



Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos

Generación Automática de Código

Autor

César Hugo Bárzano Cruz



TRABAJO 1 DE EVALUACIÓN CONTINUA

—
2017/2018

Índice general

1. Introducción	7
2. Introducción 2	9
3. Soluciones Empresariales SOA	11
3.0.1. Google Cloud	11
3.0.2. Amazon Web Services	12
3.0.3. Microsoft Azure	13
3.0.4. mLab	13
3.0.5. Yahoo!	13
3.0.6. Ether Cloud Services	14
4. Aplicación Técnica SOA	15
4.1. El Problema	15
4.2. Análisis del Problema	15
4.2.1. Procesadores	17
4.2.2. Datacore	17
4.2.3. Storage	17
4.3. Especificación de la Solución	17
4.3.1. Requisitos de Información del Sistema	17
4.3.2. Requisitos Funcionales del Sistema	18
4.3.3. Requisitos No Funcionales del Sistema	18
4.4. Implementación y Tecnología	18
4.4.1. Procesadores	19
4.4.2. Datacore	19
4.4.3. Store	19
4.4.4. Virtualización	20
4.5. Construcción	20
5. Anexo	25
5.1. generator.py	25

Índice de figuras

4.1. Arquitectura SOA	16
4.2. make docker-datacore	21

Capítulo 1

Introducción

El presente documento representa la memoria formal para el trabajo de investigación de la asignatura Arquitecturas Orientadas a Servicios. Dicho trabajo consta de 2 partes diferenciadas.

En la primera parte, se darán ejemplos de arquitecturas orientadas a servicios en un contexto laboral y empresarial, haciendo distinción entre soluciones exitosas y soluciones no tan funcionales. En la segunda parte se abordará de una manera más práctica el análisis, diseño e implementación de una arquitectura orientada a servicios con el objetivo de solucionar el problema planteado con este tipo de arquitecturas así como las tecnologías relacionadas con ellas.

Capítulo 2

Introducción 2

En la actualidad, las tecnologías web así como las soluciones basadas en servicios cloud han tomado un gran peso tanto en nuestra vida cotidiana como para el mundo empresarial. El paradigma de servicios cloud denominado "La WEB 2.0" ha revolucionado la manera en la que la información llega al usuario. El valor aportado al mundo empresarial por el tratamiento de datos para la toma de decisiones junto con la transformación digital ha hecho que innumerables empresas se suban al carro de las nuevas tecnologías ya que una pequeña inversión en innovación marca una notable mejora en los procesos empresariales.

...etc...CONMPLETAR

Capítulo 3

Soluciones Empresariales SOA

Cada vez son más las soluciones basadas en arquitecturas orientadas a servicios que el usuario tiene a su disposición y cada vez son más diversas las necesidades tecnológicas que el usuario requiere. Por ello, en los últimos años, han surgido ciertos proveedores de servicios con soluciones basadas en SOA que se han convertido en el buque insignia de la tecnología cloud. Los proveedores de soluciones para "La WEB 2.0" que se tratarán a continuación son:

1. **Google Cloud Platform**
2. **Amazon Web Services**
3. **Microsoft Azure**
4. **Mlab**
5. **Yahoo!**

De manera adicional se comentará brevemente Ether Cloud Services, la plataforma de cloud híbrida con base de gobierno presentada por el banco BBVA para este 2018 en el congreso Ether.io de noviembre, plataforma en la que actualmente desempeña el rol de desarrollador de soluciones.

3.0.1. Google Cloud

Google Cloud Platform, es una plataforma en la nube que ha unificado las aplicaciones para el desarrollo de aplicaciones cloud ofrecidas por Google. Permite crear soluciones eficientes y escalables. Ejemplos de aplicaciones basadas en SOA pueden ser las propias ofertadas por Google ya que están basadas en su infraestructura cloud:

1. **Gmail**
2. **Google Drive**
3. **Google Sites**
4. **Google Contacts**
5. **Google Apps para educación**

Aunque Google pone a disposición de los usuarios bastantes aplicaciones o servicios de carácter gratuito, el principal problema para los desarrolladores de arquitecturas basadas en microservicios es la facturación. Esta puede ser por unidades de cómputo o almacenamiento por lo que un correcto análisis y diseño de la infraestructura virtual que va a soportar la arquitectura basada en servicios que el desarrollador o desarrolladores han de implementar, así como un análisis del tráfico o concurrencia que dicha SOA ha de soportar puede suponer una correcta decisión o no el usar los servicios cloud ofertados por Google.

