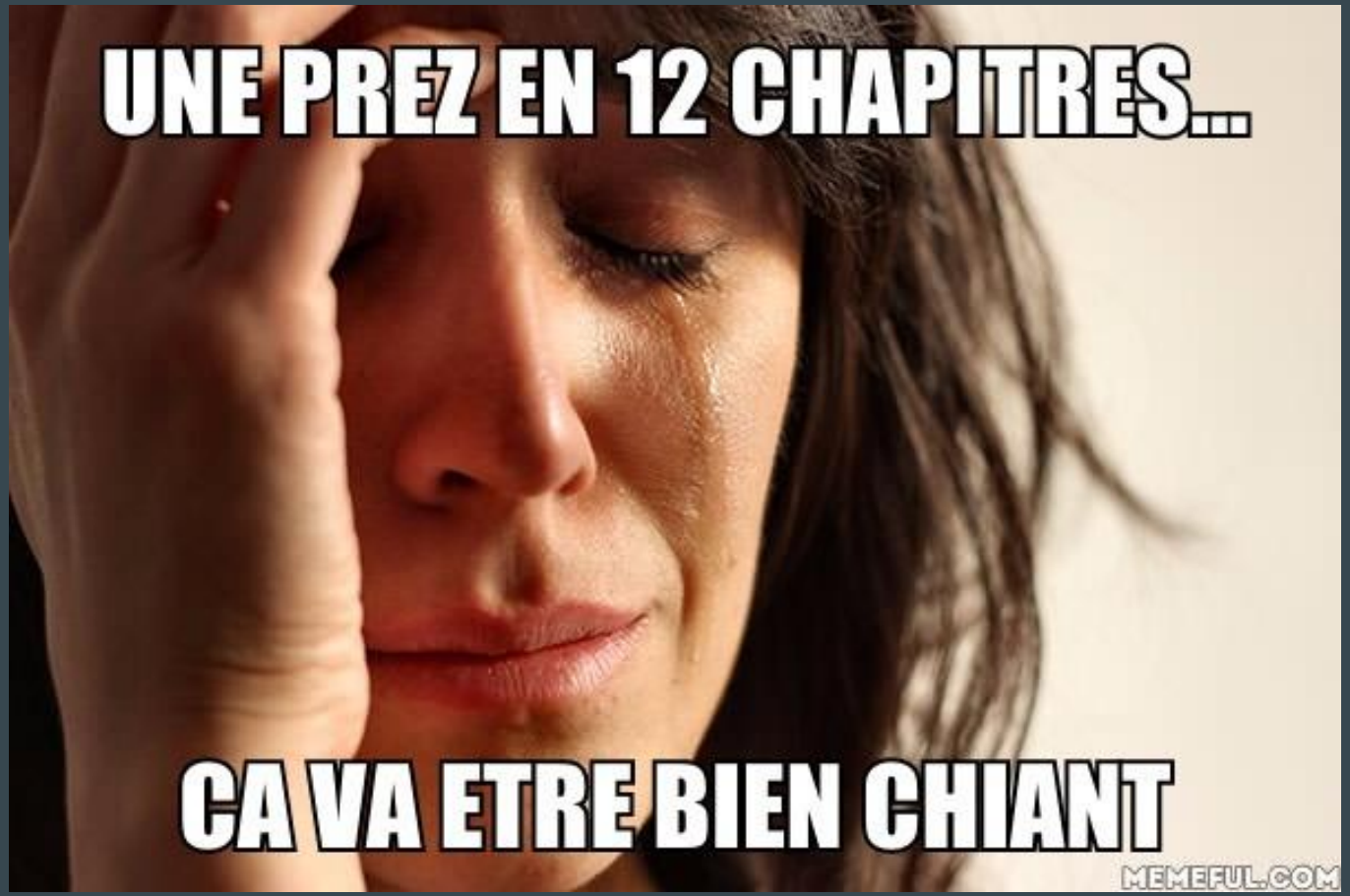
12 factor application

...

