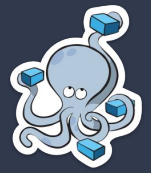
Catherin Cruz
@UserTwoGG

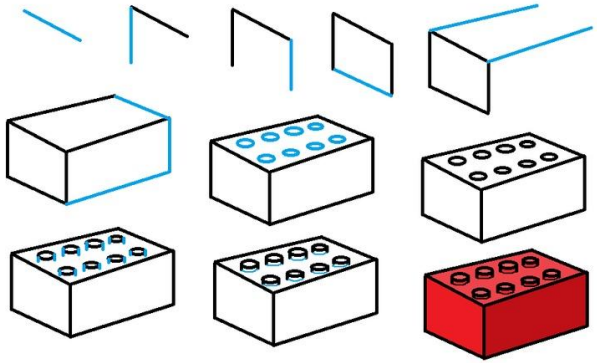
Medellín
DevOps

Repaso Docker



Docker Compose

Un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software. Proporciona una capa de abstracción y virtualización a nivel de SO en Linux.

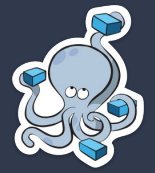


Razones para usar Docker:

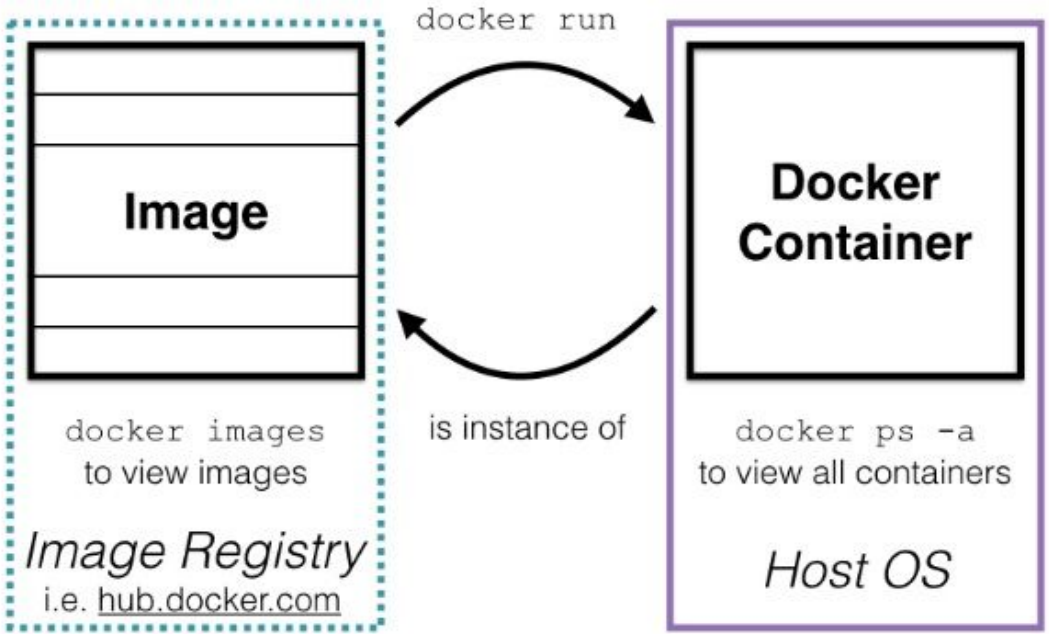
- ❏ Agnóstico al contenido
- ❏ Consistencia ambientes de desarrollo
- ❏ Ambiente de desarrollo exactamente igual al de producción
- ❏ No es necesario instalar localmente soporte para lenguajes

Ejecuta los **contenedores** independientemente dentro de una sola instancia de Linux.