Se considera una SOA exitosa debido al número de usuarios activos que dicha plataforma tiene en todo el mundo, así como la adecuada experiencia de usuario que ofrece.

3.0.2. Amazon Web Services

AWS es una colección de servicios cloud ofrecidos por Amazon bajo el abanico de su nube pública. (también llamados servicios web) que en conjunto forman una plataforma de computación en la nube, ofrecida a través de Internet. Algunos ejemplos exitosos de aplicaciones basadas en SOA que a su vez hacen uso de una arquitectura orientada a servicios como ofrece la plataforma de Amazon son:

1. **Dropbox**
2. **Foursquare**
3. **HootSuite**

La principal ventaja y a la vez problema de la nube de Amazon es su categorización de servicio por regiones o zonas de disponibilidad que se indican en la tabla de abajo. La división de sus usuarios en las regiones que por localización geográfica mejor servicio ofrezca produce una experiencia de usuario realmente buena, equilibrando el tráfico de su infraestructura cloud pero debido a las distintas legislaciones en cuanto a la protección de datos, hay ciertas regiones en las que el almacenamiento masivo de datos de carácter sensible conlleva ciertas responsabilidades. Por ejemplo un desarrollador

financiero no siempre podrá alojar sus servicios en la región de disponibilidad que el considere mejor para la latencia del mismo, si no que le vendrá impuesta por contrato o proyecto.

Se considera un ejemplo de arquitectura basada en servicios muy exitosa debido al gran número de usuarios que tiene en todo el mundo. De manera adicional, AWS facilita el aprendizaje de su tecnología a estudiantes proporcionando horas de cómputo y unidades de almacenamiento a los propietarios de cuentas de correo universitario.

TABLA ZONAS DISPONIBILIDAD

3.0.3. Microsoft Azure

Microsoft Azure es conjunto de servicios cloud de carácter empresarial para soluciones comerciales. Azure cuenta con un amplio abanico de servicios, herramientas y frameworks para la creación y administración de soluciones web a nivel mundial. Se le considera un triunfador en el mundo de los servicios cloud, con su gigantesca arquitectura basada en servicios se presenta como un pionero en el mundo de las soluciones cloud. A pesar de que Azure facilita el aprendizaje de su tecnología a los estudiantes concediendo paquetes de cómputo se la considera una de las plataformas cloud más caras en términos de cómputo y almacenamiento.

3.0.4. mLab

mLab se caracteriza por ser un servicio de base de datos cloud que ofrece capacidad de almacenamiento en forma de base de datos relacional, MongoDB. mLab proporciona a sus usuarios un servicio de base de datos no relacional totalmente administrado que se ejecuta sobre proveedores de servicio como los vistos anteriormente (Google, Amazon o Microsoft Azure)

Se considera un ejemplo de SOA exitoso debido a que el servicio ofertado es consumido por miles de desarrolladores cloud en todo el mundo para implementar soluciones SOA además de facilitar el aprendizaje de su tecnología de negocio ya que ofrece pequeñas parcelas de almacenamiento cloud de manera gratuita.

3.0.5. Yahoo!

Yahoo es un portal de internet que ofrece distintos servicios de información como correo electrónico, directorio de servicios web y principalmente motor de búsqueda. La arquitectura de este portal está basada en el paradigma de las arquitecturas basadas en microservicios pero se considera como un fracaso empresarial ya que en En 2009, Yahoo anunció que en el primer trimestre sus utilidades cayeron 78 % con respecto a igual periodo de 2008. La compañía despidió a más de 5 % de su personal, equivalente 700 empleados, que se sumaban a los 1500 anunciados el mismo año. La compañía no

podía hacer frente a Google, su principal rival por lo que en 2012 despidió a 2000 empleados, el 14 % de su equipo global.

