Node in Production

Module 2: Node Production Preparations

Azat Mardan @azat_co

Node University

Securing Your App with Environment Variables

Why not store in code?

- Accidentally publishing sensitive information [1]
- GitHub/Bitbucket/GitLab might get and *were* hacked [2]
- Code is less flexible to change in a cloud environment

---

[1] For example, In major goof, Uber stored sensitive database key on public GitHub page and How Homakov hacked GitHub and how to protect your application by Peter Nixey
Raw

[2] For example, GitHub Security Update: Reused password attack

Bad practice:

server.js:

```
const apiKey = 'BFED4C44-69DF-49F5'
const apiSecret = 'AA719195-43D2-4C32-96B9-BBD6CB9400AA'
const databaseUrl = 'mongodb://admin:EFC429E2-CB84-458F-97BB-CD9BDCCDB1B1@52-53-154-247:27017'

const dbConnection = db.connect(databaseUrl) // connect to a database
oauth.get(apiKey, apiSecret, accessCode, callback) // fetch protected resource using OAuth
```

Having config .json files (or .js) is still not good since most likely they are stored together with the code

Bad practice:

- config.json
- api/server/datasources.json (Loopback)
- source code (.js)

Use Env Vars

One-liner:

```
MY_ENV_VAR=hello node -e "console.log(`I see: ${process.env.MY_ENV_VAR}`)"
```

In session:

```
export MY_ENV_VAR=hello
node -e "console.log(`I see: ${process.env.MY_ENV_VAR}`)"
```

Use Env Vars (cont)

server.js:

```
const apiKey = process.env.API_KEY
const apiSecret = process.env.API_SECRET
const databaseUrl = process.env.DATABASE_URL

const dbConnection = db.connect(databaseUrl) // connect to a database
oauth.get(apiKey, apiSecret, accessCode, callback) // fetch protected resource using OAuth
```

A lot of libraries read from NODE_ENV

```
NODE_ENV=production
NODE_ENV=development
```

Ways to populate env vars:

- AWS EC2 User Data
- Dockerfile
- AWS CloudFormation blueprint and/or parameters
- CLI or Web interfaces of your cloud provider (AWS, Heroku, etc.)

Separating Development and Production Dependencies

The more dependencies you have, the higher the risk of an attack or a bug. Why have dependencies which you don't need?

Separate dependencies which you only need for development and don't ship them to production!

```
"dependencies": {
  "babel-register": "6.11.6",
  "body-parser": "1.13.2",
  "compression": "1.5.1",
  "errorhandler": "1.4.1",
  "express": "4.13.1",
  "express-validator": "2.13.0",
  "hbs": "4.0.0",
  "mongodb": "2.2.25",
  "morgan": "1.6.1",
  "react": "15.5.4",
  "react-dom": "15.5.4"
},
"devDependencies": {
  "axios": "0.13.1",
  "babel-core": "6.10.4",
  "babel-jest": "13.2.2",
  "babel-loader": "6.4.1",
  "babel-preset-react": "6.5.0",
  "node-dev": "3.1.3",
  "webpack": "2.4.1"
}
```

Install `dependencies` + `devDependencies`:

```
npm i
```

Install only `dependencies`:

```
npm i --production
```

Clear `devDependencies`:

```
npm prune --production
```

Or remove all and reinstall:

```
rm -rf node_modules
npm i --production
```

Locking in Dependencies Versions

Exact (recommended):

```
"express": "4.15.2"
```

Ranges (bad) could be 4.20, 4.15.4, etc.:

```
"express": "^4.15.2"
"express": "~4.15.2"
"express": "~4.x"
```

Any latest (very bad) could be 3.0, 4.15, 5.1:

```
"express": "*"
```

More: https://docs.npmjs.com/misc/semver

You are not safe yet!

Dependencies of dependencies often are not locked!

```
npm i
code@1.0.0 /Users/spq201/Code/node-in-production/code
└─┬ express@4.15.2
  ├─┬ accepts@1.3.3
  | ├─┬ mime-types@2.1.15
  | | └── mime-db@1.27.0
  | └── negotiator@0.6.1
```

Solution to lock them all

- Shrinkwrap
- Committing, e.g., GitHub, GitLabs, Bitbucket
- Packing: Zip -> S3
- Yarn (not recommended[3])
- Private registry

