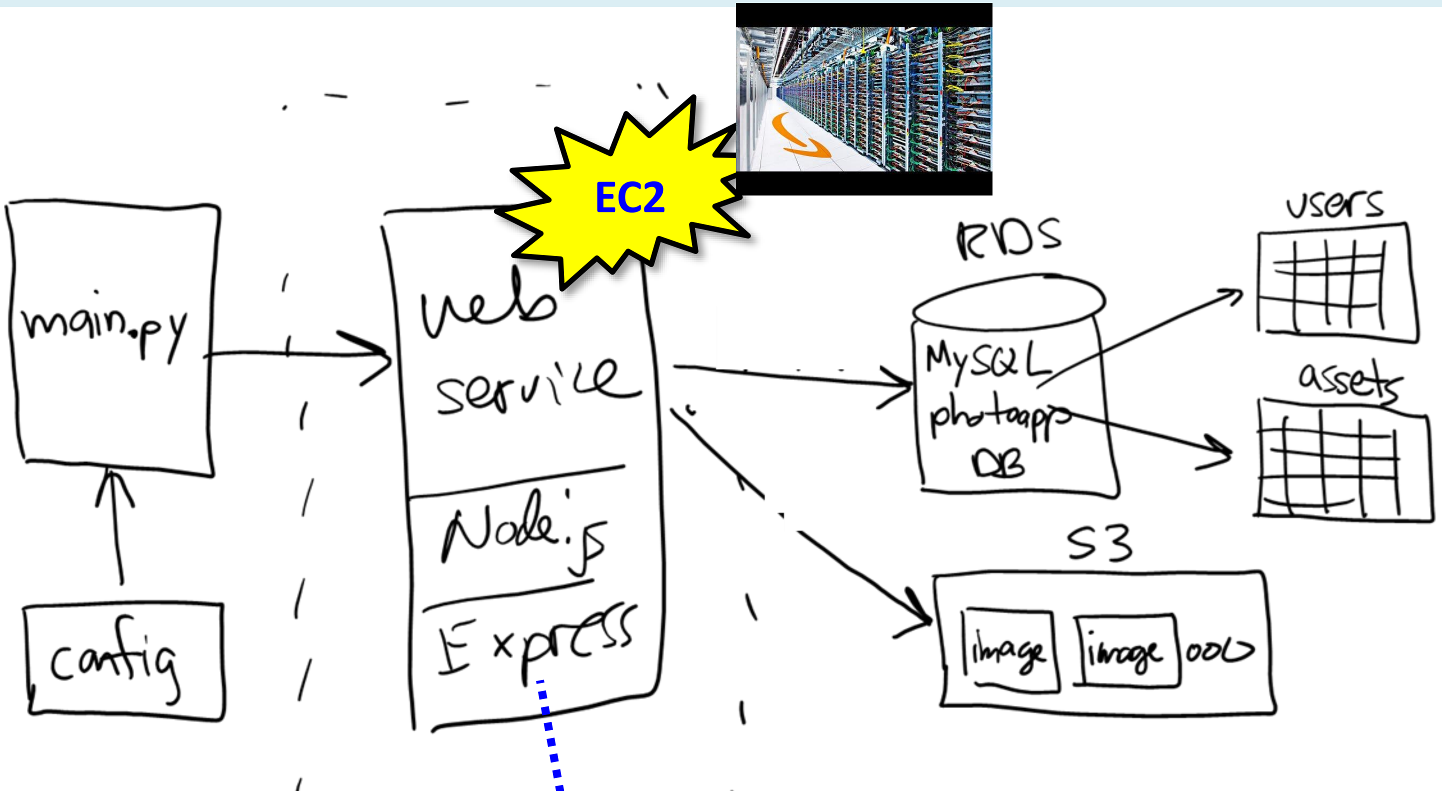
Performance

- Performance in the cloud
- Latency, bandwidth, throughput
- Measuring performance with apache benchmark