3.0.6. Ether Cloud Services

ECS se presenta como el futuro competidor de AWS y Google Cloud para desarrolladores de servicios y soluciones financieras sobre escenarios cloud. La plataforma ECS presenta un conjunto de diversas disciplinas de servicios cloud con base de gobierno para el desarrollo y despliegue de aplicaciones basadas en servicios. La plataforma en si es una arquitectura SOA pura en la que un conjunto de piezas o servicios, implementando unas interfaces o End-points comunes son capaces de formar una red cloud a nivel global para ofrecer todo tipo de servicios para desarrolladores de aplicaciones financieras, analisis continuo de flujos de datos, categorización, aprovisionamiento, testing, cómputo y almacenamiento en todo el mundo, permitiendo asi que los procesos de negocio del banco cambien de modo BATCH a REAL TIME.

Capítulo 4

Aplicación Técnica SOA

Como segunda parte del trabajo de investigación se va a realizar la especificación, análisis e implementación de un sistema cuya arquitectura esta basada en servicios.

4.1. El Problema

Como desarrolladores de soluciones, cliente ha solicitado un sistema para el tratamiento y procesamiento de datos. Cliente indica que el producto ha de ser capaz de manejar distintos conjuntos de datos muy variados. De manera adicional, el producto resultante ha de ser capaz de realizar ciertas operaciones sobre los conjuntos de datos. Una de las principales preocupaciones del cliente es su escalabilidad ya que en etapas tempranas del proyecto, el sistema no tendrá mucho tráfico pero en versiones futuras, el número de usuarios a los que tendrá que dar servicio podría expresarse en terminos globales, por lo que el sistema ha de ser rápido, eficiente y facilmente escalable.

4.2. Análisis del Problema

Para abordar el problema planteado por cliente, se ha decidido utilizar una arquitectura basada en microservicios. Para ello, se pretende definir un bloque principal para la gestión de los conjuntos de datos o datasets. Cliente será capaz de interactuar con el bloque de gestión mediante una API-RESTFUL basada en protocolo http, lo que le permitirá consumir el servicio de datos y usar el sistema como back-end para implementar aplicaciones que consuman dicho servicio de tratamiento de datos de una manera sencilla y homogenea. De manera adicional y usando dicha interfaz REST, el cliente será capaz de realizar operaciones sobre los conjuntos de datos que el sistema es capaz de gestionar. Con el objetivo de optimizar el rendimiento y la eficiencia se ha decidido delegar el procesamiento de datos a otro servicio

totalmente independiente al bloque principal de gestión, por lo que tanto el bloque de gestión como los servicios esclavos encargados del tratamiento de datos, deberán implementar una API-REST privada. De esta forma, la experiencia que percibe el usuario será transparente al procesamiento que el sistema realice en su interior, ya que cliente solo conocerá la API publica del bloque de gestión.

Este paradigma permitirá al sistema escalar fácilmente e incluir nuevas operaciones o tratamientos para los datos que el bloque de gestión disponibiliza mediante la API pública a cliente. El siguiente diagrama muestra la arquitectura basada en servicios rest que se quiere desarrollar para solucionar el problema propuesto.

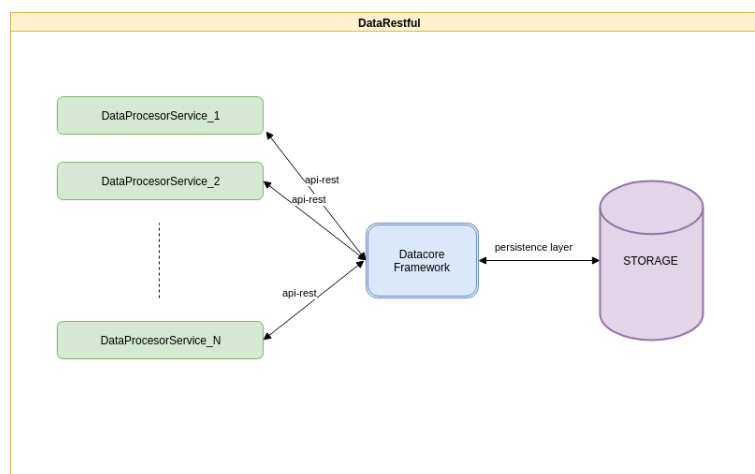


Figura 4.1: Arquitectura SOA

Podemos distinguir 3 elementos o conceptos principales dentro del sistema propuesto, bautizado como Datarestful:

1. **Procesadores**
2. **Datacore**
3. **Almacenamiento**

Para el usuario y su experiencia con el sistema, esto es transparente pues el realizará peticiones a Datarestful nivel sistema mediante la Api pública expuesta por el Datacore o framework a nivel sub-sistema y delegadas al resto del subsistemas dependiendo de si se requiere cómputo, almacenamiento o ambos mediante la Api privada no accesible para usuarios, si no solo para los servicios que formen parte de la aquitectura SOA que forma Datarestful.

