Node.js and Backend Development

Shan-Hung Wu CS, NTHU

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

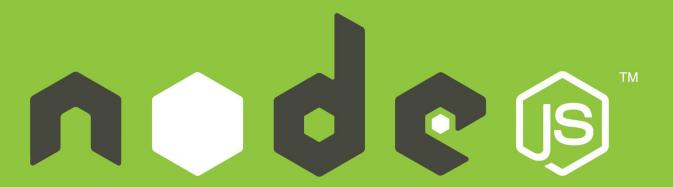
Clone hello-node

\$ npm install --save yargs

- Yargs
 - Command line interface (CLI) tools

• Usage:

\$ node src/main.js <command> [options]



- Javscript runtime engine
- Why Node.js in the backend?
 - Event-based and asynchronous I/O; very fast
 - NPM and large ecosystem

Node Runtime

Global objects:

- window → global (scope)
- document → process (.env, .argv(), .exit()
 etc.)
- os, fs, module.exports
- Node.js <u>supports ES6</u>
 - No need for Babel
 - No import (use require () instead)
- See API Docs

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

WeatherMood Posts on Node.js

Why fast as an backend engine?

Asynchronous I/O

```
// A.js
fs.readReadFile('file.name', (err, data) => {
  ... // 3
}); // 2
                                           Time Async
http.get('url', res => {
  ... // 5
}); // 4
                        CPU
                               A6
                                      A3
... // 6
                       1/01
                                 A2

    Shortens exec

                       1/02
                                                         Sync
                                    A4
  time
                                                        Time
                       CPU A1
   If with parallel
                                      A3
                                                         A6
     I/O channels
                       I/O 1
                                 A2
                       1/02
                                              A4
```

```
// User B.js
// User A.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
... // 3
                                ... // 3
}); // 2
                              }); // 2
http.get(..., () => {
                              http.get(..., () => {
... // 5
                                ... // 5
                              });
// 4
}); // 4
... // 6
               CPU
               1/01
               1/0 2
```

```
// A.js
                              // B.js
... // 1
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
                                ... // 3
... // 3
}); // 2
                              }); // 2
http.get(..., () => {
                             http.get(..., () => {
... // 5
                                ... // 5
                              });
// 4
}); // 4
... // 6
               CPU
                   A1
               I/O 1
                        A2
               1/02
```

```
// A.js
                              // B.js
... // 1
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
                                ... // 3
  ... // 3
}); // 2
http.get(..., () => {
                              http.get(..., () => {
... // 5
                                ... // 5
                              });
// 4
}); // 4
... // 6
               CPU
                   A1
               I/O 1
                        A2
               1/0 2
                           A4
```

```
// A.js
                              // B.js
... // 1
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
                                ... // 3
  ... // 3
}); // 2
http.get(..., () => {
                              http.get(..., () => {
... // 5
                                ... // 5
                              });
// 4
}); // 4
               CPU
                   A1 A6
               I/O 1
                        A2
```

