

Node.js Auto Scaling in 10 minutes with AWS CloudFormation

2012/6/28 東京Node学園 6限目

@hakobera

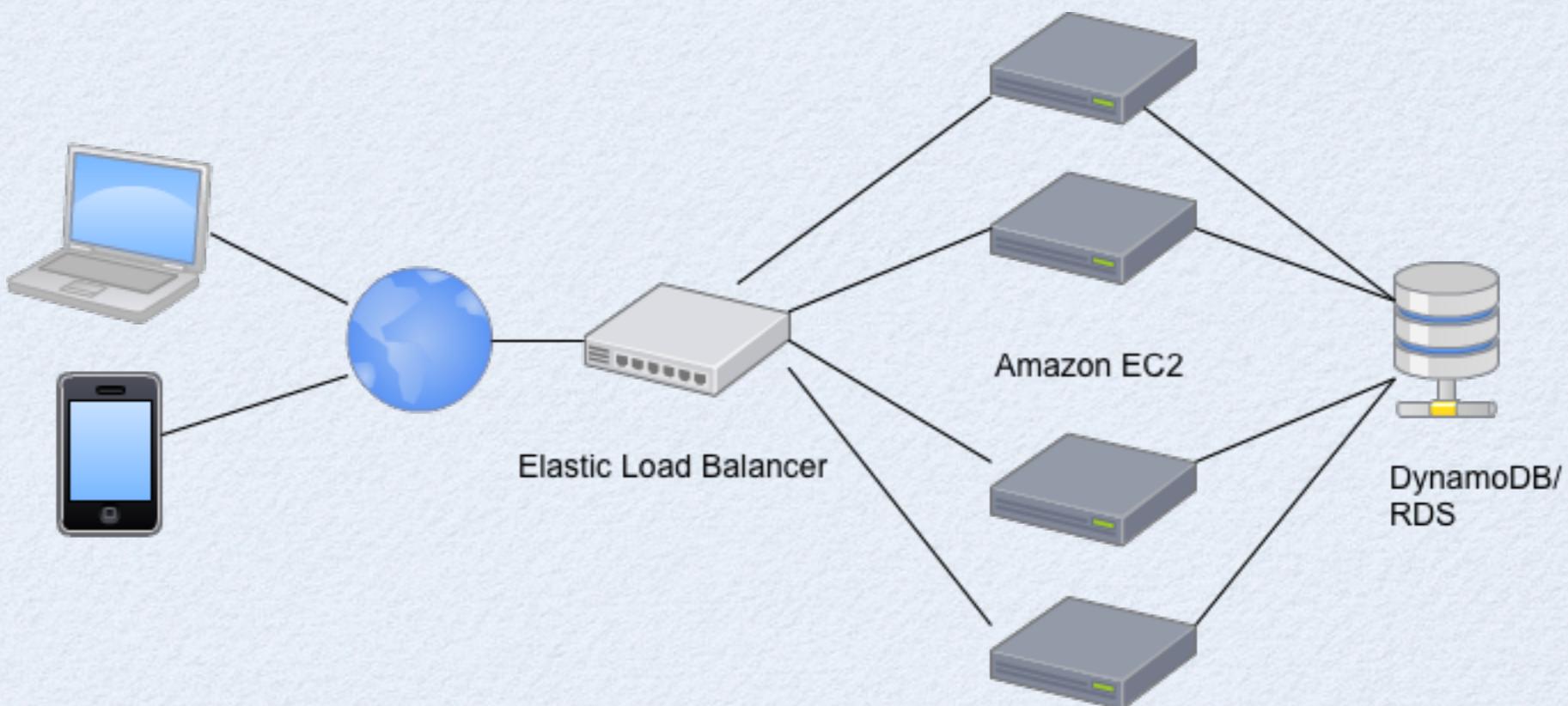
Question

- AWSでNode.jsを何ノードまで動かしたことありますか？

Node.jsで1億リクエスト／日

- 1億リクエスト／日
- => 417万リクエスト／時
- => 7万リクエスト／分
- => 1200リクエスト／秒

AWS なら簡単



AWS なら簡単

- EC2
 - AMI (VMイメージ)の作成
 - Node.js、依存パッケージのインストール
- セキュリティグループ
 - ポート番号や接続制限
- ELB
 - プロトコルとポート番号
- Auto Scale
 - CloudWatch の設定
 - Auto Scale Group ポリシーの設定
- アプリケーションのDeployとUpdate