4.2.1. Procesadores

Encargados del procesamiento y tratamiento de datos, son los encargados del procesamiento y tratamiento de datos, ofrecen servicio al core del sistema mediante Api-rest privada. Su objetivo es la computación por lo que carecen de capa de persistencia, reciben conjuntos de datos y devuelven el resultado de aplicar cierta operación sobre ese conjunto de datos.

4.2.2. Datacore

Framework encargado de gestionar los conjuntos de datos, su tarea es la de proporcionar al usuario una interfaz rest para manejar los datos y solicitar operaciones sobre los mismos. Carece de capacidad computacional pues todas las operaciones son delegadas a los servicios de procesamiento registrados en él. Datacore es el encargado de interactuar con la capa de persistencia implementando las operaciones CRUD necesarias para interactuar con la base de datos.

4.2.3. Storage

Capa de persistencia encargada del almacenamiento de datos.

4.3. Especificación de la Solución

Tras analizar el problema a resolver, en esta sección se realizará la especificación necesaria para su solución. Para ello, se van a definir los siguientes subsecciones:

4.3.1. Requisitos de Información del Sistema

Los requisitos de información se caracterizan por reunir la información relevante para el cliente, es decir, que debe gestionar y almacenar el sistema software.

RI-1. Dataset: Representación de cada uno de los conjuntos de datos en el sistema. Contenido: ID unico del conjunto de datos, título representativo del dataset, fecha de creación y el conjunto de datos en si.

RI-2. Operación: Representación de cada una de las operaciones computacionales que el sistema pone a disposición del usuario. Contenido: ID único de la operacion, operador o palabra reservada para realizar la operación, número de datasets que estan involucrados en la operación.

4.3.2. Requisitos Funcionales del Sistema

Como se define en la ingeniería de requisitos, los requisitos funcionales establecen los comportamientos del sistema.

RF-1. Gestión de datasets: El sistema será capaz gestionar el ciclo de vida de los datasets existentes en el sistema.

RF-1.1. El sistema permitirá añadir datasets.

RF-1.2. El sistema permitirá buscar datasets.

RF-1.3. El sistema permitirá modificar datasets.

RF-1.4. El sistema permitirá eliminar datasets.

RF-2. Tratamiento de Datos: El sistema deberá disponibilizar al usuario una serie de operaciones sobre los conjuntos de datos.

RF-2.1. El sistema permitirá realizar una operacion sobre uno o mas conjuntos de datos.

RF-2.2. El sistema permitirá consultar las operaciones disponibles.

RF-2.3. El sistema permitirá crear nuevos conjuntos de datos mediante el resultado de una o más operaciones.

RF-2.4. El sistema permitirá programar operaciones para ser realizadas en determinados momentos del día.

4.3.3. Requisitos No Funcionales del Sistema

Los requisitos no funcionales, se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar por el sistema, sino características de funcionamiento.

RNF-1 Necesitaremos que toda la información que se almacena sobre el sistema se mantenga segura, realizando copias de seguridad periódicas.

RNF-2 Necesitaremos que los conjuntos de datos esten disponibles de manera rápida.

RNF-3 Necesitaremos disponer de una buena conexión a internet ya que el sistema da servicio desde la nube.

4.4. Implementación y Tecnología

Una vez realizado el análisis del problema y especificada la posible solución es el momento de tomar las decisiones adecuadas para la correcta implementación de Datarestful. De manera general, los servicios que forman

el sistema han de implementar ciertos End-points o interfaces de comunicación que les permita interactuar entre ellos y con el usuario, para ello, se hará uso de HTTP (Hypertext Transfer Protocol) es un protocolo de comunicación que permite las transferencias de información en la web.