A4



```
// A.js
                                // B.js
fs.readReadFile(..., () =>
                                fs.readReadFile(..., () => {
  ... // 3
                                  ... // 3
}); // 2
http.get(..., () => {
                                http.get(..., () => {
  ... // 5
}); // 4
... // 6
                        A6
                CPU
                    A1
                           B1
                I/O 1
                         A2
                1/0 2
                            A4
```

```
// A.js
                                // B.js
fs.readReadFile(..., () =>
                                fs.readReadFile(..., () => {
  ... // 3
                                  ... // 3
}); // 2
http.get(..., () => {
                                http.get(..., () => {
  ... // 5
}); // 4
... // 6
                CPU
                    A1
                        A6
                           B1
                1/01
                         A2
       A3
                1/0 2
                            A4
```

```
// A.js
                                 // B.js
fs.readReadFile(..., () => { \Rightarrow fs.readReadFile(..., () => {
  ... // 3
}); // 2
http.get(..., () => {
                                 http.get(..., () => {
 ... // 5
                                   ... // 5
                                 });
// 4
}); // 4
... // 6
                        A6
                 CPU
                     A1
                            B1
                1/01
                          A2
                                  B2
       A3
                1/0 2
                             A4
```

```
// A.js
                               // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
                                 ... // 3
  ... // 3
}); // 2
                               http.get(..., () => {
http.get(..., () => {
 ... // 5
                               });
// 4
}); // 4
... // 6
                CPU
                       A6
                    A1
                           B1
               1/01
                         A2
                                B2
      A3
               1/0 2
                           A4
                                       B4
```

```
// A.js
                              // B.js
... // 1
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
                                 ... // 3
  ... // 3
}); // 2
                              }); // 2
http.get(..., () => {
                              http.get(..., () => {
... // 5
                                 ... // 5
                               });
// 4
}); // 4
... // 6
                       A6
                CPU
                   A1
                          B1
                              B6
               1/01
                        A2
                                B2
      A3
               1/0 2
                           A4
                                      B4
```

```
// A.js
                               // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
... // 3
                                  ... // 3
http.get(..., () => {
                               http.get(..., () => {
... // 5
                                 ... // 5
                               });
// 4
}); // 4
... // 6
                CPU
                    A1 A6
                           B1
                               B6
                                   A3
                I/O 1
                         A2
                                B2
                1/0 2
                            A4
                                       B4
```

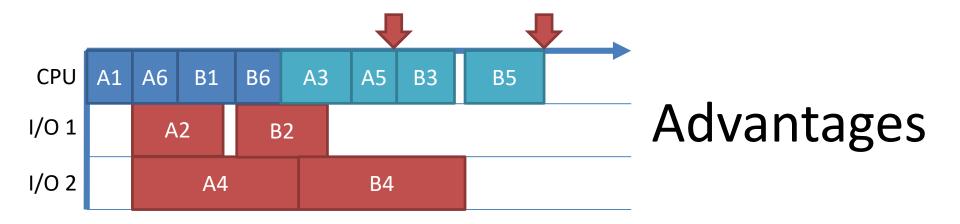
```
// A.js
                                // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
... // 3
                                  ... // 3
http.get(..., () => {
                               http.get(..., () => {
... // 5
                                  ... // 5
                               });
// 4
}); // 4
... // 6
                       A6
                CPU
                    A1
                           B1
                               B6
                                   A3
                I/O 1
                         A2
                                 B2
       A5
                1/0 2
                            A4
                                        B4
```

```
// A.js
                                // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
... // 3
                                  ... // 3
http.get(..., () => {
                                http.get(..., () => {
 ... // 5
                                  ... // 5
                                });
// 4
}); // 4
... // 6
                        A6
                CPU
                    A1
                            B1
                               B6
                                   A3
       B3
                I/O 1
                         A2
                                 B2
       A5
                1/0 2
                            A4
                                        B4
```

```
// A.js
                                // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
                                  ... // 3
  ... // 3
}); // 2
http.get(..., () => {
                                http.get(..., () => {
                                  ... // 5
                                });
// 4
... // 6
                CPU
                        A6
                            B1
                               B6
                                   A3
                                        A5
                I/O 1
                          A2
                                 B2
       B3
                1/0 2
                            A4
                                        B4
```

```
// A.js
                               // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
  ... // 3
}); // 2
http.get(..., () => {
                               http.get(..., () => {
 ... // 5
                                  ... // 5
                               });
// 4
}); // 4
... // 6
                CPU
                       A6
                           B1
                               B6
                                   A3
                                       A5
                                           B3
               I/O 1
                         A2
                                B2
                1/0 2
                            A4
                                       B4
```

```
// A.js
                                // B.js
fs.readReadFile(..., () => { fs.readReadFile(..., () => {
  ... // 3
                                  ... // 3
}); // 2
http.get(..., () => {
                                http.get(..., () => {
  ... // 5
}); // 4
... // 6
                CPU
                        A6
                            B1
                               B6
                                   A3
                                        A5
                                           B3
                                                 B5
                I/O 1
                         A2
                                 B2
                1/0 2
                            A4
                                        B4
```