AWS なら簡単

- EC2
 - AMI (VMイメージ)の作成
 - Node.js、依存パッケージのインストール
- セキュリティグループ

じゃなかったorz

- プロトコルとポート番号
- Auto Scale
 - CloudWatch の設定
 - Auto Scale Group ポリシーの設定
- アプリケーションのDeployとUpdate

そこで AWS CloudFormation の出番

<http://aws.amazon.com/jp/cloudformation/>

- AWS インフラ構成のテンプレート
- 設定の一部をパラメータ化できる
- JSON 形式 (Node.js にはお馴染み)

DynamoDBとCloudFront の先週からサポート開始
詳細は[こちら](#)を参照

自分だけの PaaS が
簡単に作れる

とりあえずやってみる

Template:

<https://cf-templates-for-nodejs.s3.amazonaws.com/cf-node-autoscale-tmpl.json>

App:

https://github.com/hakobera/tng6_sample/zipball/master

GitHub:

https://github.com/hakobera/tng6_sample

Node.js と CloudFormation の勘所

Node.js の
インストール

AutoScale

アプリの
Deploy と Update

Elastic Load
Balancer と
WebSocket

Node.js のインストール

Node.jsのインストール

バイナリパッケージを使おう

∴ micro インスタンスで Node.js のビルドが遅い

Installing Node.js via package manager

<https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager>

```
$ sudo yum localinstall --nogpgcheck http://nodejs.tchol.org/repos/amzn1/nodejs-stable-release.noarch.rpm
$ sudo yum install nodejs-compat-symlinks npm
```

残念ながら v0.8 はまだない

Node.jsのインストール

Cloud Init でパッケージ指定

```
"AppServerLaunchConfig": {  
    "Type" : "AWS::AutoScaling::LaunchConfiguration",  
    "Metadata": {  
        "AWS::CloudFormation::Init": {  
            "config": {  
                "packages": {  
                    "rpm" : {  
                        "nodejs" : "http://nodejs.tchol.org/repo/cfg/amzn1/nodejs-stable-release.noarch.rpm"  
                    },  
                    "yum": {  
                        "gcc-c++": [],  
                        "make": [],  
                        "git" : [],  
                        "nodejs-compat-symlinks": [],  
                        "npm": []  
                    }  
                }  
            },  
  
            "sources" : {  
                "/var/opt/app" : { "Ref": "AppURL" }  
            }  
        }, ...  
    }  
}
```

Node.jsのインストール

Cloud Init の起動設定

```
"Properties": {  
    (省略)  
}  
  
"UserData": {  
    "Fn::Base64": {  
        "Fn::Base64": {  
            "Fn::Join": [ "", [  
                "#!/bin/bash -v\n",  
                "yum update -y aws-cfn-bootstrap\n",  
  
                "# Helper function\n",  
                "function error_exit\n",  
                "{\n",  
                " /opt/aws/bin/cfn-signal -e 1 -r \"$1\" \"", { "Ref": "WaitHandle" }, "\n",  
                " exit 1\n",  
                "}\n",  
  
                "# Install packages\n",  
                "/opt/aws/bin/cfn-init -s ", { "Ref": "AWS::StackName" }, " -r AppServerLaunchConfig ",  
                " --access-key ", { "Ref": "HostKeys" },  
                " --secret-key ", { "Fn::GetAtt": ["HostKeys", "SecretAccessKey"] },  
                " --region ", { "Ref": "AWS::Region" }, " || error_exit 'Failed to run cfn-init'\n",  
            ]]  
        }  
    }  
}
```

