



# Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos

---

Service-Oriented Architecture

**Autor**

César Hugo Bárzano Cruz



TRABAJO DE INVESTIGACIÓN SOA

—  
2017/2018



# Índice general

<b>1. Introducción</b>	<b>7</b>
<b>2. Soluciones SOA</b>	<b>9</b>
2.0.1. Google Cloud . . . . .	10
2.0.2. Amazon Web Services . . . . .	10
2.0.3. Microsoft Azure . . . . .	11
2.0.4. mLab . . . . .	12
2.0.5. Yahoo! . . . . .	12
2.0.6. Ether Cloud Services . . . . .	12
<b>3. Aplicación Técnica SOA</b>	<b>13</b>
3.1. El Problema . . . . .	13
3.2. Análisis del Problema . . . . .	13
3.2.1. Procesadores . . . . .	15
3.2.2. Datacore . . . . .	15
3.2.3. Storage . . . . .	15
3.3. Especificación de la Solución . . . . .	15
3.3.1. Requisitos de Información del Sistema . . . . .	15
3.3.2. Requisitos Funcionales del Sistema . . . . .	16
3.3.3. Requisitos No Funcionales del Sistema . . . . .	16
3.4. Implementación y Tecnología . . . . .	17
3.4.1. Procesadores . . . . .	17
3.4.2. Datacore . . . . .	17
3.4.3. Store . . . . .	17
3.4.4. Virtualización . . . . .	18
3.5. Construcción . . . . .	18
3.6. Caso de Uso . . . . .	23
3.6.1. Step-1: Servicios Disponibles . . . . .	25
3.6.2. Step-2: Gestión de Datasets . . . . .	26
3.6.3. Step-3: Operaciones . . . . .	28
3.7. Conclusiones . . . . .	30
<b>4. Entrega</b>	<b>31</b>



# Índice de figuras

2.1. Regiones de Diponibilidad . . . . .	11
3.1. Arquitectura SOA . . . . .	14
3.2. make docker-datacore . . . . .	20
3.3. make datarestful-up . . . . .	22
3.4. make datacore-service-show . . . . .	22
3.5. make processor-service-show . . . . .	23
3.6. make datarestful-down . . . . .	23
3.7. cli . . . . .	25
3.8. Step1- http://\$HOST:\$PORT/Services/ . . . . .	25
3.9. Step 1: http://\$HOST:\$PORT/Services/ . . . . .	26
3.10. make datacore-service-show . . . . .	26
3.11. Step 2: http://\$HOST:\$PORT/ . . . . .	26
3.12. Step 2: http://\$HOST:\$PORT/ . . . . .	27
3.13. Step 2: http://\$HOST:\$PORT/ . . . . .	27
3.14. Step 2: http://\$HOST:\$PORT/datasets/\$ID . . . . .	27
3.15. Step 2: http://\$HOST:\$PORT/Search/\$T . . . . .	27
3.16. Step 2: http://\$HOST:\$PORT/UpdateDataset . . . . .	28
3.17. Step 2: http://\$HOST:\$PORT/deleteDataset/\$ID . . . . .	28
3.18. Step 3: http://\$HOST:\$PORT/Operation/ . . . . .	29
3.19. Step 3: http://\$HOST:\$PORT/Operation/ . . . . .	29
3.20. Step 3: make datacore-service-show . . . . .	30
3.21. Step 3: make processor-service-show . . . . .	30



# 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 bloques principales.

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





## Capítulo 2

# Soluciones SOA

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 sus procesos de negocio.

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 y servicios de la información que tanto usuarios como empresas 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 las tecnologías cloud. Los proveedores de soluciones SOA 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 cloud híbrida con base de gobierno presentada por el banco BBVA para este 2018 en el evento Ether.io de noviembre, plataforma en la que actualmente desempeña el rol de desarrollador de soluciones.

### 2.0.1. Google Cloud

Google Cloud[?], 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 estan 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 caracter 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.

### 2.0.2. Amazon Web Services

AWS[2] 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 él 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.