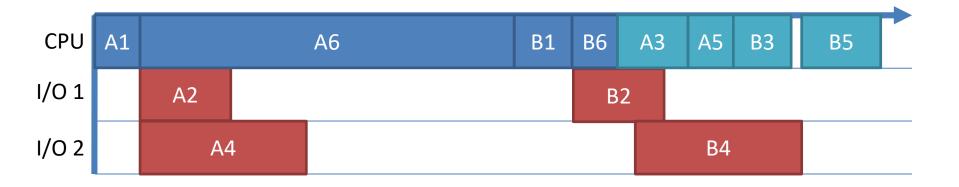
- Interleaved exec of multiple code blocks
- Short response time
- Pipelining between CPU and I/Os
 - High throughput
- Avoids overhead of concurrency control (e.g., locks, thread scheduling, etc.)
 - Higher throughput than muti-threaded engines

Multi-Core Machines?

- <u>Cluster</u> module that runs one Node.js process per core
- Scales up *linearly*, if
 - Workload can be partitioned evenly by processes
 - I/Os are not saturated

When **Not** to Use Node.js?

CPU-bound tasks



- Or, use child_process.exec() (or spawn() for streaming output from child)
 - Foreground core + background cores

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

NPM

Package manager

```
$ npm init
$ npm install --[save|save-dev] <pkg-name>
var m = require('module');
```

Packages vs. modules?

--save vs. --save-dev?

Given dependency tree:

 People who clone/fork your package will download the following packages:

```
{Pkg 1, Pkg 2, Pkg 3} // via 'npm install'
```

Command Line Interface (CLI)

- Option 1: to parse process.argv yourself
- Option 2: yargs.argv

```
$ npm install --save yargs
```

- Defines commands and their options
- Help
- Sanity checks

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

Debugging

```
// in src
debugger;

$ node debug src/main.js <command> [options]
debug> n
debug> c
debug> repl
```

Chrome inspector equivalent?

Node Inspector (Experimental)

```
$ node --inspect --debug-brk src/main.js ...
```

 Then paste "chrome-devtools://.." into Chrome

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

Node.js as Backend Engine

- Typically, web workloads are I/O bound
 - High throughput
 - Low latency
- If designed well, API requests can be easily partitioned across multiple processes
 - Scales linearly

Clone weathermood-server

Checkout the file branch

```
$ npm install --save express body-parser
$ npm install -save-dev nodemon
```

Express

- A web app framework based on Node.js
- Body-Parser
 - An Express middleware for parsing request body
- Nodemon
 - Auto-restarter (remember "webpack -w"?)

Web App Backend Development

1. Prepare static resources

- E.g., *.html, *.css, client-side JS, images, etc.
- In dist/ of branch server-file of weathermood

2. Define API for AJAX calls

- Dynamic resources, i.e., API for AJAX calls
- 3. Code API
- 4. Deploy web app to hosting server(s)
 - Web App ≠ web server(s)
 - For now, local machine as development server

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

Defining APIs

- E.g., listing posts, create a post, vote, etc.
- But HTTP defines only 4 methods
 - GET, POST, PUT, DELETE
- Option 1: define new "verbs"
 - Always POST to the same URL
 - Define body (can follow SOAP)
- Option 2: define new "nouns"
 - E.g., vote → POST vote
 - Different URLs for different nouns/resources
 - REST makes your backend simple and scalable

URL Mappings

URLs\Methods	GET	POST	PUT	DELETE
http://\${host} /\${resource}s	List all resources (satisfying query "?").	Create a new resource (unknown ID).	Replace the entire collection.	Delete the entire collection.
http://\${host} /\${resource}s /\${id}	Read a specific resource.	Treat this resource as a collection and create a new member.	Update this resource or create one (known ID).	Delete this resource.