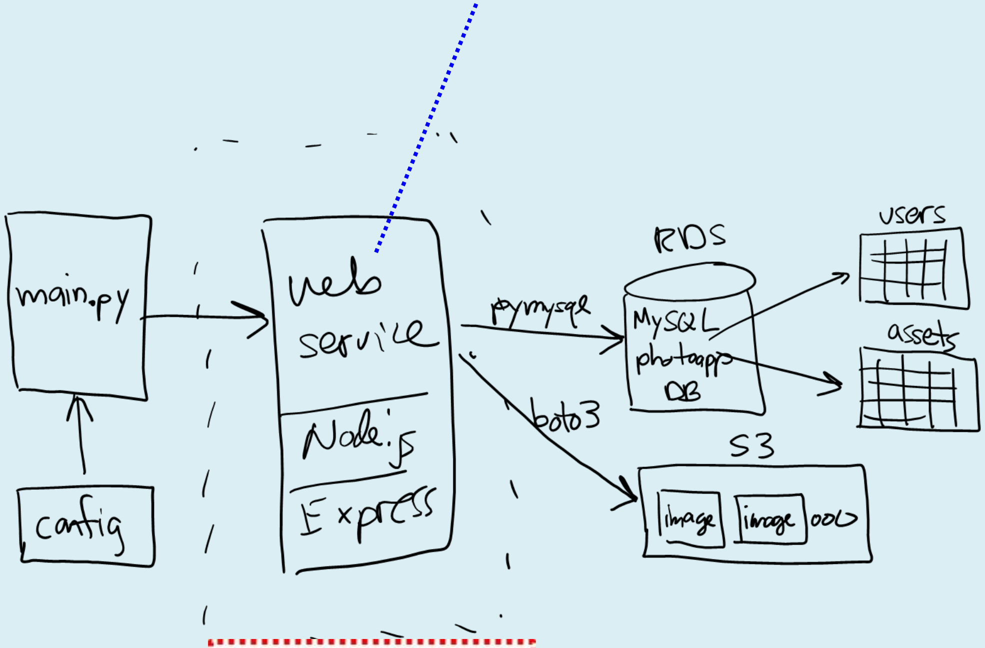
Project 02 --- photoapp web service in AWS

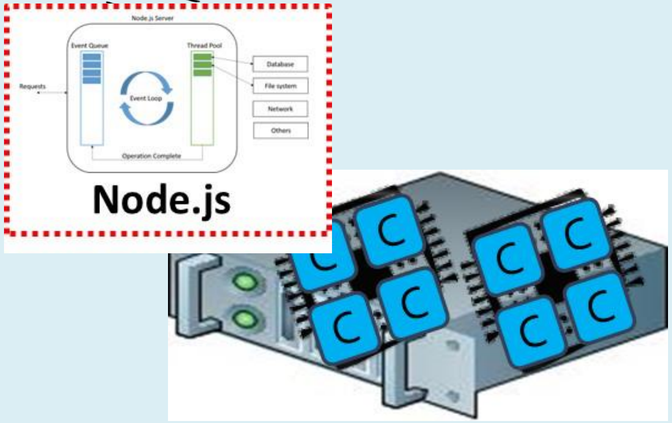


```
//
// PhotoApp web service
//
app.get('/stats', (req, res) => {…});
app.put('/user', (req, res) => {…});
app.get('/users', (req, res) => {…});
app.get('/assets', (req, res) => {…});
app.get('/bucket', (req, res) => {…});
app.get('/image/:assetid', (req, res) => {…});
app.post('/image/:userid', (req, res) => {…});
```

Demo

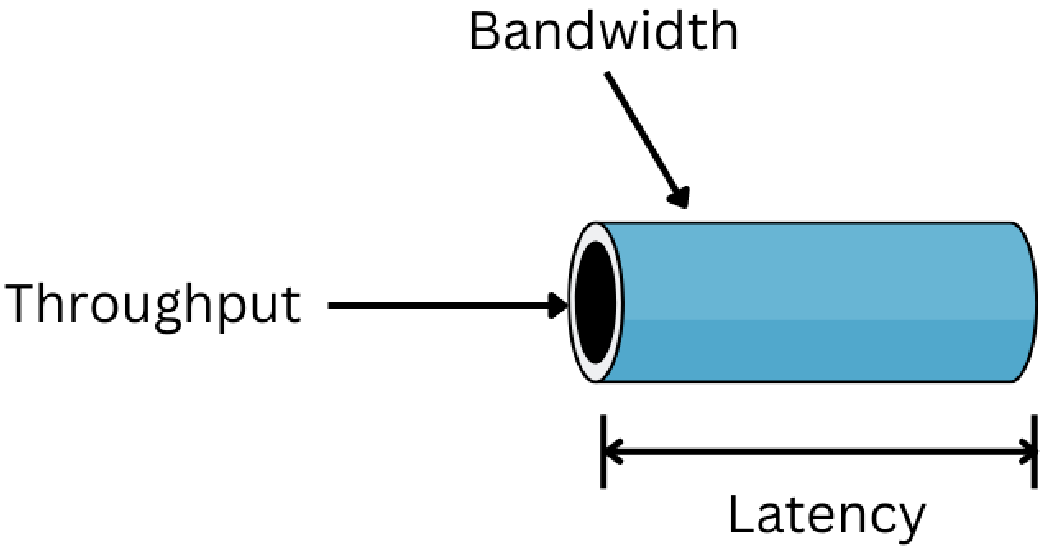
http://nu-cs-msa-s3-photoapp-web-servic-env.eba-njwcmj6i.us-east-2.elasticbeanstalk.com/stats





Network latency vs. bandwidth

- **Latency**: how long does it take? (speed = response time)
- **Bandwidth**: how much data can be transmitted? (volume)
- **Throughput**: how much processing per time unit? (# of users)