Como se vió en el apartado "Análisis del Problema" hay 3 bloques principales que trataremos a continuación:

4.4.1. Procesadores

Para la implementación de un procesador básico al que el sistema pueda solicitar operaciones sobre los datasets, se ha decidido utilizar python, mediante el micro-framework Flask para el desarrollo de aplicaciones web. Se ha decidido utilizar python como tecnología base debido a la gran variedad de librerías y algoritmos para el tratamiento de datos que existen para este lenguaje, muchas de ellas son adaptaciones de librerías pertenecientes al popular lenguaje para minería de datos denominado R.

La ventaja de la arquitectura basada en servicios que se va a implementar reside en que la tecnología utilizada es independiente al problema, es decir, el sistema puede delegar las operaciones computacionales a distintos servicios de procesamiento y estos pueden estar implementados con diversas tecnologías como python, javascript, scala...lo unico que han de hacer es implementar los End-points o interfaces rest que se definan en el API privada para que el sistema pueda delegar tareas computacionales a dichos servicios.

4.4.2. Datacore

Como se especificó anteriormente, Datacore es la pieza encargada de orquestar las peticiones de los usuarios con los servicios de cómputo y almacenamiento por eso se ha decidido implementar utilizando el lenguaje de programación GO. Esta tecnología esta orientada al desarrollo de servicios rest de altas prestaciones, optimizando el trafico y las peticiones concurrentes. Al tratarse de un lenguaje compilado, los ejecutables generados son multiplataforma lo que facilita su despliegue sobre diversas infraestructuras.

4.4.3. Store

Como capa de persistencia se ha decidido utilizar un modelo NO relacional, lo que permite almacenar datasets como documentos, esto significa que el enriquecimiento de las estructuras de datos almacenadas puede realizarse de manera simple. Para ello, se ha decidido utilizar el motor NoSql proporcionado por MongoDB. Este software permite optimizar las búsquedas de datasets mediante la indexación de sus colecciones de documentos lo que supone una ganancia en eficiencia con respecto a bases de datos relacionales orientadas a transacciones de objetos como Postgresql o Mysql.

4.4.4. Virtualización

Con el objetivo de facilitar el uso y la distribución de Datarestful como proyecto software se ha decidido utilizar el motor de virtualización ligera docker para ejecutar las piezas (datacore, procesador y store) a nivel subsistema sin necesidad de obligar al desarrollador a instalarse las dependencias necesarias o a tener un servicio de sistema de gestión de bases de datos no relacionales corriendo en su equipo, pues docker es auto-contenido, dando la posibilidad de generar imagenes funcionales de las piezas mediante el fichero Dockerfile contenido en el directorio raíz de los mismos. A nivel de sistema, se ha decidido utilizar el motor de orquestación para contenedores denominado docker-compose el cual permite al desarrollador orquestar las imagenes de cada una de las piezas para unificar su comportamiento y presentarse como un único sistema. Docker-compose hace uso del fichero docker-compose.yml en la raíz del proyecto. Este fichero permite definir de una manera clara y sencilla la arquitectura SOA presentada como solución. De manera adicional, no es necesario tener un fichero Dockerfile para mongoDB pues especificando la tag:version de las imaganes públicas disponibles en DOCKER-HUB, las imagenes especificadas, que no se encuentran generadas localmente en nuestro equipo serán descargadas, listas par funcionar. Para interactuar con el sistema Datarestful se ha preparado un CLI y un Makefile cuyo uso se ejemplificará con un caso de uso.

4.5. Construcción

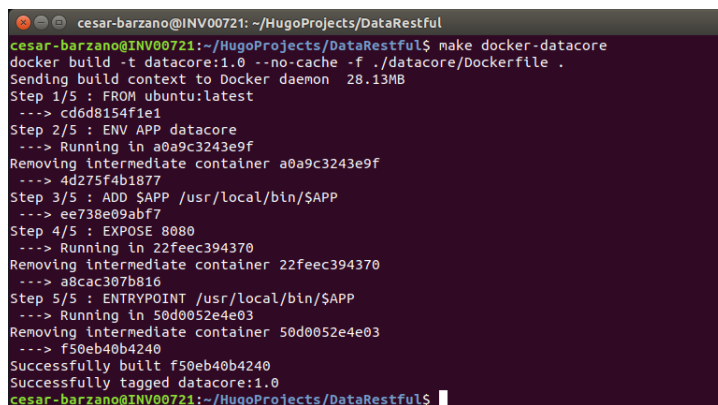
Con el objetivo de facilitar el arranque y la administración del sistema, se ha incluido el siguiente Makefile como herramienta de construcción. En dicho Makefile se especifican de manera sencillas las reglas necesarias para construir las imagenes docker necesarias para el sistema, arrancar, debuggear y detener el sistema. De manera adicional, podemos operar cada uno de los servicios que forma la arquitectura SOA de manera independiente a nivel de componente.