- Each resource type maps to 2 URL types
 - Collection URLs vs resource URLs
- List post: GET /posts?seatchText=...&...
- Create post: POST /posts
- Vote "clear": POST /posts/\${id}/clearVotes

HTTP Response Codes

- GET:
 - 200 OK (with body)
- POST:
 - 200 OK (with body)
 - 201 Created (with header Location showing ID)
 - 204 No Content
- PUT and DELETE:
 - 200 OK (with body)
 - 204 No Content
- Error:
 - 400 Bad Request, 401 Unauthorized, 404 Not Found, or 500 Internal Server Error

Requirements

- Stateless: session state (e.g., shopping cart) cannot be kept in client side
 - Session state sent with requests using cookies
 - So, requests from clients can be partitioned easily (scalable web servers)
- GET requests much have no side effect
 - Allows proxy nodes in the routing path
- PUT and DELETE requests must be idempotent: duplicated requests has no effect

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

Express

- Normal flow: routers → model calls
 - Request parsing: req.query, req.params, req.body
 - Responses: res.status(), res.json(),
 res.sendStatus()
- Error flow: throw → error handler middleware → responses

Middleware

- Error handling
- Serving static files
- Filtering
- Logging
- Validation
- Authentication and authorization
- and more...

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

Debugging

• Nodemon:

\$ npm run watch

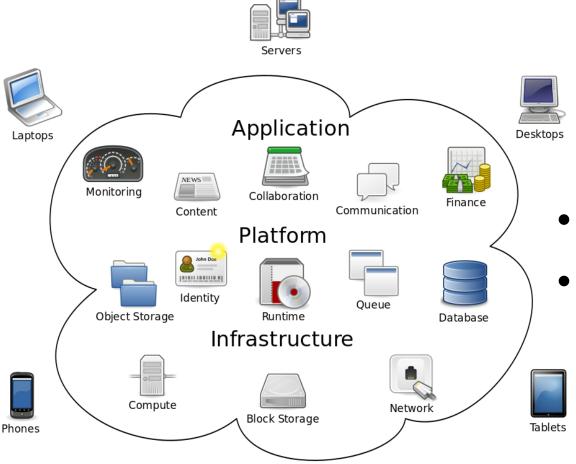
• Postman



Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

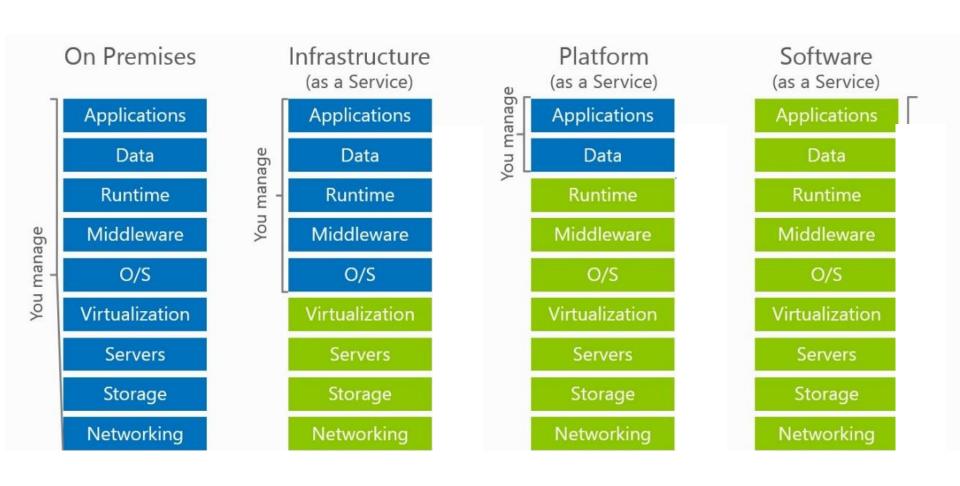
Cloud Computing



Cloud computing

- Not only services
 - Also platforms or infrastructure
 - Pay only what you used

laaS, PaaS, and SaaS



Web App Backend: PaaS or laaS?

```
Flexibility?
Cost?
Performance?

Ease of Migration?
```

Development Efficiency

- Iterations: bug fixes, updates, finding PMF, etc.
- Time is money!