... avec un peu de Node.js et un peu de Docker

En quelques mots

- De nombreuses applications en SaaS
- Retour d’expérience des déploiements sur Heroku
- Manifeste de création d’applications web en 12 facteurs
  - Portabilité
  - Scalabilité horizontale
  - Cloud-ready


UNE PREZ EN 12 CHAPITRES...
CA VA ETRE BIEN CHIANT
MEMEFUL.COM

Un fil conducteur

- Une application à faire évoluer avec la méthodologie 12 factor
- API HTTP Rest Node.js / Sails.js

```
$ sails new messageApp
$ cd messageApp
$ sails generate api message
$ npm start
```

Creation

```
$ curl -XPOST http://localhost:1337/message?text=hello
{
  "text": "hello",
  "createdAt": "2016-06-15T19:22:41.211Z",
  "updatedAt": "2016-06-15T19:22:41.211Z",
  "id": 1
}

$ curl http://localhost:1337/message
[
  {
    "text": "hello",
    "createdAt": "2016-06-15T19:22:41.211Z",
    "updatedAt": "2016-06-15T19:22:41.211Z",
    "id": 1
  }
]
```

Test


1 - Codebase
2 - Dépendances
3 - Configuration
4 - Services externes
5 - Build / Release / Run
6 - Processus
12 Factor
7 - Association de ports
8 - Concurrence
9 - Jetable
10 - Parité dev / prod
11 - Logs
12 - Processus d’admin

1. Codebase

- Une application ⇔ un repo pour versionner le code

- Plusieurs déploiement de l'application

```
$ echo "# messageApp" >> README.md
$ git init
$ git add .
$ git commit -m 'First commit'
$ git remote add origin git@github.com:lucj/messageApp.git
$ git push origin master
```

Création de messageApp sur Github

Create et push 1er commit

2. Dépendences

- Les dépendances doivent être déclarées et isolées
- Déclaration dans le fichier package.json
  - *npm install sails-mongo --save*
- Isolation dans le répertoire node_modules

```
"dependencies": {
    "ejs": "2.3.4",
    "grunt": "0.4.5",
    "grunt-contrib-clean": "0.6.0",
    "grunt-contrib-coffee": "0.13.0",
    "grunt-contrib-concat": "0.5.1",
    "grunt-contrib-copy": "0.5.0",
    "grunt-contrib-cssmin": "0.9.0",
    "grunt-contrib-jst": "0.6.0",
    "grunt-contrib-less": "1.1.0",
    "grunt-contrib-uglify": "0.7.0",
    "grunt-contrib-watch": "0.5.3",
    "grunt-sails-linker": "~0.10.1",
    "grunt-sync": "0.2.4",
    "include-all": "~0.1.6",
    "rc": "1.0.1",
    "sails": "~0.12.3",
    "sails-disk": "~0.10.9",
    "sails-mongo": "^0.12.0"
  },
```

3. Configuration

- Configuration dans les variables d'environnement

- Découplage du code et de l'environnement d'execution

config/connections.js:

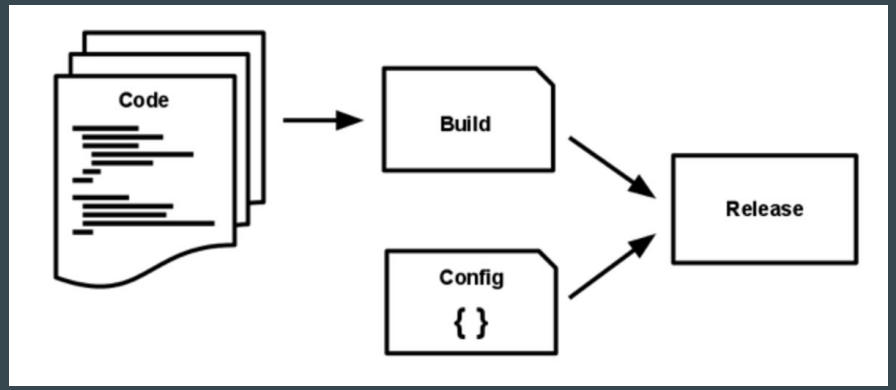
```
module.exports.connections = {
  mongo: {
     adapter: 'sails-mongo',
     url: process.env.MONGO_URL'
  }
};
```

4. Services externes

- Ensemble des services réseau utilisés par l'application
  - Database, Message Queue, Logs, API, ...
- Pas de distinction entre les services locaux et les services tiers
- Ressources que l'on doit pouvoir remplacer
- Assure le découplage (loosely coupled)

5. Build / Release / Run

- Séparation des différentes étapes



- Une release est déployée sur l'environnement d'execution et est immutable

Build

```
FROM mhart/alpine-node:4.4.5

# Copy list of dependencies
COPY package.json /tmp/package.json

# Install dependencies
RUN cd /tmp && npm install

# Copy dependencies libraries
RUN mkdir /app && cp -a /tmp/node_modules /app/

# Change working directory
WORKDIR /app

# Copy source code
COPY . /app

# Expose API port to the outside
ENV PORT 80
EXPOSE 80

# Launch application
CMD ["npm","start"]
```

Dockerfile

Build à partir d'un Dockerfile

$ docker build -t message-app .