[3] Avoid yarn for packages for now and fully enjoy its benefits for application development

npm Shrinkwrap

```
npm shrinkwrap
wrote npm-shrinkwrap.json
```

`npm i` will read from the npm-shrinkwrap.json file

Example of npm-shrinkwrap.json

```
...
"prop-types": {
  "version": "15.5.8",
  "from": "prop-types@>=15.5.7 <16.0.0",
  "resolved": "https://registry.npmjs.org/prop-types/-/prop-types-15.5.8.tgz"
},
"react": {
  "version": "15.5.4",
  "from": "react@15.5.4",
  "resolved": "https://registry.npmjs.org/react/-/react-15.5.4.tgz"
},
"setimmediate": {
  "version": "1.0.5",
  "from": "setimmediate@>=1.0.5 <2.0.0",
  "resolved": "https://registry.npmjs.org/setimmediate/-/setimmediate-1.0.5.tgz"
},
"ua-parser-js": {
  "version": "0.7.12",
  "from": "ua-parser-js@>=0.7.9 <0.8.0",
  "resolved": "https://registry.npmjs.org/ua-parser-js/-/ua-parser-js-0.7.12.tgz"
},
"whatwg-fetch": {
  "version": "2.0.3",
  "from": "whatwg-fetch@>=0.10.0",
  "resolved": "https://registry.npmjs.org/whatwg-fetch/-/whatwg-fetch-2.0.3.tgz"
}
...
```

Committing to version control systems

Add

```
git add node_modules
```

Rebuild on CI server:

```
npm rebuild
```

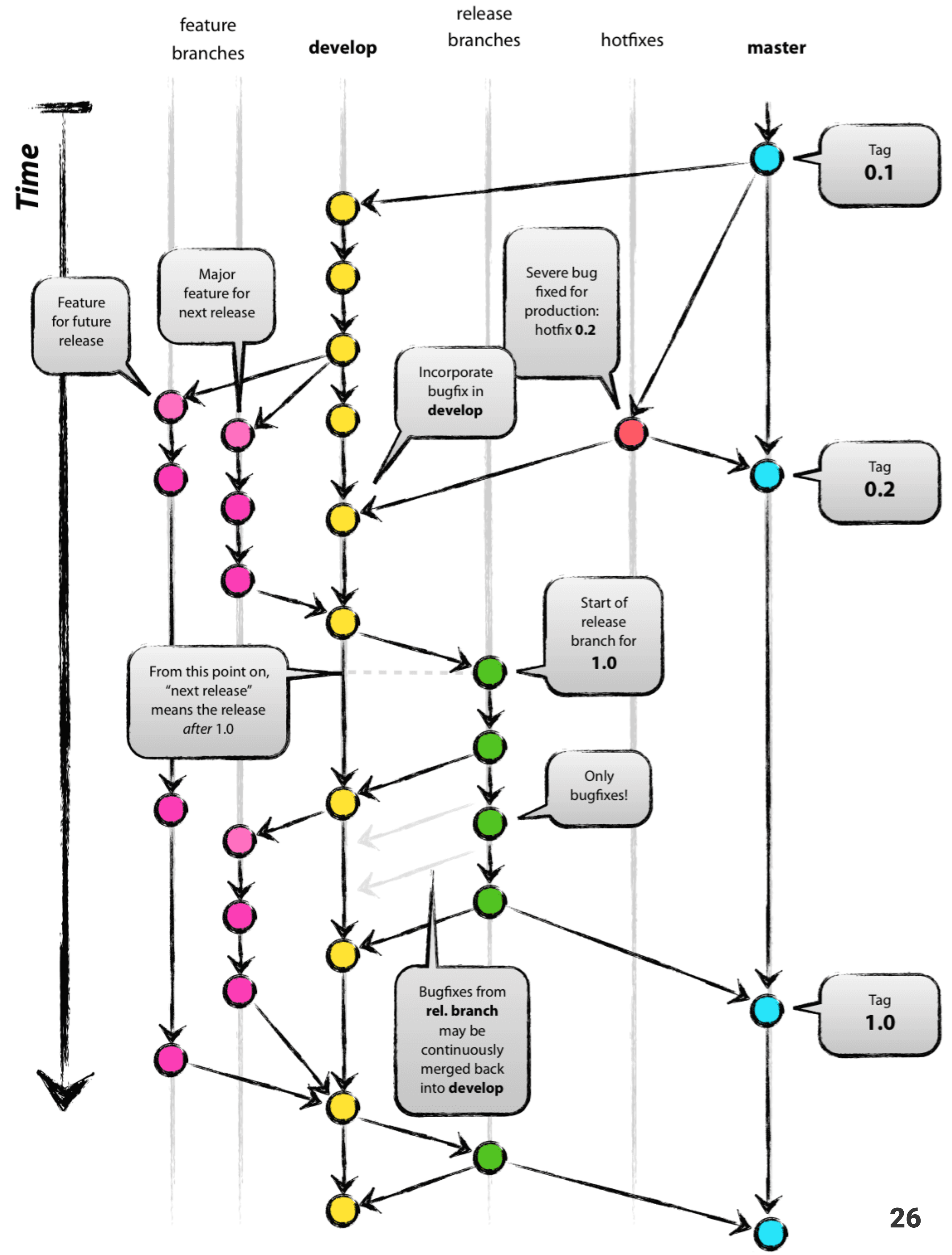
Deployment from Branches

Git flow branches for deployment and releases

- master
- develop
- feature branches
- hotfix branches
- deploy (release)

Source: http://nvie.com/posts/a-successful-git-branching-model and http://nvie.com/files/Git-branching-model.pdf