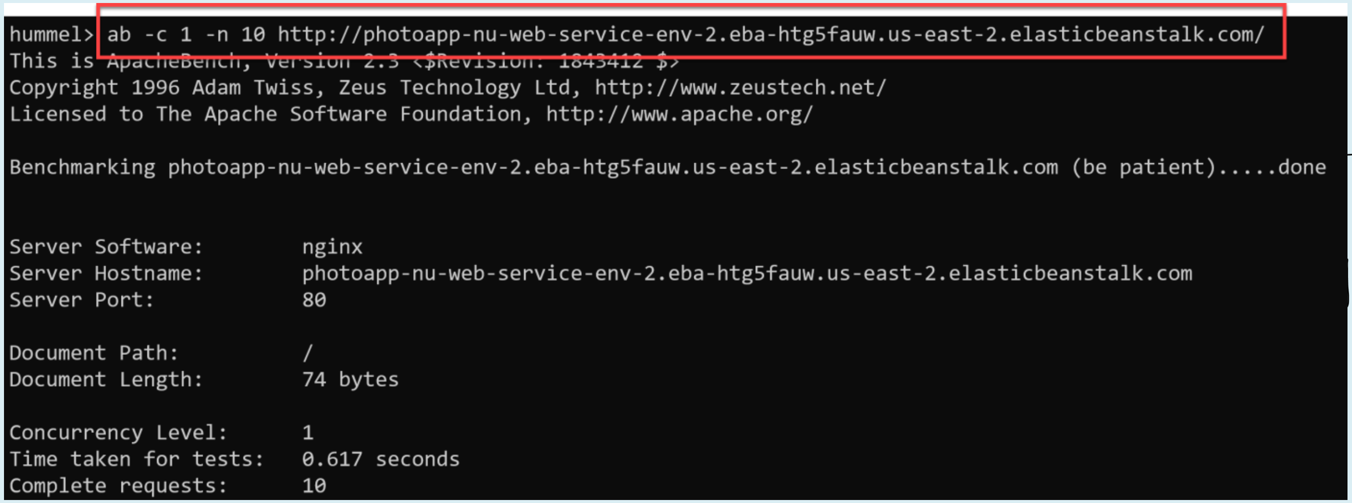


Benchmarking

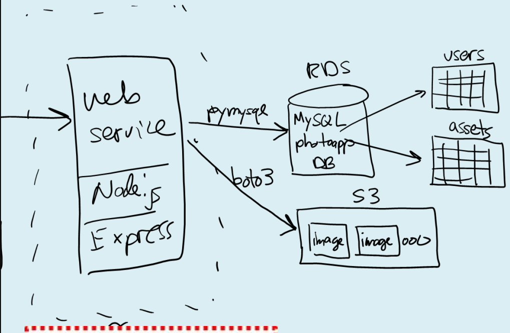
- **Let's benchmark project 02 web service**
- **Baseline:**
  - *cost of a round-trip with no computation*

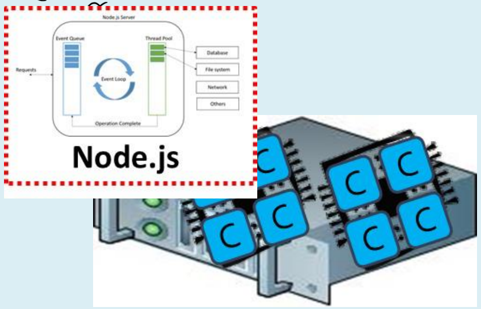
  ==> *time to retrieve "home page"…*

  http://nu-cs-msa-s3-photoapp-web-servic-env.eba-njwcmj6i.us-east-2.elasticbeanstalk.com/

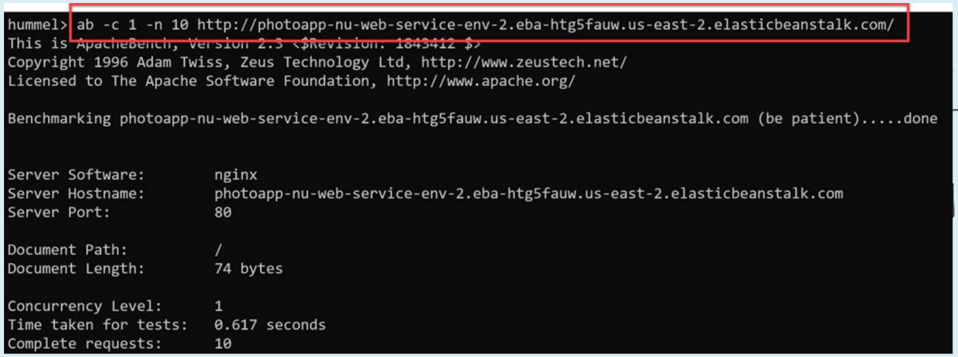


```
hummel> ab -c 1 -n 10 http://photoapp-nu-web-service-env-2.eba-htg5fauw.us-east-2.elasticbeanstalk.com/
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking photoapp-nu-web-service-env-2.eba-htg5fauw.us-east-2.elasticbeanstalk.com (be patient).....done


Server Software:        nginx
Server Hostname:        photoapp-nu-web-service-env-2.eba-htg5fauw.us-east-2.elasticbeanstalk.com
Server Port:            80

Document Path:          /
Document Length:        74 bytes

Concurrency Level:      1
Time taken for tests:   0.617 seconds
Complete requests:      10
```