$ docker images

REPOSITORY	*TAG*	*IMAGE ID*	*CREATED*	*SIZE*
message-app	*latest*	*f35464cf4b0b*	*2 seconds ago*	*769 MB*

Release

```
version: '2'
services:
  mongo:
    image: mongo:3.2
    volumes:
      - mongo-data:/data/db
    expose:
      - "27017"
  app:
    image: message-app
    ports:
      - "8000:80"
    links:
      - mongo
    depends_on:
      - mongo
    environment:
      - MONGO_URL=mongodb://mongo/messageApp
volumes:
  mongo-data:
```

docker-compose.yml

- Release définie à partir de docker-compose.yml
- Injection de la configuration via MONGO_URL
- Execution
  - via un orchestrateur (Docker Cloud, Rancher, …)
  - directement avec Compose CLI

6. Processus

- Une application est constituée de processus stateless
- Pas de stockage en local mais dans une ressource stateful
- Assure la scalabilité et la tolérance aux pannes

config/sessions.js

```
module.exports.session = {
  ...
  adapter: 'redis',
  host: process.env.REDIS_HOST || 'localhost',
  ...
};
```


6 FACTOR APP...
DONE !!!
MEMEFUL.COM

7. Associations de ports

- Exposition d'une application via un port
- Utilisation de dépendances spécialisées (serveur http, ...)
- Requêtes routées depuis l'environnement d'execution

```
...
app:
  image: message-app
  ports:
    - "8000:80"
  links:
    - mongo
  depends_on:
    - mongo
  environment:
    - MONGO_URL=mongodb://mongo/messageApp
...
```

8. Concurrence

- Application constituée de différents types de processus:
  - ex: web, worker, cron
- Scalabilité horizontale à l'aide du modèle de processus
- Chaque processus peut avoir son propre multiplexage interne
  - thread, event loop