- laaS: high management cost, low elastisity
- PaaS: low cost and elastic, but
 "Well, it works on my computer..."

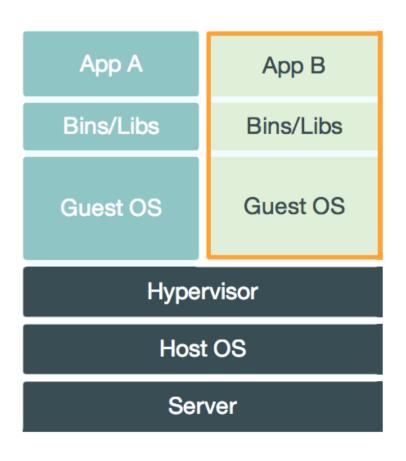
Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk

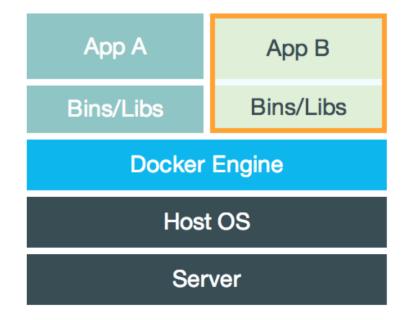


- PaaS
- You define your runtime

Virtual Machines vs. *Containers*

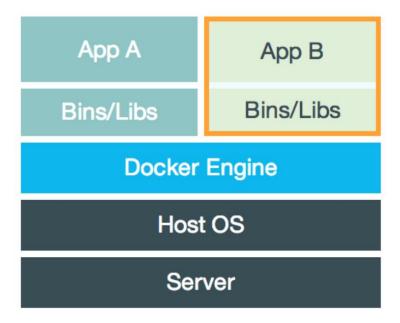


- Lightweight, elastic
- Consistent runtime



"Dockerizing" an App

- 1. Build a Docker Image
- 2. Upload the image to targeted server(s)
- 3. Launch *container* from image



Example: Local Development Server

Install <u>Docker Community Edition</u> first

```
// in project folder
$ vim Dockerfile
$ vim .dockerignore
$ docker build -t <name:tag> .
$ docker images
$ docker run -p 80:8080 -d <image>
$ docker <stop|restart> <container>
$ docker ps [-a]
$ docker system prune
```

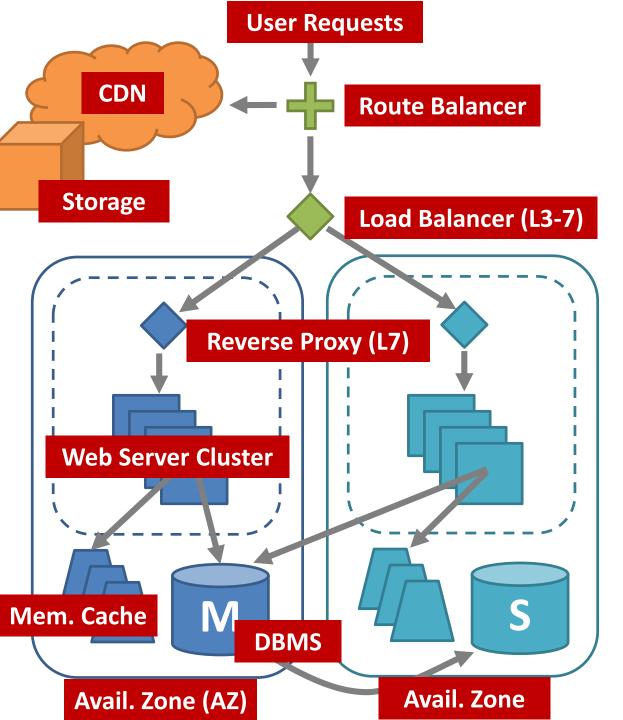
Watch out Data Loss!