Related: https://hackernoon.com/a-branching-and-releasing-strategy-that-fits-github-flow-be1b6c48eca2 and https://datasift.github.io/gitflow/IntroducingGitFlow.html


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Stateless Architecture

- Multiple machines in preferably different data centers (AZs)
- Load balancer
- No state in the app layer
- Vertical scaling (pm2, forever, cluster)

Vertical Scaling

- Process managers
- Nginx
- HAProxy
- Varnish Cache
- Redis
- AWS services: ELB, ECS, ElasticCache, Route53, CloudFront, EC2, S3, CloudFormation, AMI

```javascript
var cluster = require('cluster');
var http = require('http');
var numCPUs = require('os').cpus().length;
var express = require('express');
var stats = {};

if (cluster.isMaster) {
  console.log (' Fork %s worker(s) from master', numCPUs)
  for (var i = 0; i < numCPUs; i++) {
    cluster.fork()
  }
  cluster.on('online', function(worker) {
    console.log ('worker is running on %s pid', worker.process.pid)
  })
  cluster.on('exit', function(worker, code, signal) {
    console.log('worker with %s is closed', worker.process.pid)
  })
} else if (cluster.isWorker) {
  var port = 3000
  stats[cluster.worker.process.pid] = 0
  console.log('worker (%s) is now listening to http://localhost:%s',
    cluster.worker.process.pid, port)
  var app = express()
  app.get('*', function(req, res) {
    stats[cluster.worker.process.pid] += 1
    var l ='cluster '
      + cluster.worker.process.pid
      + ' responded \n'
    console.log(l)
    res.status(200).send(l)
  })
  app.listen(port)
}
process.on('SIGINT', function(){
  console.log(stats)
  console.log('Execute "$ killall node" to terminate')
})
```

Process Managers

- pm2
- StrongLoop PM
- forever - 208 open issues

Install pm2

```
npm i -g pm2@2.4.6
```

Start in development:

```
pm2-dev start server.js
```

Start in production:

```
pm2-docker start -i 0 server.js
pm2 start -i 0 server.js
```

Consolidated Uogs

- https://github.com/trentm/node-bunyan
- https://github.com/winstonjs/winston

winston-example.js:

```
const winston = require('winston')

const logger = new (winston.Logger)({
  transports: [
    new (winston.transports.Console)(),
    new (winston.transports.Couchdb)({ 'host': 'localhost', 'db': 'logs' })
    // if you need auth do this:
    // new (winston.transports.Couchdb)({
    //  'user': 'admin', 'pass': 'admin', 'host': 'localhost', 'db': 'logs'
    //})
  ]
})

logger.log('info', 'Hello log files!', { 'message': 'CouchDB logs' })
```

Health-Check Endpoint

GET /status

Response: memory/server load, version SHA, other data.

server.js:

```
app.get('/status', require('routes/status.js'))
```

Node system info

```
const os = require('os')
os.freemem()
os.hostname()
```

Node process info

```
process.memoryUsage()
process.uptime()
process.version
process.env.NODE_ENV
```

Node any info

```
const cp = require('child_process')
cp.exec(posixCommand, (e, result)=>{...})
```

Some POSIX command Node can launch[4]:

```
vmstat -SM -s | grep "used swap" | sed -E "s/[^0-9]*([0-9]{1,8}).*/\1/"
netstat -an | grep :80 | wc -l
netstat -an | grep :3000 | wc -l
```

[4] vmstat: http://linuxcommand.org/man_pages/vmstat8.html

code/status.js:

```
var os = require('os'),
  exec = require('child_process').exec,
  async = require('async'),
  started_at = new Date()

module.exports = function(req, res, next) {
  ...
        res.send({
          status: 'up',
          version: server.get('version'),
          sha: server.et('git sha'),
          started_at: started_at,
          node: {
            version: process.version,
            memoryUsage: Math.round(process.memoryUsage().rss / 1024 / 1024)+"M",
            uptime: process.uptime()
          }, system: {
            loadavg: os.loadavg(),
            freeMemory: Math.round(os.freemem()/1024/1024)+"M"
          },
          228
          env: process.env.NODE_ENV,
          hostname: os.hostname(),
          connections: connections,
          swap: swap
          })
      })
  ...
}
```

Horizontal Scaling

- AWS ELB: Classic and App ELB
- ECS clusters in 2+ AZs
- S3 for static files
- RDS, DynamoDB or ElasticCache

CI/CD

- AWS CodeDeploy, CodePipeline and build (see AWS Intermediate)
- CircleCI and TravisCI - SaaS
- Drone and Jenkins - hosted

Lab 0: Installs

- Slides, labs and code https://github.com/azat-co/node-in-production
- Node and npm (v6 and v4)
- Docker engine
- AWS account and AWS CLI

Detailed instructions and links are in labs/0-installs.md

Time: 15 minutes to download and install, go! 🚀

End of Module 2. Continue with Docker and AWS 👉