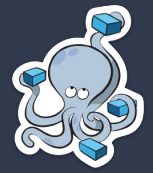
Componentes Básicos



Docker Compose



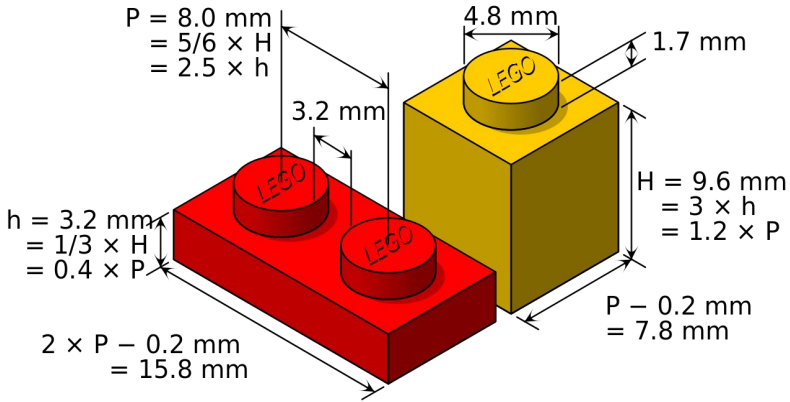
Múltiples Contenedores



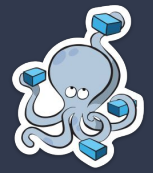
Docker Compose

Cuando manejas contenedores/servicios relacionados:

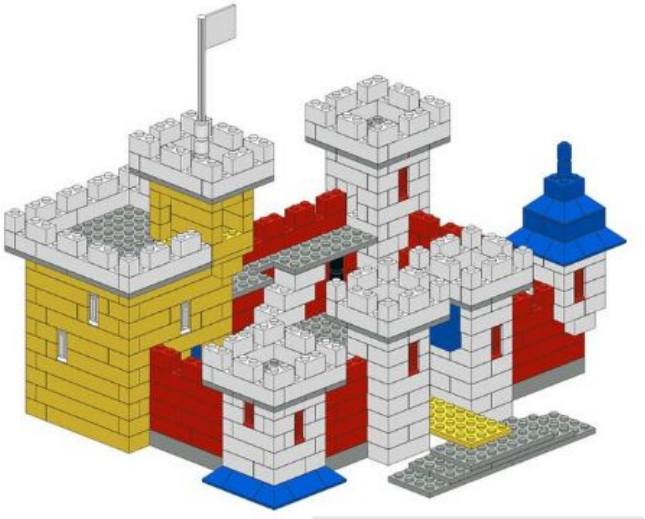
- Debes controlar los estados de cada servicio
- Definir dependencias, redes, accesos
- Validar construcciones y actualizaciones
- Reiniciar solo ciertos servicios
- Mapear puertos y volúmenes por cada contenedor
- Compartir variables de configuración
- Escalar servicios y validar su funcionamiento



Docker Compose



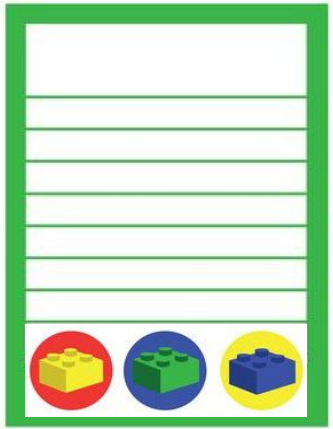
Funcionalidades para la orquestación y manejo de **múltiples contenedores/servicios**.



- Manejo de un único archivo de configuración para todos los contenedores/servicios requeridos.
- Resolución mediante DNS de los servicios en ejecución directamente con Docker.
- Definición de dependencias, orden de despliegue y escalamiento de servicios.

Archivo YAML

Docker Compose



Puede nombrar de ambas formas

`.yml` `.yaml`

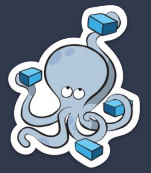
Ubicación estándar del archivo .yaml

`./docker-compose.yml`

```yaml
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: "redis:alpine"
```

Catherin Cruz

YAML: Port



Docker Compose

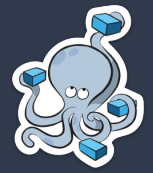
Comando para exponer los puertos. Primero el puerto que va a ser usado por la maquina y despues el puerto del contenedor **HOST:CONTAINER**

```yaml
ports:
 - "3000"
 - "3000-3005"
 - "8000:8000"
 - "9090-9091:8080-8081"
 - "49100:22"
 - "127.0.0.1:8001:8001"
```

```yaml
ports:
  - target: 80
    published: 8080
    protocol: tcp
    mode: host
```