- Modifications to filesystem are *local* to a container
 - Gone if container deleted
- If necessary, use <u>data volumes</u> to persists data across containers
 - Basically, specially-designated directories

Beyond development server?

Outline

- Node.js
 - Events and asynchronous I/O
 - NPM and CLI tools
 - Debugging
- Backend Development using Express
 - RESTful API
 - Express: routers and middleware
 - Testing and debugging with Postman
- Deployment
 - Cloud computing and Docker
 - Amazon Elastic Beanstalk



"Typical" Environment

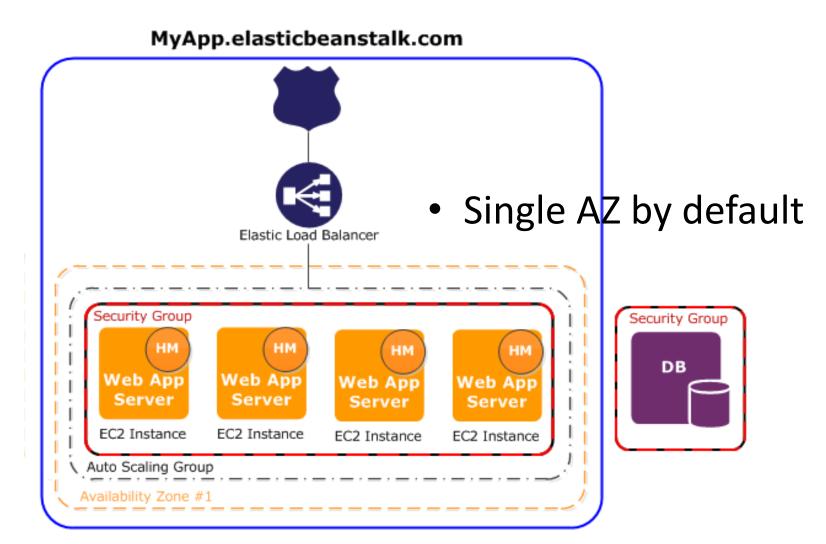
- DDoS protection
- Static vs. dynamic resources
- High availability
- LB, caching, acceleration (SSL), etc.
- Auto-scaling groups
- Scale up vs. out
- Slave DBMS can serve read-only requests

Solutions

Node	Amazon (AWS)	Alternatives	
Route Balancer	Route 53	Cloudflare	
CDN	CloudFront	Cloudflare, Google Cloud CDN	
Storage	Simple Storage Service (\$3)	HDFS	
Load Balancer	Elastic Load Balancing (<i>ELB</i>)	Nginx	
Reverse Proxy		Nginx	
Web Server	Elastic Compute Cloud (<i>EC2</i>)	Heroku	
Mem. Cache	ElastiCache	Redis, Memcached	
DBMS	Relational DB Service (RDS)	PostgreSQL, MySQL, MongoDB	

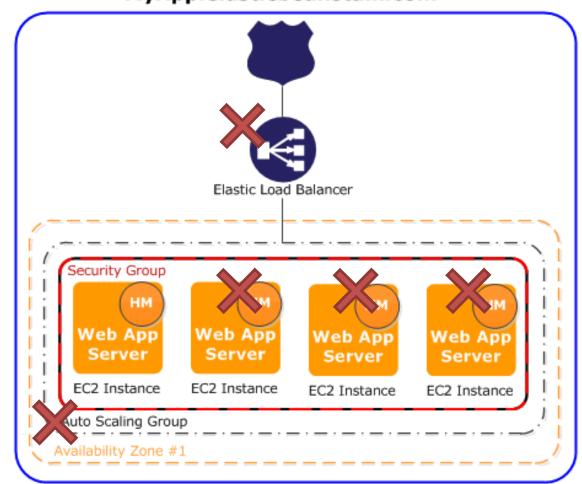
- Web app on EC2 (with Docker and Nginx preinstalled)
- But what about the rests?