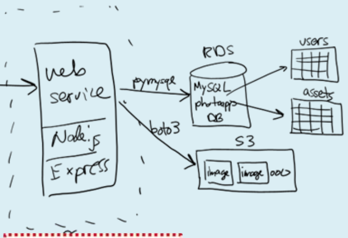


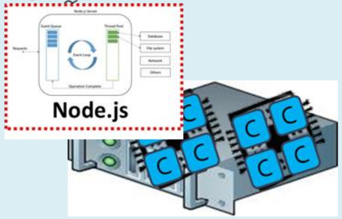


ab == apache benchmark



```
hummel> ab -c 1 -n 10 http://photoapp-nu-web-service-env-2.eba-htg5fauw.us-east-2.elasticbeanstalk.com/
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 photoapp-nu-web-service-env-2.eba-htg5fauw.us-east-2.elasticbeanstalk.com (be patient).....done


Server Software:        nginx
Server Hostname:        photoapp-nu-web-service-env-2.eba-htg5fauw.us-east-2.elasticbeanstalk.com
Server Port:            80

Document Path:          /
Document Length:        74 bytes

Concurrency Level:      1
Time taken for tests:   0.617 seconds
Complete requests:      10
```





- **Reference: https://httpd.apache.org/docs/2.4/programs/ab.html**
  - *Linux: sudo apt-get install apache2-tools*
  - *Mac: already installed (terminal window)*
  - *Windows: download apache (https://www.apachelounge.com/download/), extract /bin/ab.exe*

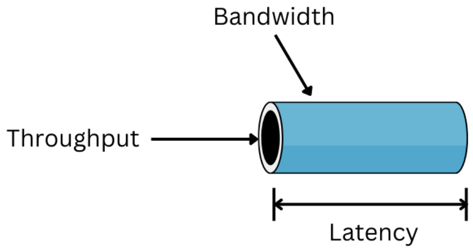
- **Usage: ab -k -c 10 -n 1000 URL**

  -k => keep-alive the TCP connection (cold-start vs. warm-start)

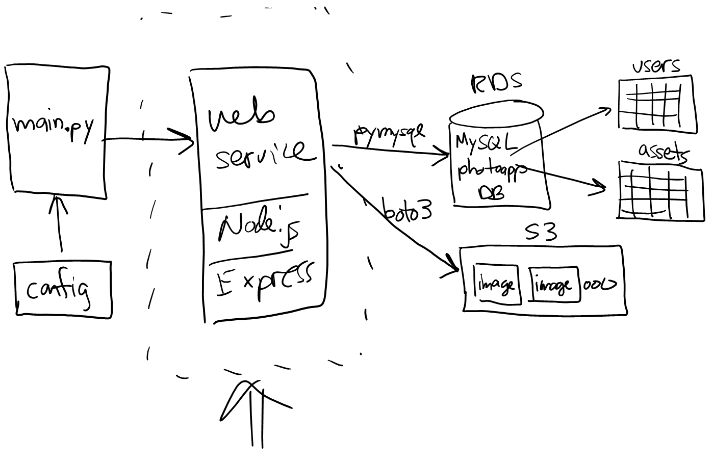
  -c => concurrency level (use this to simulate concurrent users / load on the server)

  -n => # of requests (use this to get a more accurate "average" response time)

Question #1



- **Baseline latency is _____ ms**
- **Best guess --- which API function(s) will have fastest response time / lowest latency?**



```
//
// PhotoApp web service
//
app.get('/stats', (req, res) => {…});
app.get('/users', (req, res) => {…});
app.get('/assets', (req, res) => {…});
app.get('/bucket', (req, res) => {…});
app.get('/image/:assetid', (req, res) => {…});
```