Code	Nombre
us-east-1	US East (N. Virginia)
us-east-2	EE.UU. Este (Ohio)
us-west-1	EE.UU. Oeste (Norte de California)
us-west-2	EE.UU. Oeste (Oregón)
ca-central-1	Canadá (Central)
eu-central-1	UE (Fráncfort)
eu-west-1	UE (Irlanda)
eu-west-2	UE (Londres)
eu-west-3	UE (París)
ap-northeast-1	Asia Pacífico (Tokio)
ap-northeast-2	Asia Pacífico (Seúl)
ap-northeast-3	Asia Pacífico (Osaka-local)
ap-southeast-1	Asia Pacífico (Singapur)
ap-southeast-2	Asia Pacífico (Sídney)
ap-south-1	Asia Pacífico (Mumbai)
sa-east-1	América del Sur (São Paulo)

Figura 2.1: Regiones de Disponibilidad

### 2.0.3. Microsoft Azure

Microsoft Azure[3] 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.

#### **2.0.4. mLab**

mLab[4] 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.

#### **2.0.5. Yahoo!**

Yahoo[5] 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.

#### **2.0.6. Ether Cloud Services**

ECS[6] 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 sí 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, análisis continuo de flujos de datos, categorización, aprovisionamiento, testing, cómputo y almacenamiento en todo el mundo, permitiendo así que los procesos de negocio del banco cambien de modo BATCH a REAL-TIME.

## Capítulo 3

# 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.

### 3.1. 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.

### 3.2. 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 interaccionar con el bloque de gestión mediante una API-RESTFUL[7] basada en protocolo HTTP[8], 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 pública 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.

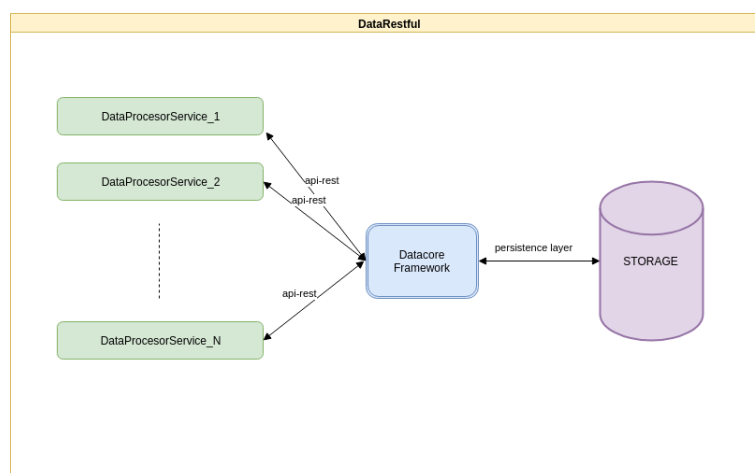


Figura 3.1: Arquitectura SOA

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.

### 3.2.1. 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.

### 3.2.2. 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.

### 3.2.3. Storage

Capa de persistencia encargada del almacenamiento de datos.

## 3.3. 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 sub-secciones:

### 3.3.1. 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, datasets que estan involucrados en la operación.

**RI-3. Servicio:** Representación de cada uno de los servicios operacionales registrados en el sistema, capaces de realizar operaciones sobre conjuntos

de datos. Contenido: ID único del servicio, Título del servicio, url mediante la cual se ofrece el servicio.

### 3.3.2. 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.

### 3.3.3. 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.



### 3.4. 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:

#### 3.4.1. Procesadores

Para la implementación de un procesador básico al que el sistema pueda solicitar operaciones sobre los datasets, se ha decidido utilizar Python[9], mediante el micro-framework Flask[10] 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 único 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.

#### 3.4.2. 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[11]. Esta tecnología está orientada al desarrollo de servicios rest de altas prestaciones, optimizando el tráfico y las peticiones concurrentes. Al tratarse de un lenguaje compilado, los ejecutables generados son multiplataforma lo que facilita su despliegue sobre diversas infraestructuras.

#### 3.4.3. 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[12]. 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.

#### 3.4.4. 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[12] 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 imágenes 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 imágenes 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 imágenes públicas disponibles en DOCKER-HUB, las imágenes especificadas, que no se encuentren generadas localmente en nuestro equipo serán descargadas, listas para funcionar. Para interactuar con el sistema Datarestful se ha preparado un CLI y un Makefile cuyo uso se ejemplificará con un caso de uso.