Auto Scale

Cloud Watch

<http://aws.amazon.com/jp/cloudwatch/>

<http://aws.amazon.com/jp/autoscaling/>

- Auto Scale の条件（拡張・縮小のトリガー）は、Cloud Watch のアラームで実現している
- アプリの特性の応じて、チェックする対象を変更
 - CPUUtilization, DiskReadBytes, DiskReadOps, DiskWriteBytes, DiskWriteOps, NetworkIn, NetworkOut
 - Node.js だと、CPUUtilization か NetworkOut のどちらかで大体OK

設定例: CPUUtilization

アプリの Deploy と Update

アプリのDeploy

Create Stack

Cancel X

SELECT TEMPLATE SPECIFY PARAMETERS REVIEW

Template Description: Node.js AutoScale Template. This template creates an Amazon EC2 instance. You will be billed for the AWS resources used by your application. Review the terms and conditions before you create this stack.

Specify Parameters

Below are the parameters associated with your CloudFormation template. You can proceed with the default parameters or make customizations.

RootDeviceType ebs
EC2 root device type ('ebs' or 'instanceStore')

AppServerPort 3000
TCP/IP port of the application server.

AppURL zipball または tarball の URL
Application Archive URL

AppServerInstanceType t1.micro
Application server instance type

KeyName
Name of an existing EC2 KeyPair to enable SSH access.

I acknowledge that this template may create IAM users.

Back Continue ▶

```
"AppServerLaunchConfig": {  
    "Type" : "AWS::AutoScaling::LaunchConfiguration",  
    "Metadata": {  
        "AWS::CloudFormation::Init": {  
            "sources" : {  
                "/var/opt/app" : { "Ref": "AppURL" }  
            }  
        },  
        "Properties": {  
            "UserData": {  
                "Fn::Base64": {  
                    "Fn::Base64": {  
                        "Fn::Join": [ "", [  
                            "# Start application\n",  
                            "cd /var/opt/app\n",  
                            "npm config set PORT ", { "Ref": "AppServerPort" }, "\n",  
                            "npm install --production\n",  
                            "NODE_ENV=production npm start > app.log 2>&1 &\n",  
                        ]]  
                }  
            }  
        }  
    }  
},  
指定了したフォルダにダウンロード、展開してくれる
```

環境変数の取得

```
/**  
 * Returns a value related by given key.  
 * This method search following priority.  
 *  
 * 1. NPM package config  
 * 2. NPM config  
 * 3. process.env  
 *  
 * @param {String} key Environment key  
 * @param {String} [defaultValue] Default value if key not found.  
 * @return {String}  
 */  
  
module.exports = function (key, defaultValue) {  
  var NPM_PACKAGE_PREFIX = 'npm_package_config_';  
  var NPM_PREFIX = 'npm_config_';  
  
  var value = process.env[NPM_PACKAGE_PREFIX + key];  
  if (!value) {  
    value = process.env[NPM_PREFIX + key];  
    if (!value) {  
      value = process.env[key];  
    }  
  }  
  value = (!value && defaultValue) ? defaultValue : value;  
  return value;  
};
```

NPM package config
NPM config (global)
process.env
の順で環境変数を解決する

環境変数の取得

```
var http = require('http'),  
    env = require('./env');  
  
var server = http.createServer(function (req, res) {  
    res.writeHead(200, { 'Content-Type': 'text/plain' });  
    res.end('hello');  
});  
server.listen(env('PORT'), function () {  
    console.log('Server is running on port %d', server.address().port);  
});
```



開発時は package.json を参照

```
{  
  "name": "harite",  
  "version": "0.1.0",  
  "scripts": {  
    "start": "node app.js",  
    "test": "make test"  
  },  
  "config": {  
    "PORT": 5100,  
    "REDIS_URL": "redis://localhost"  
}
```

AWS では CloudInit で指定

```
"npm config set harite:HOST_NAME `curl http://169.254.169.254/latest/meta-data/public-hostname`", "\n",  
"npm config set harite:PORT ", { "Ref": "HariteServerPort" }, "\n",  
"npm config set harite:REDIS_URL ", { "Fn::Join" : [ "", [ "redis://", { "Fn::GetAtt" : [ "RedisServer", "PrivateIp" ]}], "\n",  
"npm install --production\n",  
"NODE_ENV=production npm start > harite.log 2>&1 &\n",
```