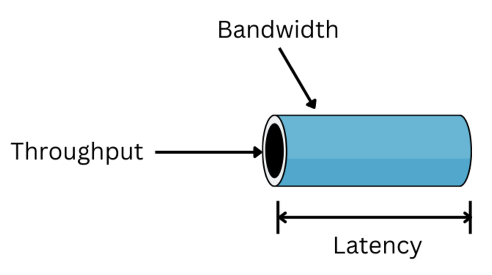
Observations…

1. **/users and /assets are fastest**

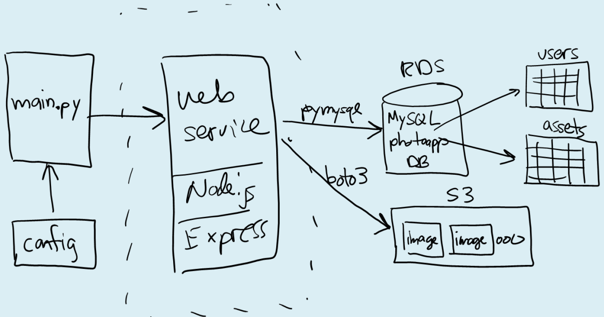
2. **Why? RDS is faster than S3**
   - *DB vs. Web service*

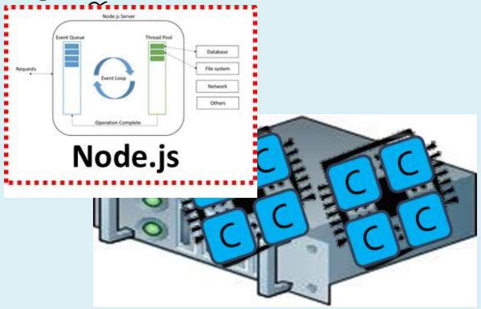
3. **Notice that /users latency is close to baseline**
   - *The trip to AWS is the largest part of the cost…*
   - *Remember to always minimize the # of trips…*

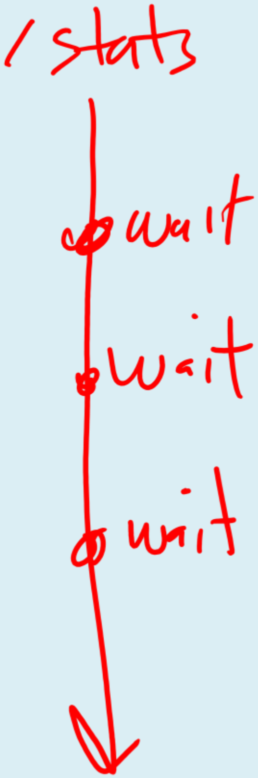
Question #2

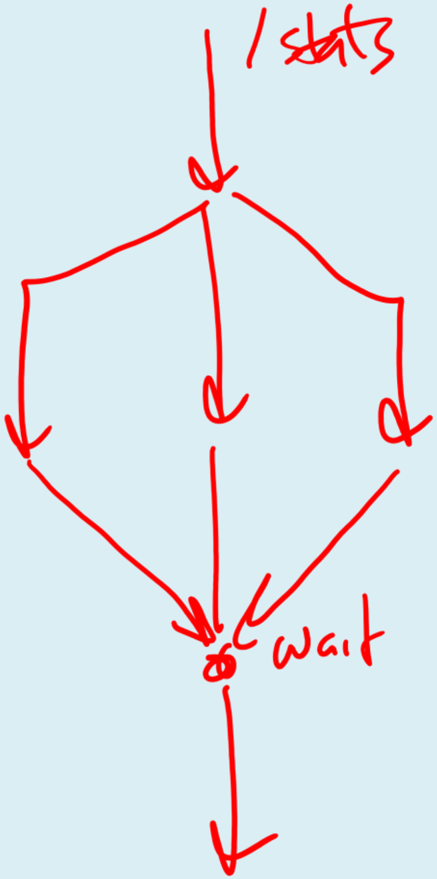


- **Does async programming make a difference on the server?**

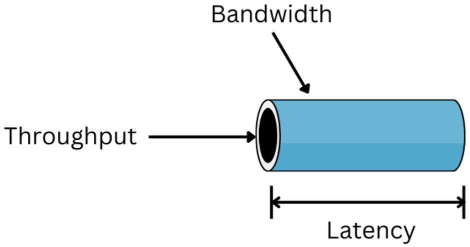




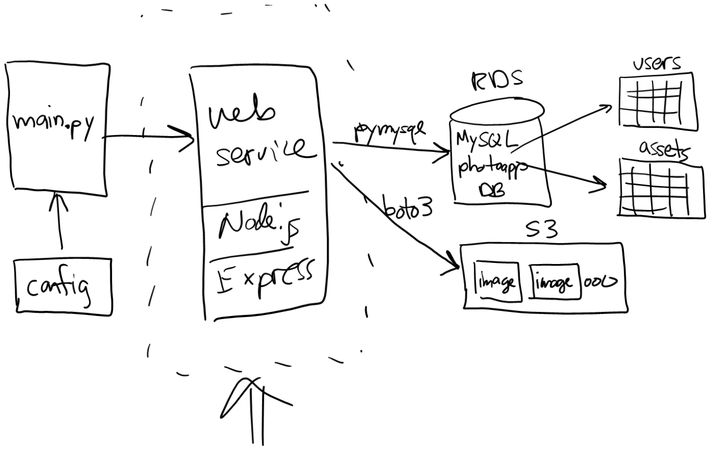




Question #3



- **Let's look at bandwidth vs. latency…**
- **Asset 1004 is 43K, asset 1014 is 1,964K --- 46x bigger**
  - *Time to download image # 1004 is _____ ms*
  - *How much longer will it take to download image # 1014? (______ ms)*