### 3.5. 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 imágenes 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 3.1: 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 .
14
15 datarestful-up:
16     docker-compose up
17
18 datarestful-down:
19     docker-compose down
20
21 datarestful-show:
22     docker-compose logs
23
24 datacore-service-up:
25     docker-compose up datacore
26
27 processor-service-up:
28     docker-compose up pythonprocessor
29
30 mongo-service-up:
31     docker-compose up mongodb
32
33 datacore-service-down:
34     docker-compose down datacore
35
36 processor-service-down:
37     docker-compose down pythonprocessor
38
39 mongo-service-down:
40     docker-compose down mongodb
41
42 datacore-service-show:
43     docker-compose logs datacore
44
45 processor-service-show:
46     docker-compose logs pythonprocessor
47
48 mongo-service-show:
49     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 3.2: 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 3.2: 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
8
9 ENV SERVICE_URL http://pythonprocessor:5000/basicOperator
10 ENV DATARESTFUL_URL http://datacore:8080
11 ENV APP app.py
12 ADD . /usr/local/bin/
13
14 RUN pip install Flask==0.10.1
15 RUN pip install requests==2.19.1
16 RUN pip install urllib3==1.23
17
18
19 EXPOSE 5000
20 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 3.3: 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**.

```

cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make datarestful-up
docker-compose up
Creating network "datarestful_default" with the default driver
Creating datarestful_mongodb_1 ... done
Creating datarestful_datacore_1 ... done
Creating datarestful_pythonprocessor_1 ... done
Attaching to datarestful_mongodb_1, datarestful_datacore_1, datarestful_pythonprocessor_1
mongodb_1 | about to fork child process, waiting until server is ready for connections.
mongodb_1 | forked process: 23
mongodb_1 | 2018-09-12T16:36:48.862+0000 I CONTROL [main] ***** SERVER RESTARTED *****
datacore_1 | API End-points::
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] MongoDB starting : pid=23 p
ort=27017 dbpath=/data/db 64-bit host=95edf43c609e
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] db version v3.5.6
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] git version: 6408dcd1b5f4fa
1747fa2acac50b8cd004343ca7
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.
0.1t 3 May 2016
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] allocator: tcmalloc
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] modules: none
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] build environment:
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] distmod: debian81
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] distarch: x86_64
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] target_arch: x86_64
datacore_1 | 2018/09/12 16:36:53 Index
datacore_1 | 2018/09/12 16:36:53 AddDataset
datacore_1 | 2018/09/12 16:36:53 UpdateDataset
datacore_1 | 2018/09/12 16:36:53 GetDataset
datacore_1 | 2018/09/12 16:36:53 DeleteDataset
datacore_1 | 2018/09/12 16:36:53 SearchDataset
mongodb_1 | 2018-09-12T16:36:48.866+0000 I CONTROL [initandlisten] options: { net: { bindIp: "
127.0.0.1", port: 27017, ssl: { mode: "disabled" } }, processManagement: { fork: true, pidFilePath: "/tm
p/tmp.MHPxnrB29X" }, systemLog: { destination: "file", logAppend: true, path: "/proc/1/fd/1" } }
mongodb_1 | 2018-09-12T16:36:48.869+0000 I STORAGE [initandlisten]
mongodb_1 | 2018-09-12T16:36:48.869+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS f
ilesystem is strongly recommended with the WiredTiger storage engine
mongodb_1 | 2018-09-12T16:36:48.869+0000 I STORAGE [initandlisten] ** See http://doch
ub.mongodb.org/core/prodnotes-filesystem

