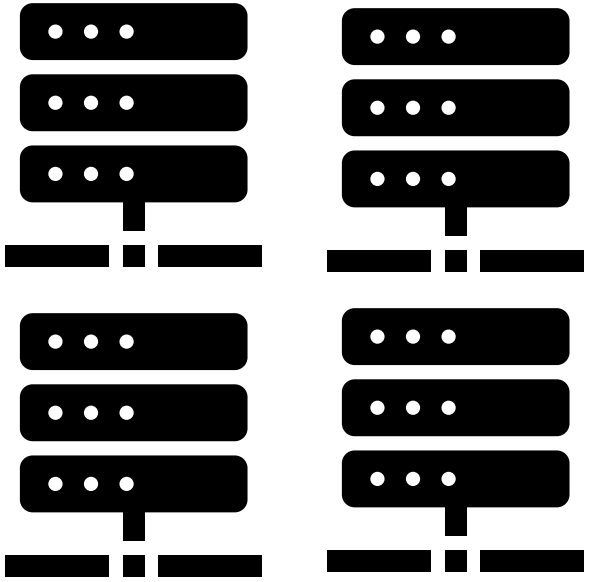
Demo Results

- Using the power of serverless, we were able to scale up the algorithm in a very short amount of time.
- What took ~6 seconds to compute 10 times on my laptop was computed 1000 times on a serverless platform in ~20 seconds.


Python script calling API on localhost
Python script running API



Python script calling API on AWS Lambda



How much money did I just waste?

Number of executions: 1000

Average Time Per Execution: 3-4seconds (3000-4000ms, maximal runtime)

Allocated Memory: 512MB

0.00001667/GB-S

Total GB-S Used: (3.5seconds)*(1000 executions)*(.5GB) = 1750GB-S

Cost: $.02917

How difficult was this to do?

10 minutes to write the Python code

10 minutes to deploy with Zappa

Scenario 1

- Internal Business App
- Programmed with TypeScript

Motivation for Serverless Design

- A lot of the API queries for the business app are event driven and require horizontal scaling for processing.
- At times during the day, the business app goes unused, but at other times, users flood onto the system.

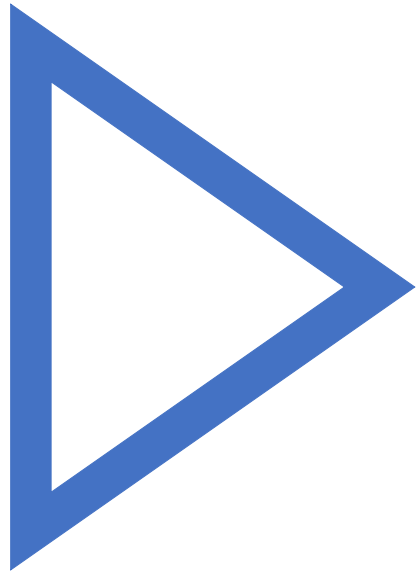
Development Flow

- Prototyped a V1 with a Node based backend deployed to AWS Lambda.
- Soon, we ran into a lot of issues with utilizing a serverless backend.
  - For users logging into the application, cold starts became very hard to deal with.
  - Monolithic Deployment VS Microservices?
    - Cold start
    - Development Difficulty, Redeployment
  - Connection Reuse

Were there benefits?

- Yes, for the event driven API exposed on the same backend platform was heavily sped up. Horizontal scaling for this API was *amazing*
- Cost benefits: We were able to move from ~$40 VM to spending about a $1/month

Serverless development is not magic



Solution

- Splitting our backend into two separate deployments: the event driven APIs are now serverless deployed and the user driven frontend is deployed on a VM.

Engineering Time

- If we had access to more engineering resources, we could technically adapt the user driven frontend to a serverless environment.

Scenario 2

- Sign up system for a school club
- Requests to the API are event driven
  - They come in bursts when events are put out in the website
  - Up to 200 students can register for events on the website in any given minute, causing Rqs/sec to peak at 100/s.

Advantages of using a serverless environment

Horizontal scaling:

- Cold starts were not really perceived as an issue from users, as it only happened 2/3 times every scaleup.

Pricing

- No longer paying for idle time when users aren’t signing up for events.

Deployment Speed

- No configuring VMs Scale Sets, etc.

Take Aways

- Serverless is *not* a be all end all solution for everything
  - Really good in the case fast horizontal scaling is required
- Serverless programming tends to encourage vendor-lock in, so modularize your code in that it can easily adopted to another vendor's serverless design.
- In cases for backends where serverless can be a good tool to use, it can be split out into a serverless deployed backend, and a normally deployed backend.

Conclusion

- Serverless Programming: All Hype? Are there benefits?
  - Of course, there are benefits.
  - Benefits do not come without drawbacks
- Ultimately, you will need to put in the effort to really determine if serverless deployment would be a beneficial technology to use in engineering time, deployment costs, or end user experience.

Thanks for listening!

Open for internships!

Pavan Agrawal, @pavan_agrawal_1, pavanagrawal.me