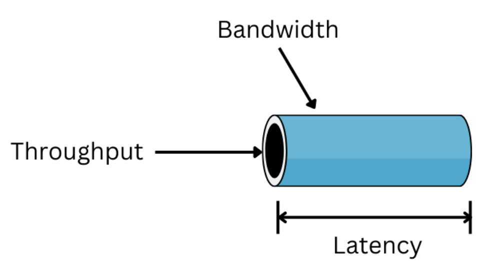
```
//
// PhotoApp web service
//
app.get('/stats', (req, res) => {…});
app.get('/users', (req, res) => {…});
app.get('/assets', (req, res) => {…});
app.get('/bucket', (req, res) => {…});
app.get('/image/:assetid', (req, res) => {…});
```

Observations…

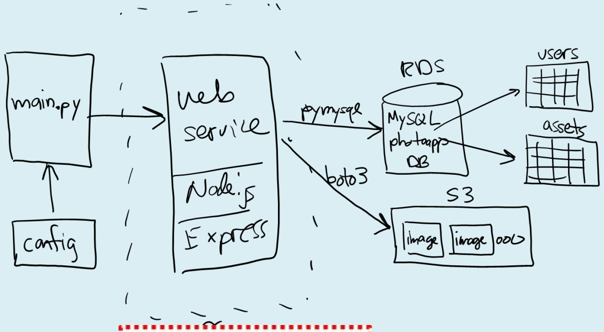
- **You can generally ignore payload size, and focus on minimizing the # of trips**
  - *Downloading an image doesn't take much longer than other functions*
  - *Even though #1007 is 40x larger, only takes 3x time*

==> *not faster to break apart and send separately*

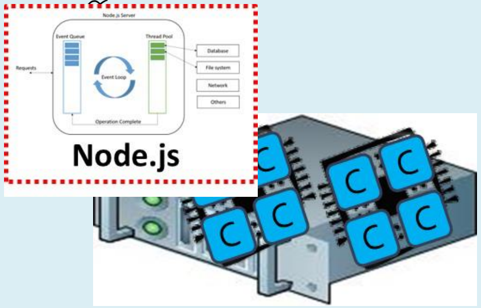
Question #4



- **Throughput…**
- **Node.js is single-threaded and runs on one core. How many users can it support before latency suffers?**
  - *use ab -c option*



```
//
// PhotoApp web service
//
app.get('/stats', (req, res) => {…});
app.get('/users', (req, res) => {…});
app.get('/assets', (req, res) => {…});
app.get('/bucket', (req, res) => {…});
app.get('/image/:assetid', (req, res) => {…});
```