```

Figura 3.3: 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 o los del servicio de procesamiento:

```

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 | 2018/09/12 16:36:53 Index
datacore_1 | 2018/09/12 16:36:53 AddDataset
datacore_1 | 2018/09/12 16:36:53 UpdateDataset
datacore_1 | 2018/09/12 16:36:53 GetDataset
datacore_1 | 2018/09/12 16:36:53 DeleteDataset
datacore_1 | 2018/09/12 16:36:53 SearchDataset
datacore_1 | 2018/09/12 16:36:53 PerformsOperation
datacore_1 | 2018/09/12 16:36:53 RegisterServices
datacore_1 | 2018/09/12 16:36:53 ListServices
datacore_1 | 2018/09/12 16:36:53 DeleteService
datacore_1 | Listenig on ::8080
datacore_1 | 2018/09/12 16:36:58 [123 34 117 114 108 34 58 32 34 92 34 1
04 116 116 112 58 47 47 112 121 116 104 111 110 112 114 111 99 101 115 115 111 1
14 58 53 48 48 48 47 98 97 115 105 99 79 112 101 114 97 116 111 114 92 34 34 44
32 34 116 105 116 108 101 34 58 32 34 98 97 115 105 99 80 121 116 104 111 110 80
114 111 99 101 115 115 111 114 34 125]
datacore_1 | 2018/09/12 16:36:58 { basicPythonProcessor "http://pythonpr
ocessor:5000/basicOperator"}
datacore_1 | Added New Service - basicPythonProcessor
cesar-barzano@INV00721:~/HugoProjects/DataRestful$

```

Figura 3.4: make datacore-service-show

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make processor-service-show
docker-compose logs pythonprocessor
Attaching to datarestful_pythonprocessor_1
pythonprocessor_1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.5: make processor-service-show

Se puede observar que los servicios no han atendido ninguna request, pues solo los hemos arrancado, y la única petición atendida ha sido por el datacore, realizada por el servicio de procesamiento para darse de alta como servicio disponible.

Finalmente, cuando se quiera detener Datarestful podemos emplear las reglas down, por ejemplo, podemos detener el sistema completo mediante **make datarestful-down** donde se aprecia como los servicios son detenidos en el orden adecuado, primero el procesador básico, seguido del servicio datacore y finalmente la instancia de mongo:

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make datarestful-down
docker-compose down
Stopping datarestful_pythonprocessor_1 ... done
Stopping datarestful_datacore_1 ... done
Stopping datarestful_mongodb_1 ... done
Removing datarestful_pythonprocessor_1 ... done
Removing datarestful_datacore_1 ... done
Removing datarestful_mongodb_1 ... done
Removing network datarestful_default
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.6: make datarestful-down

### 3.6. Caso de Uso

Con el objetivo de mostrar las funcionalidades implementadas, se va a ejecutar un caso de uso. Al tratarse de una arquitectura basada en servicios REST que carece de frontal, se pueden utilizar herramientas como postman para realizar las distintas peticiones que un usuario de Datarestful realizaría de manera nominal. Teniendo en cuenta herramientas como esta, finalmente se ha decidido implementar el siguiente CLI para interactuar con el sistema mediante el uso del comando curl con funciones de interprete bash:

Listing 3.4: resources/cli/datacli.sh

```
1
2 #!/usr/bin/env bash
3 HOST=localhost
4 PORT=8080
5
6 function Datarestful_index() {
7     curl -X GET \
8         "http://$HOST:$PORT/" \
9         -H 'Content-Type: application/json' \
```

```

10     -H 'Accept: application/json'
11 }
12
13 function Datarestful_addDataset() {
14     TITLE=$1
15     DATA=$2
16     curl -X POST \
17         "http://$HOST:$PORT/AddDataset" \
18         -H 'Content-Type: application/json' \
19         -H 'Accept: application/json' \
20         -d '{
21             "title": "'$TITLE'",
22             "data": ["1", "2", "3"]
23         }'
24 }
25
26 function Datarestful_getDataset() {
27     ID=$1
28     curl -X GET \
29         "http://$HOST:$PORT/datasets/$ID" \
30         -H 'Content-Type: application/json' \
31         -H 'Accept: application/json'
32 }
33
34 function Datarestful_getDatasetByTitle() {
35     T=$1
36     curl -X GET \
37         "http://$HOST:$PORT/Search/$T" \
38         -H 'Content-Type: application/json' \
39         -H 'Accept: application/json'
40 }
41
42 function Datarestful_deleteDataset() {
43     ID=$1
44     curl -X DELETE \
45         "http://$HOST:$PORT/deleteDataset/$ID" \
46         -H 'Content-Type: application/json' \
47         -H 'Accept: application/json'
48 }
49
50 function Datarestful_updateDataset() {
51     ID=$1
52     TITLE=$2
53     #DATA=[1, 2, 3]
54     curl -X PUT \
55         "http://$HOST:$PORT/UpdateDataset" \
56         -H 'Content-Type: application/json' \
57         -H 'Accept: application/json' \
58         -d '{
59             "_id": "'$ID'",
60             "title": "'$TITLE'",
61             "data": ["1111", "2222", "3333"]
62         }'
63 }
64
65 function Datarestful_listservices() {
66     curl -X GET \
67         "http://$HOST:$PORT/Services/" \
68         -H 'Content-Type: application/json' \
69         -H 'Accept: application/json'
70 }
71