Se recomienda usar siempre formato texto “ ”

YAML: Volumes



Docker Compose

Formatos para definir el volumen:

Definiendo solo el path
- /var/lib/mysql

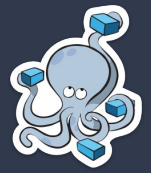
Especificar el path completo
- /opt/data:/var/lib/mysql

Path definido por usuario
- ~/configs:/etc/configs/:ro

Definición por nombre
- datavolume:/var/lib/mysql

```
version: "3.2"
services:
  web:
    image: nginx:alpine
    volumes:
      - type: volume
        source: mydata
        target: /data
  db:
    image: postgres:latest
    volumes:
      - "dbdata:/var/lib/postgresql/data"

volumes:
  mydata:
  dbdata:
```

YAML: Build



Docker Compose

Opciones y comandos aplicados al momento de construir el contendor

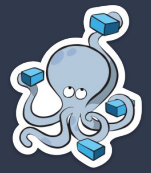
```
version: '2'
services:
  webapp:
    build: ./dir
    image: webapp:tag
```

Este ejemplo construirá una imagen llamada **webapp** usando el Dockerfile ubicado en la carpeta ./dir

```
version: '2'
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
```

Versión con más argumentos

YAML: Args



Docker Compose

Parámetros usados únicamente para el momento de construcción. Importante definirlos en el Dockerfile

```
ARG buildno
ARG password

RUN echo "Build number: $buildno"
RUN script-requiring-password.sh "$password"
```

Valores booleanos deben enviarse en formato texto “false”

```
( true , false , yes , no , on , off )
```

Parámetros tipo: Mapa

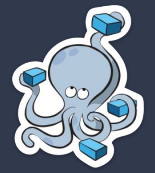
```
build:
  context: .
  args:
    buildno: 1
    password: secret
```

Parámetros tipo: Lista

```
build:
  context: .
  args:
    - buildno=1
    - password=secret
```

Catherin Cruz

YAML: Depens_on



Docker Compose

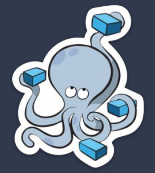
Define la dependencia entre servicios, realiza dos tareas:

`docker-compose up` iniciara los servicios en el orden de dependencia. En el ejemplo primero iniciará **db** y **redis**

`docker-compose up SERVICE` incluirá de forma automática la dependencia de los servicios. Es decir, `docker-compose up web` creará e iniciara db y redis.

```yaml
version: '3'
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

YAML: Entrypoint



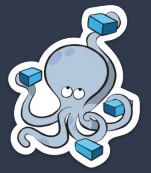
Docker Compose

```
entrypoint: /code/entrypoint.sh
```

El uso de la opción entrypoint sobreescribirá el `ENTRYPOINT` del Dockerfile.

Adicionalmente, de ser usado se ignoraran el `CMD`

YAML: Healtcheck



Docker Compose

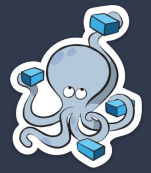
Ejecución automática de un comando de forma periódica verificando el estado del servicio.

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 1m30s
  timeout: 10s
  retries: 3
```

Para desactivar el healthcheck en imágenes:

```
healthcheck:
  disable: true
```

YAML: Links



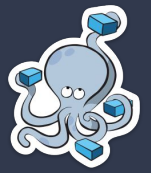
Docker Compose

Conecta servicios entre contenedores.

```
web:
  links:
   - db
   - db:database
   - redis
```

De usar Links y Network de forma simultánea, los servicios deben compartir al menos una red para poder comunicarse. Esta opción es ignorada cuando se hace un despliegue en modo swarm usando una versión 3 del archivo Compose.