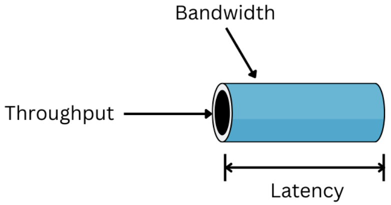


Observations…

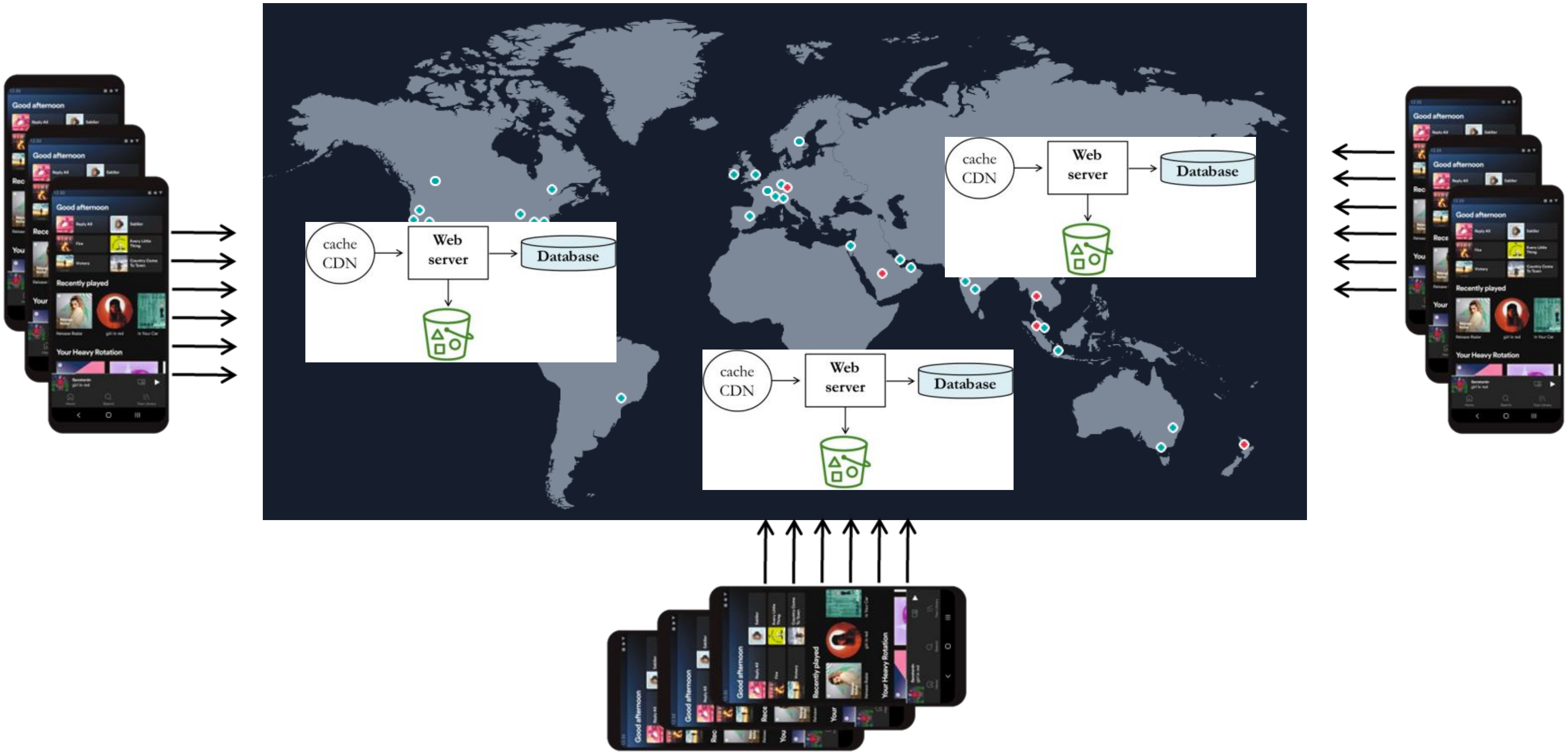
1. **Definitely more than one user…**
   - *Due to asynchronous programming model*

2. **Typically 5-10 users depending on the function**

How to improve latency?

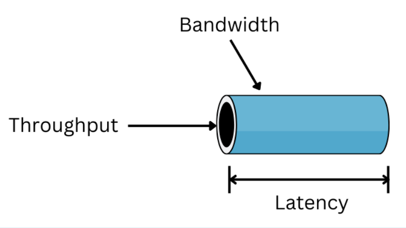


- **Faster internet connection or shorten the distance…**
  - *Cache data in CDN (AWS CloudFront)*
  - *Replicate in different regions (USA, Europe, Asia, Africa, …)*



How to improve bandwidth?

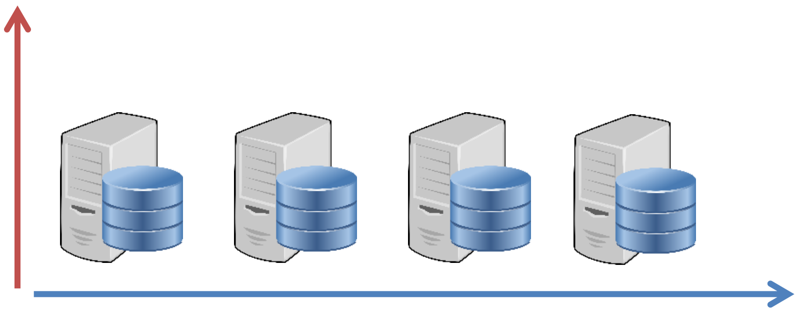
- **Bandwidth is a function of the network connection**



- **You generally don't have much control**
  - *Mobile, Wifi, hard-wired, …*
  - *All different speeds with different reliability*

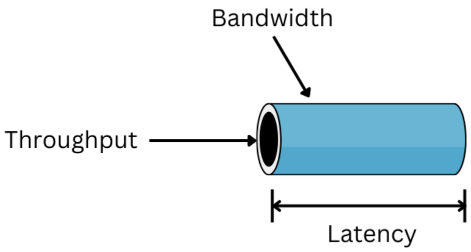
How to improve throughput (# of users)?

Vertical scaling makes your machine(s) bigger and stronger.
Think more cores, RAM.