Listing 4.1: Makefile

```
1 #Makefile
2
3 build:
4     /usr/local/go/bin/go build ./datacore/src/datacore/main.go
5     mv ./datacore/main ./datacore/build/datacore
6
7 docker-datacore:
8     docker build -t datacore:1.0 --no-cache -f ./datacore/
9         Dockerfile .
10
11 docker-basicpythonprocessor:
12     docker build -t basicpythonprocessor:1.0 --no-cache -f ./
13         basicPythonProcessor/Dockerfile .
```

```
12
13 datarestful-up:
14     docker-compose up
15
16 datarestful-down:
17     docker-compose down
18
19 datarestful-show:
20     docker-compose logs
21
22 datacore-service-up:
23     docker-compose up datacore
24
25 processor-service-up:
26     docker-compose up pythonprocessor
27
28 mongo-service-up:
29     docker-compose up mongodb
30
31 datacore-service-down:
32     docker-compose down datacore
33
34 processor-service-down:
35     docker-compose down pythonprocessor
36
37 mongo-service-down:
38     docker-compose down mongodb
39
40 datacore-service-show:
41     docker-compose logs datacore
42
43 processor-service-show:
44     docker-compose logs pythonprocessor
45
46 mongo-service-show:
47     docker-compose logs mongodb
```

Por ejemplo para generar la imagen docker necesaria para el servicio datacore podemos ejecutar `make docker-datacore`, lo cual producirá la siguiente salida:



```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make docker-datacore
docker build -t datacore:1.0 --no-cache -f ./datacore/Dockerfile .
Sending build context to Docker daemon 28.13MB
Step 1/5 : FROM ubuntu:latest
--> cd6d8154f1e1
Step 2/5 : ENV APP datacore
--> Running in a0a9c3243e9f
Removing intermediate container a0a9c3243e9f
--> 4d275f4b1877
Step 3/5 : ADD $APP /usr/local/bin/$APP
--> ee738e09abf7
Step 4/5 : EXPOSE 8080
--> Running in 22fec394370
Removing intermediate container 22fec394370
--> a8cac307b816
Step 5/5 : ENTRYPOINT /usr/local/bin/$APP
--> Running in 50d0052e4e03
Removing intermediate container 50d0052e4e03
--> f50eb40b4240
Successfully built f50eb40b4240
Successfully tagged datacore:1.0
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 4.2: `make docker-datacore`

Puesto que el servicio datacore ha sido implementado con GO, lenguaje compilado capaz de generar ejecutables multiplataforma, la construcción de la imagen es rápida, pues no es necesario instalar nada en ella. En el caso de la imagen docker que representa al servicio de procesamiento python, es necesario instalar las dependencias necesarias, las cuales son establecidas en el Dockerfile que se muestra a continuación. La imagen que representa este fichero puede ser generada mediante `make docker-basicpythonprocessor`:

Listing 4.2: basicPythonProcessor/Dockerfile

```
1 FROM ubuntu:latest
2 RUN apt-get update
3 RUN apt-get install -y python
4 RUN apt-get install -y python-pip
5 RUN pip install --upgrade pip
6
7 ENV APP app.py
8 ADD ./ /usr/local/bin/
9
10 RUN pip install Flask==0.10.1
11 RUN pip install requests==2.19.1
12 RUN pip install urllib3==1.23
13
14 EXPOSE 5000
15 ENTRYPOINT python /usr/local/bin/$APP
```

Una vez generadas las imagenes necesarias para los servicios que componen Datarestful podemos levantar la arquitectura completa, para ello se ha definido el siguiente `docker-compose.yml`

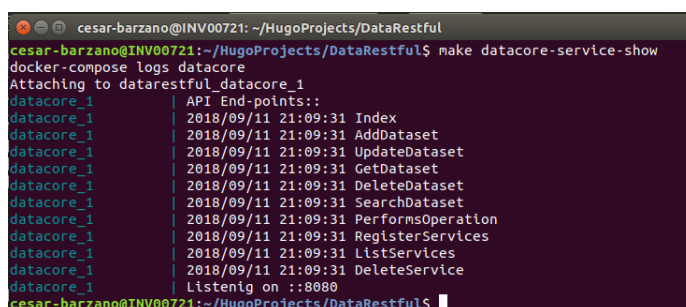
Listing 4.3: docker-compose.yml

