

# Lambda

- **Lambda functions**
- **Intro to serverless computing**



# Execution continuum



**EC2, EKS, ECS, Fargate**

- *Run any software you want for as long as you want*
- *Complete control over HW and SW*
- *Hardest to config*

**Elastic Beanstalk**

- *Upload .zip file*
- *Limited software choices*
- *Some control over HW and SW*

**API Gateway + Lambda**

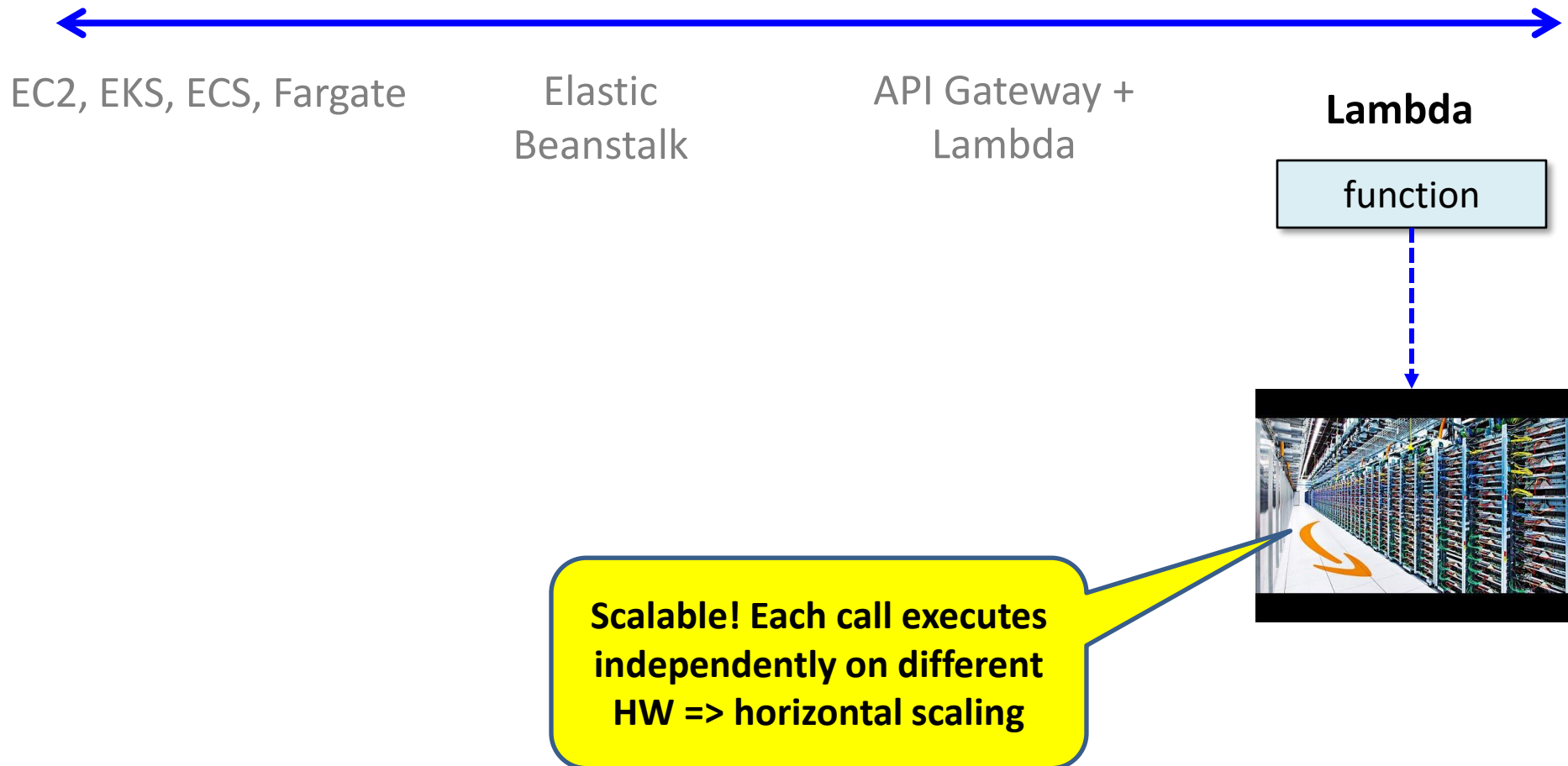
- *Function based*
- *Near-zero config*
- *Web service + functions (15-min limit)*