YAML: Networks



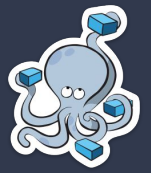
Docker Compose

Configuración de tipo global, define la red a la que pertenecerán los servicios, permite agregar la configuración IPV4/IPV6, drivers, límite de acceso.

```
services:
  some-service:
    networks:
     - some-network
     - other-network
```

```
version: '2'

services:
  proxy:
    build: ./proxy
    networks:
      - outside
      - default
  app:
    build: ./app
    networks:
      - default

networks:
  outside:
    external: true
```

YAML: Deploy



Docker Compose

Configuración para el despliegue de los servicios, solo aplica su uso en **modo swarm**.

Únicamente soportado desde la versión 3.

Es ignorado cuando se usan los comandos:

`docker-compose up` `docker-compose run`

```
version: '3'
services:
  redis:
    image: redis:alpine
    deploy:
      replicas: 6
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
```

Comandos CLI

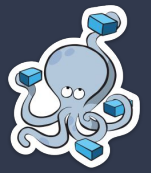
Docker Compose

```
Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help
```

https://docs.docker.com/compose/reference/overview/

```
docker-compose --help
```

Comandos: Build



Docker Compose

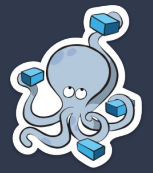
```
VPC-CCRUZ:docker-compose Cruz$ docker-compose build
db uses an image, skipping
Building web
Step 1/6 : FROM node:0.10
0.10: Pulling from library/node
b8c6812ba469: Pull complete
0adf07c73141: Pull complete
Status: Downloaded newer image for node:0.10
 ---> 0f9a0e09fbda
Step 2/6 : ADD package.json /code/
 ---> 3184ae0dc4f7
Removing intermediate container 05b617e2c481
Step 3/6 : WORKDIR /code
 ---> e1d7ac2ebd46
Removing intermediate container ec97c6a8237a
Step 4/6 : RUN npm install
 ---> Running in fe33f2e5f374
```

Opciones:

- Remueva contenedores intermedios
- No use cache cuando construya
- Siempre haga pull de las imágenes
- Variables de construcción

```
Options:
    --force-rm
    --no-cache
    --pull
    --build-arg key=val
```

Comandos: Up



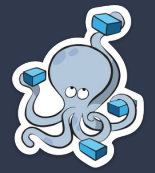
Docker Compose

```
$ docker-compose up
Creating network "composetest_default" with the default driver
Creating composetest_web_1 ...
Creating composetest_redis_1 ...
Creating composetest_web_1
Creating composetest_redis_1 ... done
Attaching to composetest_web_1, composetest_redis_1
web_1    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1    |  * Restarting with stat
web_1    |  * Debugger is active!
redis_1  | 1:M 17 Aug 22:11:10.483 # Server initialized
web_1    |  * Debugger PIN: 330-787-903
redis_1  | 1:M 17 Aug 22:11:10.483 * Ready to accept connections
```

```
Options:
    -d

    --no-deps

    --force-recreate
    --no-recreate

    --no-build
    --build

    --remove-orphans
    --scale SERVICE=NUM
```

Construye, inicia y conecta los contenedores a un servicio. También realiza la inicialización de los servicios linkeados y valida cambios en contenedores.

Comandos: Pull y Push



Docker Compose

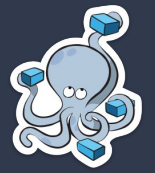
```
Usage: pull [options] [SERVICE...]

Options:
    --ignore-pull-failures
    --parallel
    --quiet
```

- Obtiene los cambios ignorando imágenes con errores
- Obtiene imágenes en paralelo
- Obtiene sin salida en consola del progreso

```
Usage: push [options] [SERVICE...]

Options:
    --ignore-push-failures
```

- Sube los cambios e ignora las imágenes con errores.

