Docker, AWS et intégration continue

Jauffrey Flach

Version 1.0

Pré-requis

- un compte chez Amazon AWS
- Le client Elastic Beanstalk
  - pour les mac users:
    - `brew install aws-elasticbeanstalk`
  - pour les autres:
    - https://docs.aws.amazon.com/fr_fr/elasticbeanstalk/latest/dg/eb-cli3-install.html
  - Docker
  - NodeJS
  - Git
  - Un editeur de texte

Docker

Nous allons tout d'abord construire un simple projet avec npm (un hello world), le mettre dans un conteneur Docker, et l'éxécuter. Une fois ce projet "fianlisé", nous allons le faire tourner sur Elastic Beanstalk.

Construction du projet

- Créez un nouveau dossier **aws**
- Dans ce dossier, initialisez l'app avec `npm init -f.`
- Créez un fichier app.js

```
// Get dependencies
var express     = require('express');
var morgan      = require('morgan');

var app = express();
var server = require('http').Server(app);
port = process.env.PORT || 8080;

// Catch all other routes and return the index file
app.get('/', (req, res) => {
  res.send("hello world!!");
});

// use morgan to log requests to the console
app.use(morgan('dev'));

server.listen(port);
console.log('App running at http://localhost:' + port);
```

- Installez les dépendances avec `npm install --save express morgan`
- Modifiez le package.json pour y ajouter un script de démarrage:

```
{
  "name": "aws",
  "version": "1.0.0",
  "description": "A simple project",
  "main": "index.js",
  "scripts": {
    "start": "node app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.2",
    "morgan": "^1.9.0"
  }
}
```

- Testez votre projet:
  - `npm start`
  - `curl localhost:8080`

Docker FTW

- Ajoutez le Dockerfile

```
FROM node:alpine

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json .
# For npm@5 or later, copy package-lock.json as well
# COPY package.json package-lock.json .

RUN npm install

# Bundle app source
COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]
```

- Ajoutez le fichier .dockerignore

```
node_modules
npm-debug.log
```

- Nous pouvons maintenant construire notre image avec Docker avant de la déployer avec EB.
  - `docker build -t aws .`
- Testons notre image dans un conteneur:
  - `docker run -d -p 8080:8080 --name aws aws`
  - `curl $(docker-machine ip):8080`

AWS

- Création des crédentials
  - Allez sur https://console.aws.amazon.com/iam
    - Créez un nouveau groupe ElasticBeanStalkTest
    - Ajoutez-y toutes les permissions S3 et Beanstalk
    - Créez un nouvel utilisateur et mettez le dans le groupe.
    - A la fin du processus de création, AWS vous propose de sauvegarder les credentials en csv. Faites-le.
- Initialisation du projet. Dans votre dossier 'aws':
  - `eb init`
- Renseignez les informations relatives à vos credentials, et utilisez les valeurs par défaut pour le reste.
  - `eb create`
    - Répondez aux questions en utilisant les valeurs par défaut, sauf pour Docker, utilisez la 17.06-ce

Voilà, après le déploiement votre Hello World est disponible. Pour accéder à la console de gestion: https://eu-west-3.console.aws.amazon.com/elasticbeanstalk/home?region=eu-west-3#/applications

Pour déployer des changements directement sur aws: `eb deploy` dans le répertoire de travail

Pour éteindre votre serveur: `eb terminate`

Intégration continue avec Github et Travis CI

Partie 1: Travis CI et Github

Nous avons déployé dans la première partie notre application en utilisant la ligne de commande. Essayons maintenant de déployer cette application avec Travis CI et Github

- Pushez votre code sur un repo Github
- Ajoutez l'intégration TravisCI sur votre compte Github
  - https://docs.travis-ci.com/user/getting-started/
- Ajoutez un fichier .travis.yml dans votre repo
  - Pour notre application, il devrait ressembler à ceci:

```
language: node_js

node_js:
  - "8"

sudo: required

services:
  - docker

after_deploy:
  - echo "done deploying"
```

- Si votre build passe, vous allez pouvoir intégrer elasticbeanstalk

Partie 2: elasticbeanstalk

- Récupérez avec la console ou la ligne de commande la région et le bucket name que vous utilisez
- Dans l'interface de Travis CI, ajoutez des variables d'environnement sécurisées contenant vos credentials AWS
  - SECRETACCESSKEY
  - ACCESSKEYID
- Ajoutez à votre fichier .travis.yml les éléments relatifs à EB:

```
deploy:
  provider: elasticbeanstalk
  access_key_id: $ACCESSKEYID
  secret_access_key:
    secure: "$SECRETACCESSKEY"
  region: "<votre-region>"
  app: "aws"
  env: "aws-dev"
  bucket_name: "<bucket-name>"
  on:
    branch: master
```

Une fois que vous allez faire une modification sur la branche master, Travis va lancer un build et déployer sur AWS.

Voilà, vous avez automatisé le déploiement de vos applications Docker. Plus rien ne vous arrête (sauf peut-être votre limite CB si vous dépassez les quotas AWS :))

Webographie

Ce tutorial a été repris et adapté depuis les liens suivants:

- https://medium.com/@sommershurbaji/continuous-delivery-with-aws-elastic-beanstalk-and-travis-ci-2dd54754965f
- https://medium.com/@sommershurbaji/deploying-a-docker-container-to-aws-with-elastic-beanstalk-28adfd6e7e95