**Lambda**

- *Function based*
- *Near-zero config*
- *Short execution (< 15 mins)*

# AWS lambda

- By far the simplest, least expensive way to compute



# AWS lambda / Azure functions / Google functions

- **Standalone functions executed on demand**
  - *Can be written in JavaScript, Python, Java, C++, etc.*
  - *Execution time is limited (AWS => 15 minutes)*
- **Callable in a variety of ways:**
  - *Like a traditional function( ) using AWS library*
  - *Based on **events** that occur (e.g. uploading an item into S3)*
  - *Via **function URL** through AWS-managed web server*
  - *Via **API Gateway** offering a more customizable AWS-managed web server (e.g. test vs. production, more authentication options, ...)*

# Example: prime factors in Python

<https://2noicxltxjwxxt4ego5d7q4uc40bcgjw.lambda-url.us-east-2.on.aws/?n=600851475143>

<https://2noicxltxjwxxt4ego5d7q4uc40bcgjw.lambda-url.us-east-2.on.aws/?n=6008514751439999>

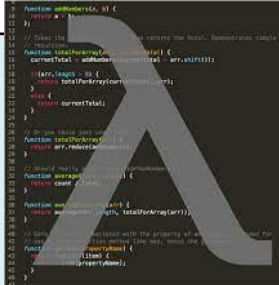
```
import json

def prime_factors(n):
    i = 2

    factors = []
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)

    if n > 1:
        factors.append(n)

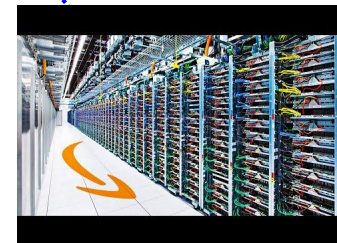
    return {
        'statusCode': 200,
        'body': json.dumps(factors)
    }
```



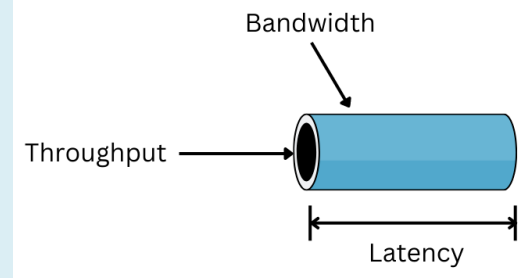
```
function handler(event, context) {
    // Parse the input
    const { n } = JSON.parse(event.body);

    // Calculate the prime factors
    const factors = prime_factors(n);

    // Return the result
    return {
        statusCode: 200,
        body: JSON.stringify(factors)
    };
}
```