Comandos Útiles



Docker Compose

```
Usage: start [SERVICE...]
```

```
Usage: stop [options] [SERVICE...]
```

```
Usage: restart [options] [SERVICE...]
```

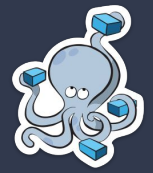
```
Usage: pause [SERVICE...]
```

```
Usage: unpause [SERVICE...]
```

Comandos son útiles al momento de manejar los servicios,
ya sea de forma singular o el grupo de servicios del compose

```
$ docker-compose stop
```

E1: CoffeeScript + MongoDB

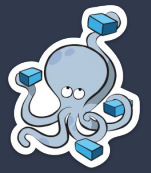


Docker Compose

https://github.com/twogg-git/docker-compose

- Dos servicios **web** y **db**
- Se establece la conexión con **links**
- Para el servicio Mongo solo es necesario definir la imagen y la versión
- Se expone el puerto 80

E1: CoffeeScript + MongoDB



Docker Compose

docker-compose.yaml

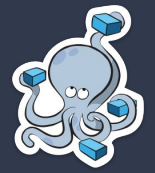
```
web:
  build: ./web
  ports:
    - "8080:8080"
  links:
    - db

db:
  image: mongo:2.6
```

Dockerfile servicio web

```
FROM node:0.10

ADD package.json /code/
WORKDIR /code
RUN npm install
ADD . /code

CMD ["./node_modules/.bin/coffee", "./server.coffee"]
```

No es necesario exponer los puertos de Mongo, ni definir un tipo de network.

E1: CoffeeScript + MongoDB



Docker Compose

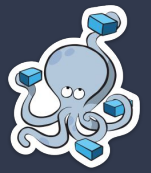
Construcción servicios, ejecución comando **docker-compose build**

```
VPC-CCRUZ:docker-compose Cruz$ docker-compose build
db uses an image, skipping
Building web
Step 1/6 : FROM node:0.10
 ---> 0f9a0e09fbda
Step 2/6 : ADD package.json /code/
 ---> Using cache
 ---> 3184ae0dc4f7
Step 3/6 : WORKDIR /code
 ---> Using cache
 ---> e1d7ac2ebd46
Step 4/6 : RUN npm install
 ---> Using cache
 ---> 56e70f6c2b73
Step 5/6 : ADD . /code
 ---> Using cache
 ---> 4151c35fb903
Step 6/6 : CMD ./node_modules/.bin/coffee ./server.coffee
 ---> Using cache
 ---> 0dfd36f23e46
Successfully built 0dfd36f23e46
```

Pasos:

1. NodeJS
2. Package JSON
3. CoffeeScript
4. NPM Install
5. Add code src
6. Run code

E1: CoffeeScript + MongoDB



Docker Compose

Ejecución **docker-compose up**, inicia todos los servicios

```
VPC-CCRUZ:docker-compose Cruz$ docker-compose up
Starting dockercompose_db_1
Starting dockercompose_web_1
Attaching to dockercompose_db_1, dockercompose_web_1
db_1    | mongod --help for help and startup options
```

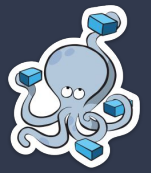
Detiene todos los servicios ya que no se usó el flag **-d** detached

```
db_1    | 2017-10-24T19:00:01.648+0000 [clientcursormon]  connections:5
^CGracefully stopping... (press Ctrl+C again to force)
Stopping dockercompose_web_1 ... done
Stopping dockercompose_db_1 ... done
```

Inicio de los servicios con el comando **docker-compose start**

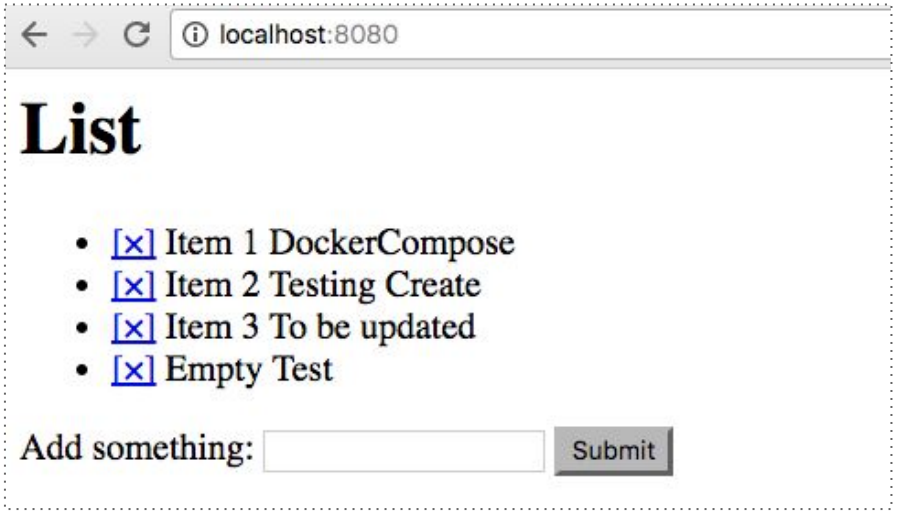
```
VPC-CCRUZ:docker-compose Cruz$ docker-compose start
Starting db ... done
Starting web ... done
```