参考資料：<http://d.hatena.ne.jp/sugyan/20110909/1315575343>

<http://d.hatena.ne.jp/Jxck/20120410/1334071898>

Github の場合

Branch 単位で zipball が取得できる

https://github.com/hakobera/tng6_sample/zipball/master

リポジトリ

ブランチ名

アプリのUpdate

Update Stack =>



- LoadBalancer から現行バージョンのサーバを徐々に切り離す
- AutoScale の最低サーバ数以下になると、新規サーバが起動
- 新規サーバは新しいアプリがデプロイされた状態で起動する
- (今回のテンプレートはなっていないが) タグにアプリケーションのバージョンを仕込んでおいて、API 経由で判定 => 停止などやれば自動化できるかも

WebSocket

ELBと相性が悪い

- プロトコルでHTTPを選択すると、HTTP1.1 のUpdateリクエストが通らないので、WebSocket がつながらない
- プロトコルをTCPを選択すれば接続はできるが、接続先サーバを固定できない (Socket.IO で苦労する)
- 60秒で必ずタイムアウトする (設定の変更はできない)

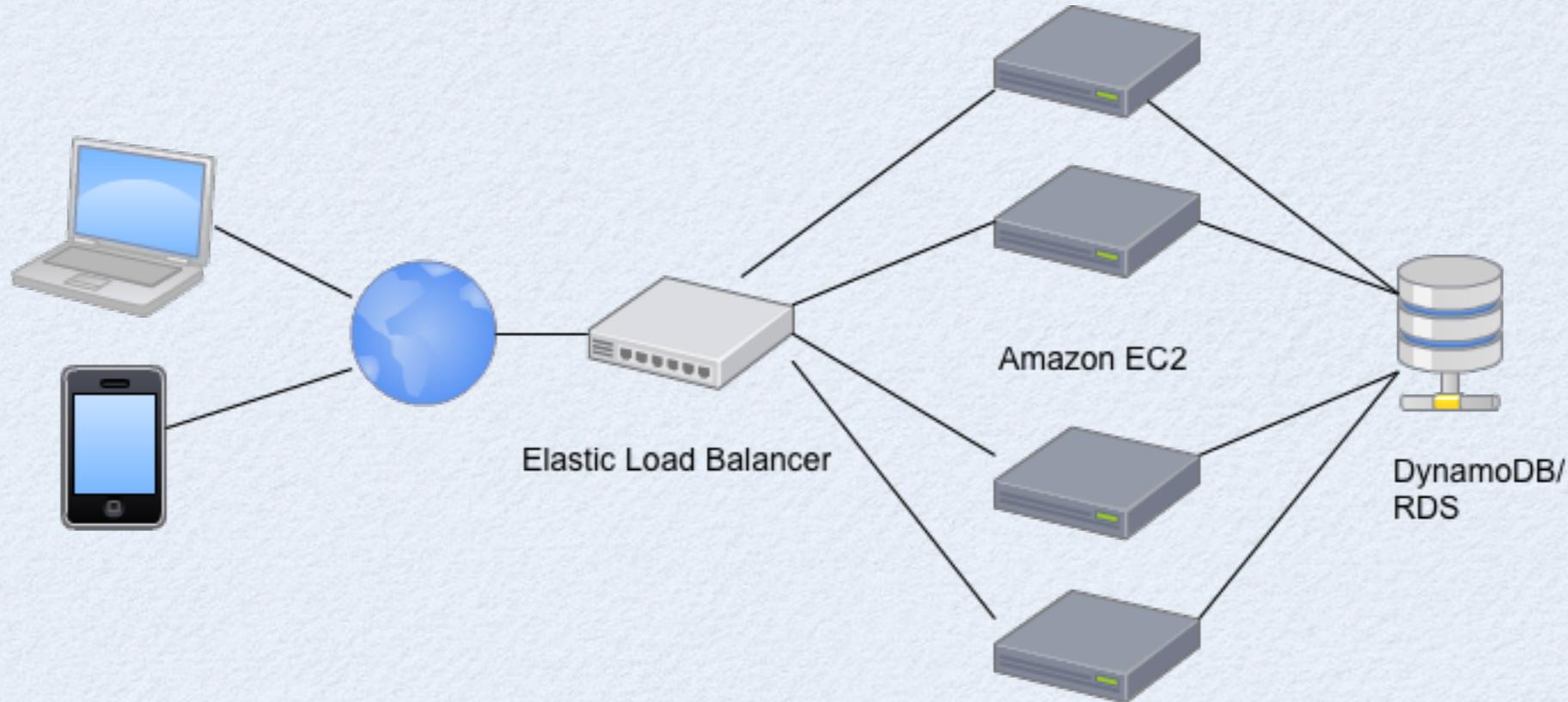
参考： Socket.IO or WebSocket を AmazonELB でバランスする検証