```
1
2 version: "2"
3
4 services:
5   mongodb:
6     image: mongo:3.5.6
7     ports:
8       - 27019:27017
9
10  datacore:
11    image: datacore:1.0
12    depends_on:
13      - mongodb
14    ports:
15      - "8080:8080"
16
17  pythonprocessor:
18    image: basicpythonprocessor:1.0
19    depends_on:
20      - datacore
21    ports:
22      - "5000:5000"
```

Como se puede observar, para la capa de persistencia se usa una imagen

mongoDB oficial, por lo que no es necesaria generarla mediante un Dockerfile, tan solo es necesaria tenerla presente en nuestro sistema o descargarla en caso contrario. En este fichero se especifica el comportamiento del sistema, es decir, el orden en el que nuestros servicios han de estar disponibles y el orden en el que nuestros servicios han de detenerse. El servicio que representa la capa de persistencia (mongoDB) será el primero en arrancar, pues el servicio datacore depende de él, siendo este el siguiente en arrancar. El último servicio en arrancar será el procesador python pues depende de datacore para funcionar. Este es el comportamiento nominal, pues datacore necesita mongoDB ya que es el encargado de interactuar con la base de datos y el resto de procesadores (en este caso solo uno) necesitan a datacore pues la primera acción que realizan estos servicios es el de registrarse como servicio disponible de procesamiento. Cuando uno de estos servicios sufre un fallo y se detiene, se eliminará de la colección de servicios disponibles para el usuario, pues no será capaz de atender peticiones de ninguna clase. Datarestful puede ser levantado mediante `make datarestful-up`

Se puede apreciar que todos los logs de los servicios que acabamos de levantar se muestran por la misma salida estandar, lo que puede dificultar las tareas debug, para ello, podemos consultar los logs de cada servicio de manera independiente, por ejemplo con `make datacore-service-show` es posible chequear los logs del servicio datacore:



```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make datacore-service-show
docker-compose logs datacore
Attaching to datarestful_datacore_1
datacore_1 | API End-points::
datacore_1 | 2018/09/11 21:09:31 Index
datacore_1 | 2018/09/11 21:09:31 AddDataset
datacore_1 | 2018/09/11 21:09:31 UpdateDataset
datacore_1 | 2018/09/11 21:09:31 GetDataset
datacore_1 | 2018/09/11 21:09:31 DeleteDataset
datacore_1 | 2018/09/11 21:09:31 SearchDataset
datacore_1 | 2018/09/11 21:09:31 PerformsOperation
datacore_1 | 2018/09/11 21:09:31 RegisterServices
datacore_1 | 2018/09/11 21:09:31 ListServices
datacore_1 | 2018/09/11 21:09:31 DeleteService
datacore_1 | Listenig on ::8080
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 4.3: `make datacore-service-show`

Bibliografía

- [1] XML, *XML Format* <https://www.w3.org/XML/>
- [2] JSON, *JSON Format* <https://www.json.org/>
- [3] PYTHON, *Python 2.7.6 language* <https://www.python.org/download/releases/2.7.6/>
- [4] IMPORT JSON, *JSON encoder and decoder* <https://docs.python.org/2/library/json.html>
- [5] IMPORT XMLTODIC, *Makes working with XML feel like you are working with JSON* <https://pypi.python.org/pypi/xmltodict>
- [6] IMPORT SYS, *System-specific parameters and functions* <https://docs.python.org/2/library/sys.html>
- [7] IMPORT OS, *Miscellaneous operating system interfaces* <https://docs.python.org/2/library/os.html>
- [8] IMPORT GETOPT, *C-style parser for command line options* <https://docs.python.org/2/library/getopt.html>

Capítulo 5

Anexo

5.1. `generator.py`