E1: CoffeeScript + MongoDB



Docker Compose

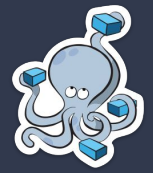
CoffeeScript en ejecución y conectado con MongoDB



Catherin Cruz

Medellín DevOps

E2: JAVA + Postgres +SMTP

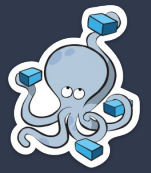


Docker Compose

https://github.com/twogg-git/docker-compose-java

- JAVA REST para recibir las peticiones de envíos de email
- DB para logueo de eventos con Postgres
- SMTP para el envío de emails
- Inicialización de base de datos mediante .sql

E2: JAVA + Postgres +SMTP



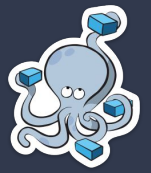
Docker Compose

docker-compose.yml

```
webapp:
  build: server
  ports:
    - "8080:8080"
  links:
    - database
    - smtp

database:
  build: database
  restart: always
  ports:
      - "5432:5432"
  environment:
      - DEBUG=true
      - POSTGRES_USER=emailboot
      - POSTGRES_PASSWORD=emailboot

smtp:
  build: smtp
  ports:
    - "25:25"
  environment:
    - maildomain=emailboot.net
    - smtp_user=user:pwd
    - messageSizeLimit=40960000
    - mailboxSizeLimit=40960000
```

Servicios: webapp, database, smtp. **Links:** database y smtp. **Único** archivo de configuración.

E2: JAVA + Postgres +SMTP



Docker Compose

Archivos Dockerfile por servicio:

JAVA Server

```
1  FROM tomcat:8.5-jre8
2  MAINTAINER twogg-git
3  ARG ARTIFACTS_DIR
4  RUN rm -rf /usr/local/tomcat/webapps/ROOT
5  COPY /webapp/emailboot.war /usr/local/tomcat/webapps/ROOT.war
```

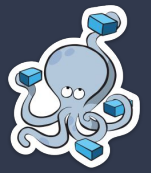
Postgres

```
1  FROM postgres:9.5
2  EXPOSE 5432
3  ADD script/emailboot.sql /docker-entrypoint-initdb.d/
```

SMTP

```
1  FROM catatnight/postfix
2  EXPOSE 25
```

E2: JAVA + Postgres +SMTP



Docker Compose

Ejecución docker-compose up

```
VPC-CCRUZ:docker-compose-java Cruz$ docker-compose up
dockercomposejava_smtp_1 is up-to-date
dockercomposejava_database_1 is up-to-date
Starting dockercomposejava_webapp_1
Attaching to dockercomposejava_smtp_1, dockercomposejava_database_1, dockercomposejava_webapp_1
```

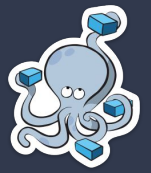
Inicialización base de datos gracias al archivo emailboot.sql

```
database_1  | CREATE TABLE
database_1  | ALTER TABLE
database_1  | server started
database_1  | CREATE DATABASE
```

Tomcat iniciado y listo para recibir peticiones en el puerto 8080

```
webapp_1    | 24-Oct-2017 20:27:51.527 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
webapp_1    | 24-Oct-2017 20:27:51.542 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
webapp_1    | 24-Oct-2017 20:27:51.548 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in 3862 ms
```