<http://d.hatena.ne.jp/Jxck/20120228/1330444857>

AutoScaleと相性が悪い

- AutoScale は WebSocket がつながっていても縮退してしまう
- クライアント側での再接続処理が必要
- ただし、普通に再接続すると、EC2インスタンス自体が落ちているので、つながらない

回避策



- WebSocket の URL を返す API に ELB 経由でアクセス (HTTPヘッダで返すのでも良い)
 - EC2 の Public DNS名は `curl http://169.254.169.254/latest/meta-data/public-hostname` で取得可能
- クライアントは各EC2インスタンスと直接接続する
- 再接続時には、また ELB 経由で新しい WebSocket サーバのURLを取得
 - これを実現するクライアントライブラリ (WebSocketラッパー) を作成する

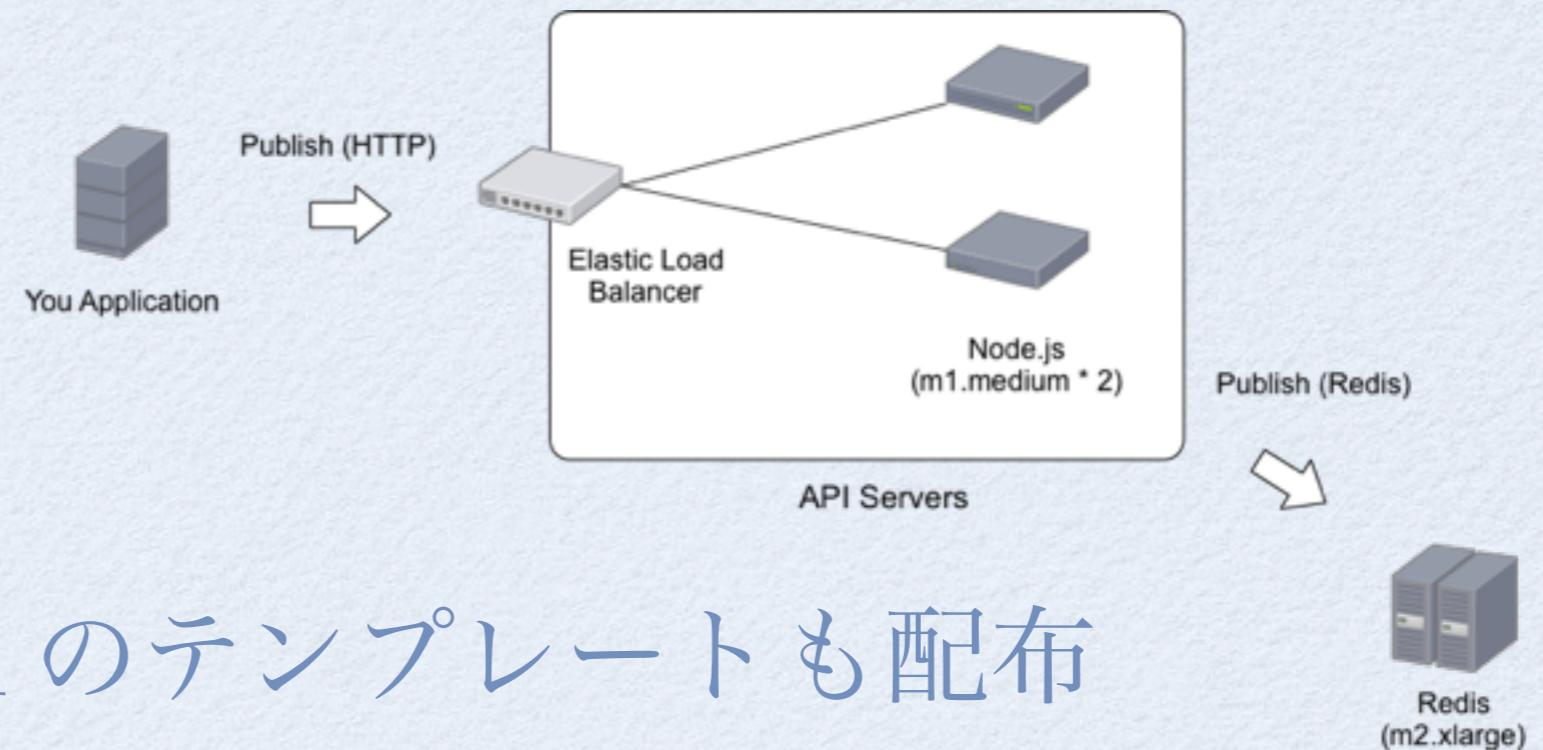
閑話休題

Pusher クローン作ってます

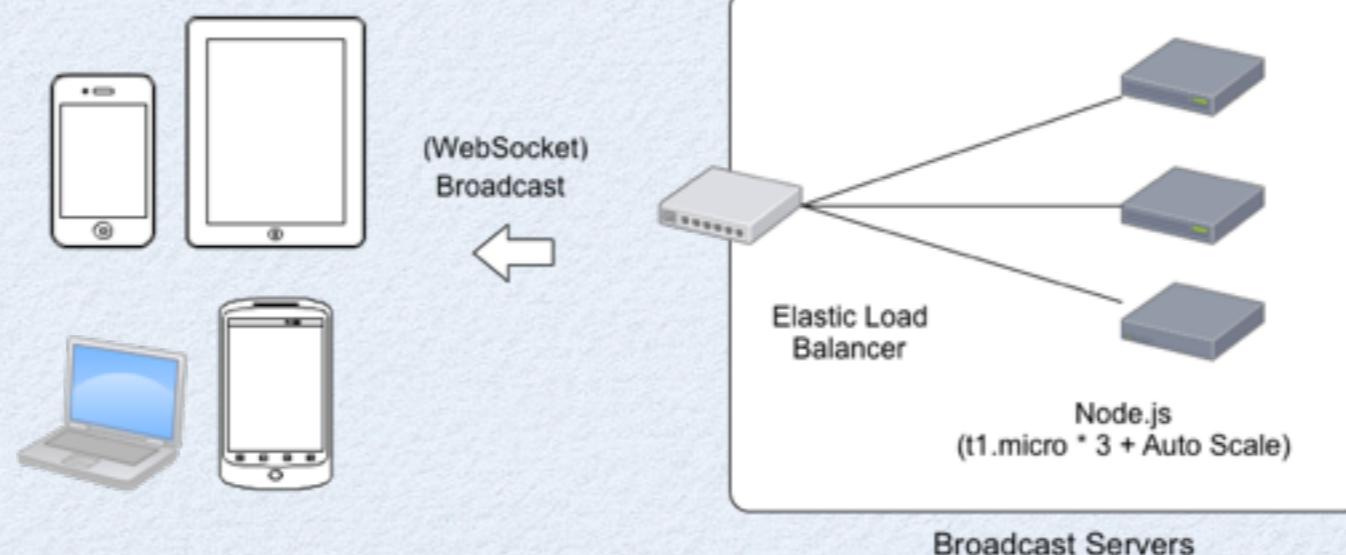
<https://github.com/hakobera/tuppari>

来週公開予定

<https://github.com/hakobera/tuppari-servers>



CloudFormation のテンプレートも配布



Any Question ?