AWS Elastic Beanstalk



Staging Environment

MyApp.elasticbeanstalk.com





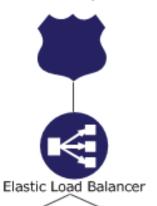
Instructions

- 1. Create an AWS account
 - Free-tier for the first year, and free credits from AWS Educate
 - Credit card needed (one per group); no immediate charge
- 2. Install Elastic Beanstalk CLI
 - Install Python runtime (e.g., <u>Aanaconda</u>) first

```
$ pip install --upgrade awsebcli
$ eb -version

// in project folder
$ eb init // create eb application
$ eb create [--single] // create environment
$ eb terminate <env>
// upates
$ git commit
$ eb deploy <env> // deploys the latest commit
$ eb use <env> // env for current branch
```

User Requests



"Typical" Environment

How to enable

Add AZs easily in console

CDN? нм нм нм нм Web App Web App Web App Web App Server Server Server Server CloudFront Distribution EC2 Instance EC2 Instance EC2 Instance EC2 Instance Auto Scaling Group Auto Scaling Group Database Database Database Backups RDS DB Instance RDS DB Instance RDS DB Instance S3 Bucket Standby Standby \Availability Zone #2 \ Availability Zone #1

Cache Control

 CloudFront and client browsers use Cache-Control response header to cache static files

```
// in server.js
app.use(express.static('dist', {
 maxAge: ... // in ms. For client browsers and proxies
} ) );

    Be careful about file versioning

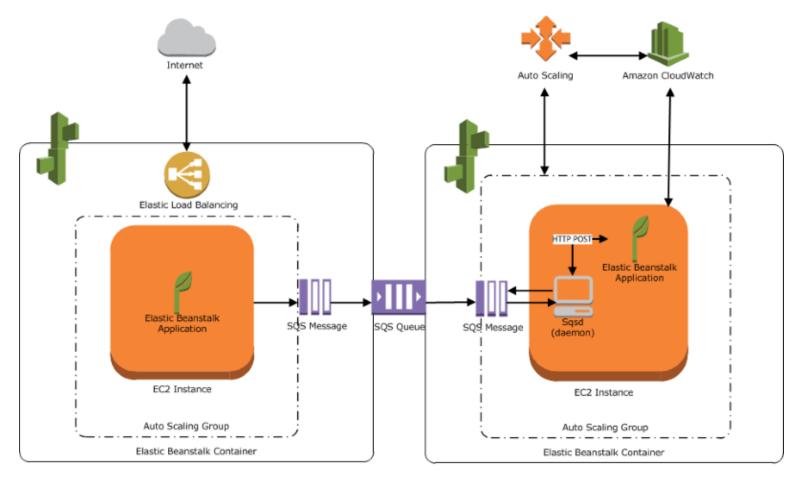
// or
app.use(express.static('dist', {
  setHeaders: (res, path, stat) => {
    // in seconds. For proxies only
    res.set('Cache-Control', 'public, s-maxage=...');
```

Readings (Optional)

- HTTPS
 - Production environments
 - Staging environments
- Custom domain names

Worker Environments

For CPU-bound tasks (e.g., data analytics)



Web Server Environment Tier Worker Environment Tier 70

Assignment: TODOs

- RESTful API
- Server-side routers, model, etc.
- Client-side AJAX calls
- AWS Deployment (one per group)