# Latency vs. Throughput



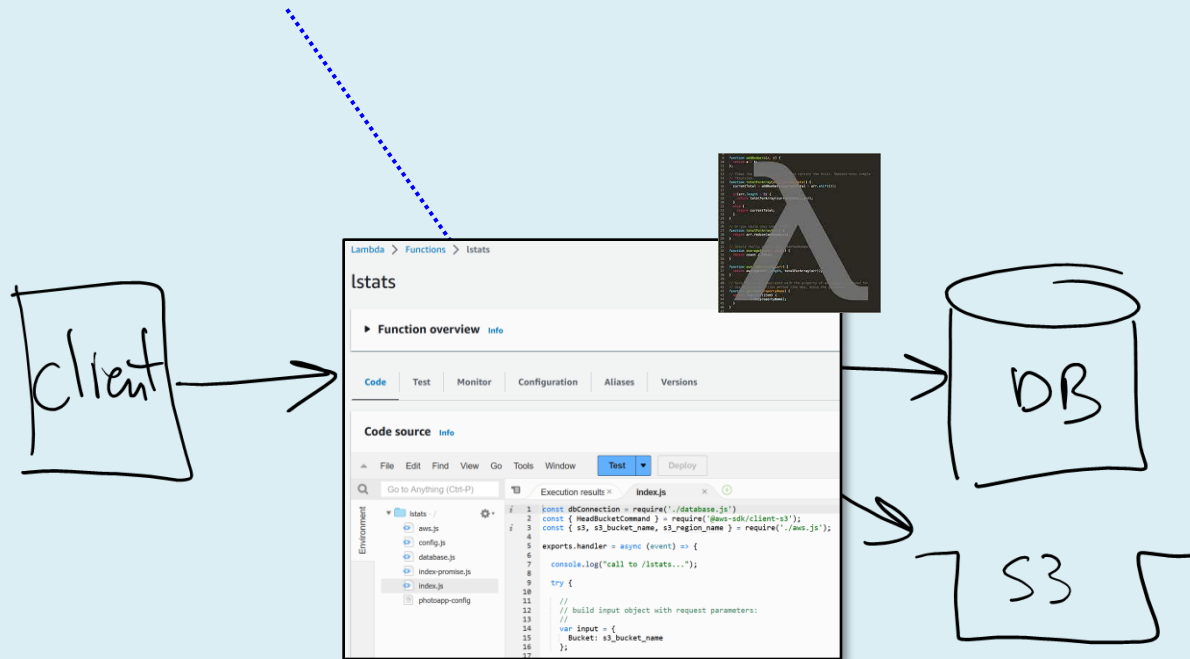
- **Lambda trades latency for throughput**
- **Lambda functions have a longer latency (i.e. slower)**
  - *Takes time to load function + support libraries*
- **Lambda offers higher throughput (supports more clients) by automatically scaling calls across EC2**

```
hummel> ab -c 200 -n 2000 https://k7hwywasoufmgzefszfgpfhzfq0iejss.lambda-url.us-east-2.on.aws/
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

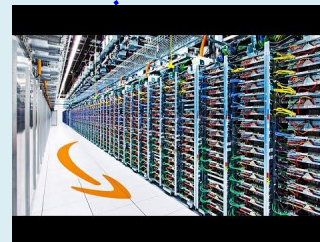
Benchmarking k7hwywasoufmgzefszfgpfhzfq0iejss.lambda-url.us-east-2.on.aws (be patient)
Completed 200 requests
Completed 400 requests
Completed 600 requests
Completed 800 requests
Completed 1000 requests
Completed 1200 requests
Completed 1400 requests
Completed 1600 requests
Completed 1800 requests
Completed 2000 requests
Finished 2000 requests
```

# Demo #2: lambda-based stats

<https://k7hwywasoufmgzefszfgpfhzhfq0iejss.lambda-url.us-east-2.on.aws/>

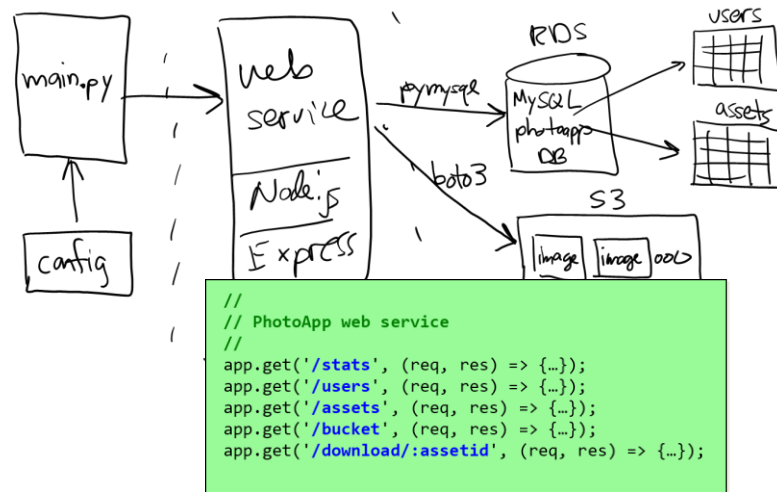


Let's compare our project 02 web service /stats vs. a lambda-based version of /stats