E2: JAVA + Postgres +SMTP



Docker Compose

CURL prueba del servicio REST

```
curl -X POST

'http://localhost:8080/v1/emails
    ?subject=subject
    &content=content
    &recipients=mailtest@gmail.com'

    -H 'cache-control: no-cache'
```

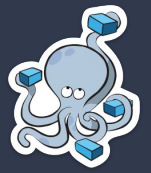
JSON respuesta POST


```
{
   "code": 202,
   "status": "ACCEPTED",
   "message": "Email task was accepted and sent to SMTP",
   "data": {
      "serial": "1508881028855",
      "from": "172.17.0.1",
      "mailsList": "test@mail.com",
      "subject": "Subject",
      "content": "Content"
   }
}
```


Log servicio webapp


```
webapp_1    | 24-Oct-2017 21:37:10.934 INFO [http-nio-8080-exec-7]
emailboot.rest.RestMail.postEmail Email task was accepted and sent
to SMTP{serial=1508881028855, from='172.17.0.1', mailsList='test@ma
il.com', subject='Subject', content='Content', warnings=null, malfo
rmedDirectEmails=null}
```


E2: JAVA + Postgres +SMTP



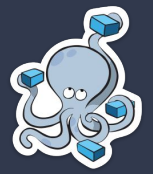
Docker Compose

http://localhost:8080/v1/logger/serial/1508881028855

Respuesta JSON llamada GET por serial de email


```
{
   "code": 200,
   "status": "OK",
   "url": "[GET] http://localhost:8080/v1/logger/serial/1508881028855",
   "message": "EmailBoot request response.",
   "data": [
      {
         "serial": "1508881028855",
         "subject": "Subject",
         "delivered": true,
         "content": "Email build and send to SMTP",
         "versionDate": "10/24/2017 21:37:10"
      }
   ]
}
```


E2: JAVA + Postgres +SMTP



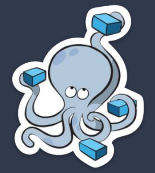
Docker Compose

http://localhost:8080/v1/logger?startDate=2017-01-01 00:00&endDate=2017-12-01 00:00

Respuesta JSON llamada GET por serial de email

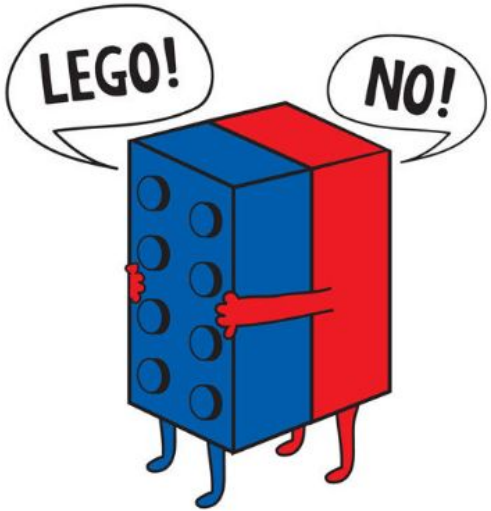

```
{
   "code": 200,
   "status": "OK",
   "url": "[GET] http://localhost:8080/v1/logger?startDate=2017-01-01%2000:00&endDate=2017-12-01%2000:00",
   "message": "EmailBoot request response.",
   "data": [
      {
         "serial": "1508877313501",
         "subject": "subject",
         "delivered": true,
         "content": "Email build and send to SMTP",
         "versionDate": "10/24/2017 20:35:14"
      },...
```


Referencias & Links

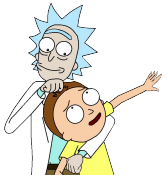


Docker Compose

https://docs.docker.com/compose
http://labs.play-with-docker.com/
http://training.play-with-docker.com/



DZONE. Orchestrating and Deploying Containers
Packt. Neependra Khare, Docker Cookbook
O'Reilly. Daniel Bryant, Continuous Delivery in Java
O'Reilly. Sebastien Goasguen, Docker in the Cloud



Catherine Cruz
Gracias y docker run!!!

Medellín DevOps