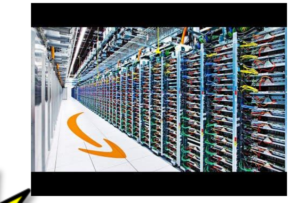


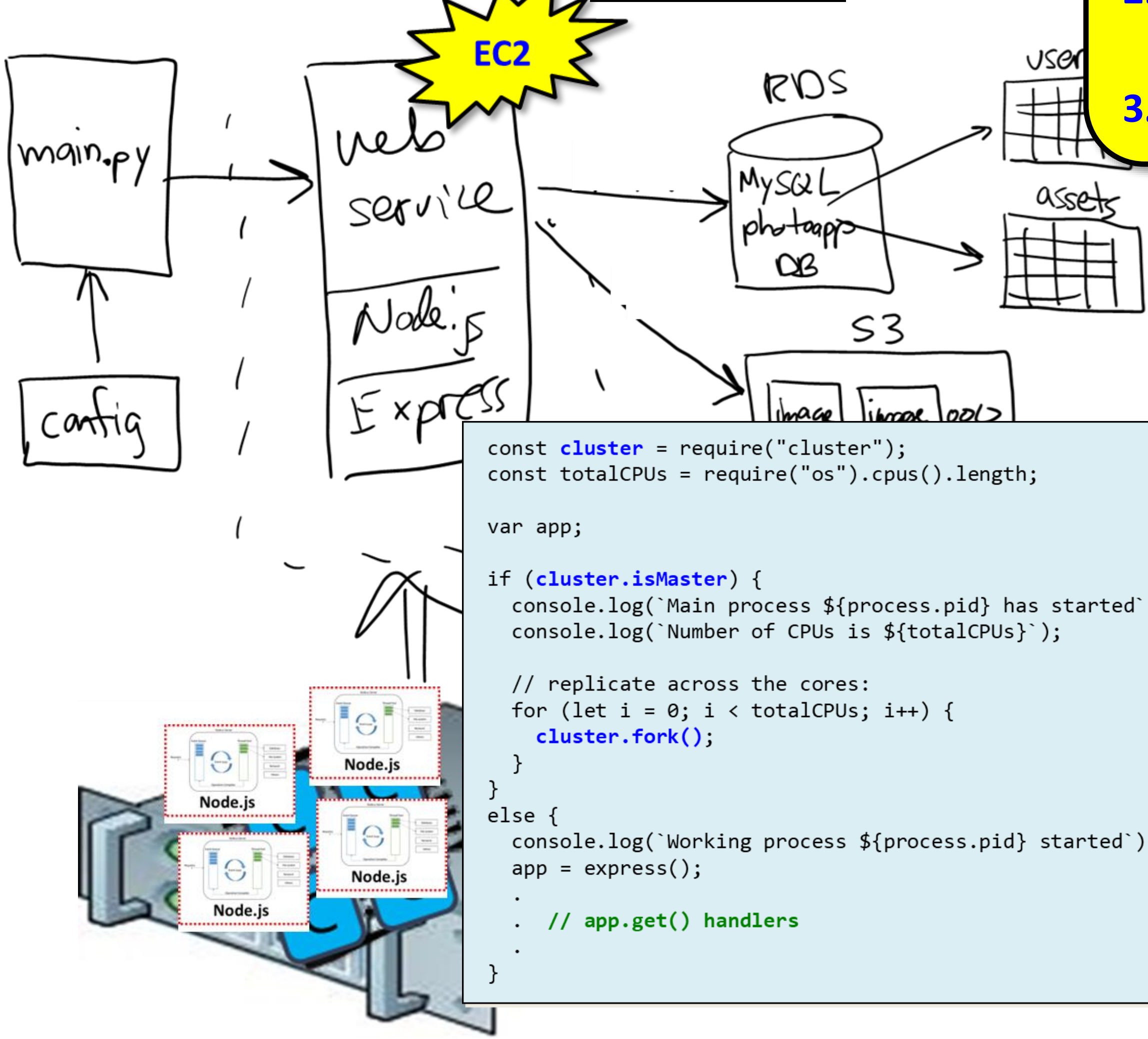
Horizontal scaling adds more machines.
Think of them standing side-by-side.

- **Vertical scaling is easy, rent more cores / RAM**
  - *Instant scalability, but cores may sit idle most of the time (wasting $)*
- **Horizontal scaling supports any # of users**
  - *May take a few minutes for machines to startup (some users wait)*
- **Throughput improved by** ______scaling______



Scaling up project 02 on EB / EC2?





1. Rent a VM with more cores & RAM
2. Rewrite app.js to fork / replicate
3. Deploy new version

```
const cluster = require("cluster");
const totalCPUs = require("os").cpus().length;

var app;

if (cluster.isMaster) {
  console.log(`Main process ${process.pid} has started`);
  console.log(`Number of CPUs is ${totalCPUs}`);

  // replicate across the cores:
  for (let i = 0; i < totalCPUs; i++) {
    cluster.fork();
  }
}
else {
  console.log(`Working process ${process.pid} started`);
  app = express();
  .
  .  // app.get() handlers
  .
}
```

That's it, thank you!