# Serverless computing

- Architects realized the web server is often just a "gateway" to the functions

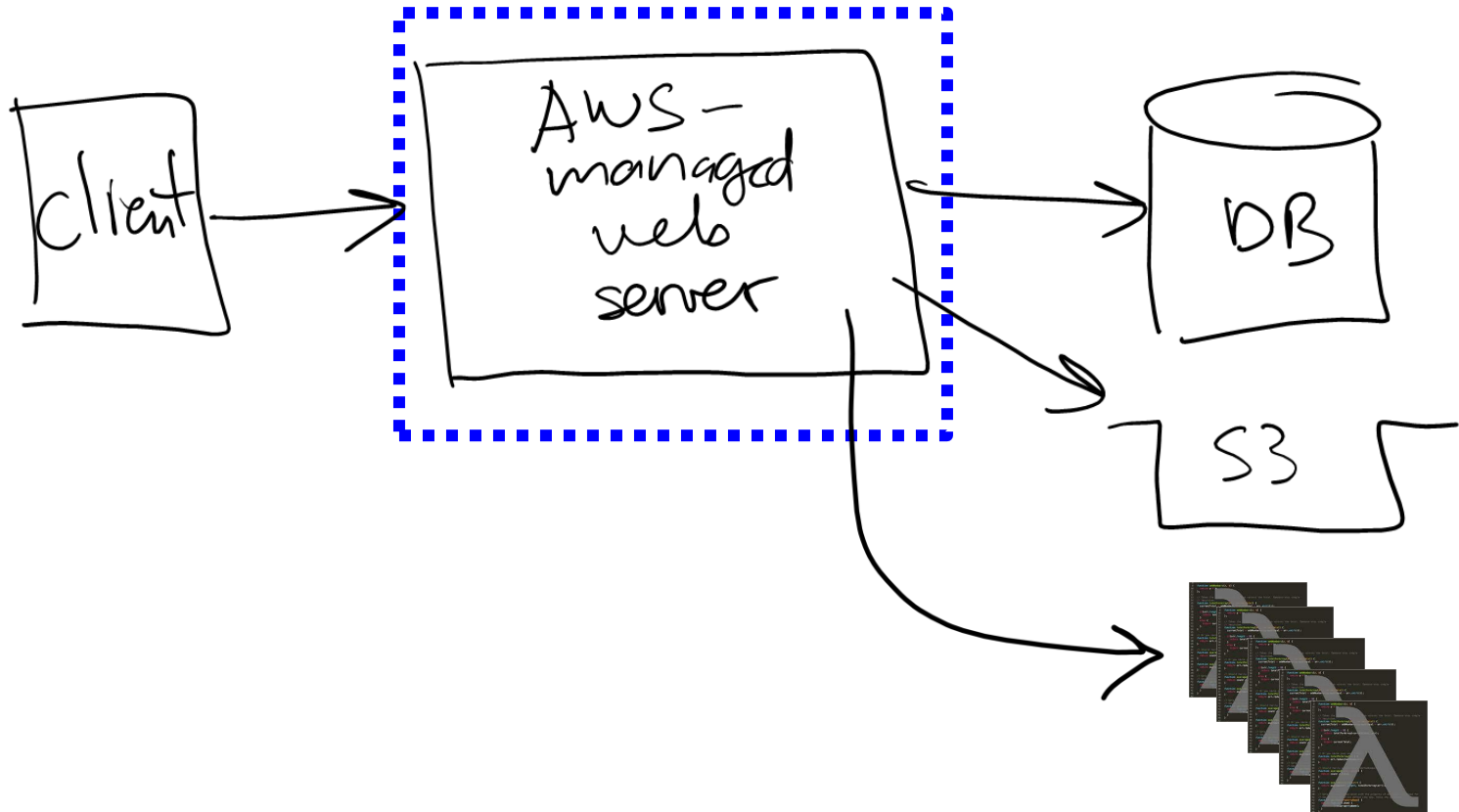


- Break the monolithic code base => functions or microservices...
- ... and let AWS manage the web server!



# Serverless doesn't mean no server

- We still have a web server...
- We just don't manage it



**That's it, thank you!**