9. Jetable

- Pet vs Cattle
- Démarrage rapide
- Arrêt gracieux
- Robustesse au crash

```
# Kafka message broker
zookeeper:
  image: wurstmeister/zookeeper
  ports:
    - "2181:2181"
kafka:
  image: wurstmeister/kafka
  ports:
    - "9092:9092"
  links:
    - zookeeper:zk
  environment:
    KAFKA_ADVERTISED_HOST_NAME: 192.168.99.100
    KAFKA_CREATE_TOPICS: "DATA:1:1"
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

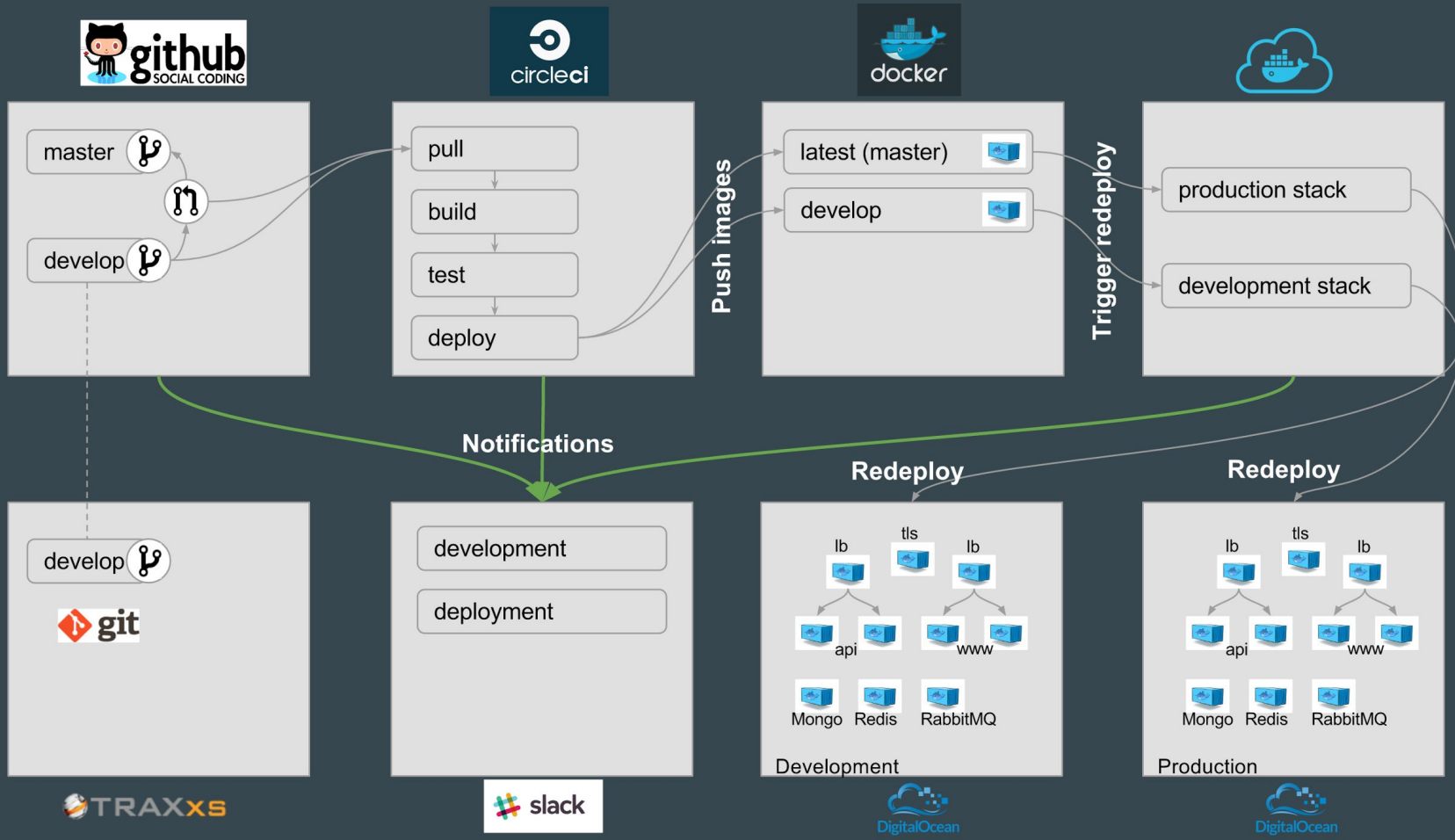
Kafka pour processus de type worker

10. Parité dev / prod

- Alignement des environnements
- Docker très bon pour minimiser le gap
- Services externes disponibles sur Docker Hub
- Orienté déploiement continu
  - release plusieurs fois par jour
- Release peut être déployée sur tout Docker Host

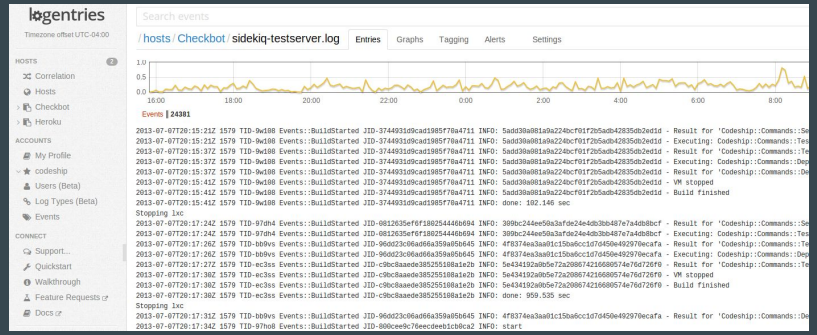
```
version: '2'
services:
  mongo:
    image: mongo:3.2
    volumes:
      - mongo-data:/data/db
    expose:
      - "27017"
  app:
    image: message-app
    ports:
      - "8000:80"
    links:
      - mongo
    depends_on:
      - mongo
    environment:
      - MONGO_URL=mongodb:
//mongo/messageApp
volumes:
  mongo-data:
```

docker-compose.yml


github
SOCIAL CODING
circleci
docker
master
develop
pull
build
test
deploy
Push images
latest (master)
develop
Trigger redeploy
production stack
development stack
Notifications
Redeploy
Redeploy
develop
git
development
deployment
lb
tls
lb
api
www
Mongo
Redis
RabbitMQ
Development
Production
TRAXXS
slack
DigitalOcean
DigitalOcean

11. Logs

- Un flux d'évènements temporel
- Dirigé vers stdout / stderr (pas de fichier de logs en local)
- Gestion centralisée pour faciliter leur exploitation




```
log:
  command: '-t a80277ea-4233-7785203ae328'
  image: 'logentries/docker-logentries'
  restart: always
  tags:
    - development
  volumes:
    - '/var/run/docker.sock:/var/run/docker.sock'
```

Utilisation de de logentries

12. Processus d’administration

- One-off processus
- Executé sur la même release que l’application
- Définition d’un service en “standby” dans le fichier compose
  - accès à l’ensemble des services de l’application

En résumé

- Méthodologie à tester
  - bon sens
  - points plus complexes
- Prépare une application à un déploiement “cloud”
- Accent mis sur
  - scalabilité
  - portabilité
  - dépendances


12 FACTOR C'EST COOL !!!
(ET EN PLUS C'EST FINI)
MEMEFUL.COM

Questions ?

...