```

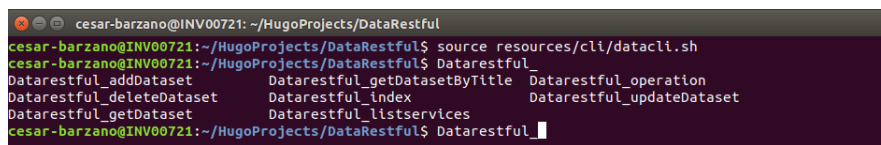


```

72 function Datarestful_operation() {
73     ID=$1
74     VALUE=$2
75     OPE=$3
76     URL=$4
77     curl -X POST \
78         "http://$HOST:$PORT/Operation/" \
79         -H 'Content-Type: application/json' \
80         -H 'Accept: application/json' \
81         -d '{
82             "dataset_id": "'$ID'",
83             "value": "'$VALUE'",
84             "operator": "'$OPE'",
85             "service_url": "'$URL'"
86         }'
87 }

```

Para habilitar este CLI es necesario abrir una nueva terminal y ejecutar **source resources/cli/datacli.sh** en la raíz del proyecto, a continuación, si escribimos **Data\_** e intentamos tabular para que nos auto-complete la terminal, podemos confirmar que efectivamente las operaciones están disponibles solo para ser ejecutadas desde esta terminal.



```

cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ source resources/cli/datacli.sh
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful
Datarestful_addDataset      Datarestful_getDatasetByTitle  Datarestful_operation
Datarestful_deleteDataset  Datarestful_index             Datarestful_updateDataset
Datarestful_getDataset     Datarestful_listservices
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_

```

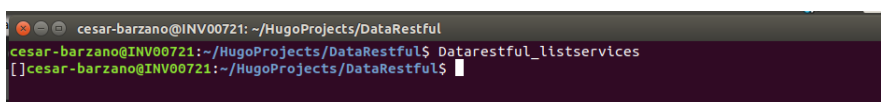
Figura 3.7: cli

Por simplicidad, algunos de los datos que son enviados en el body de las peticiones son estáticos. Para mostrar la usabilidad del sistema y representar un caso de uso, se van a enumerar una serie de pasos o steps con las distintas peticiones que se van a realizar.

### 3.6.1. Step-1: Servicios Disponibles

Para comenzar, solo hemos levantado el servicio **datacore** mediante **make datacore-service-up**, para que el usuario pueda gestionar los datasets y consultar los servicios de procesamiento disponibles.

El usuario de **Datarestful** desea conocer los servicios de procesamiento que tiene disponibles, para ello, desde la terminal con el cli cargado, ejecuta **Datarestful\_listservices** y obtiene una lista vacía, pues aun no hemos levantado el servicio de procesamiento.



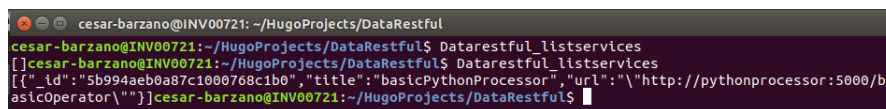
```

cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_listservices
[]cesar-barzano@INV00721:~/HugoProjects/DataRestful$

```

Figura 3.8: Step1- http://\$HOST:\$PORT/Services/

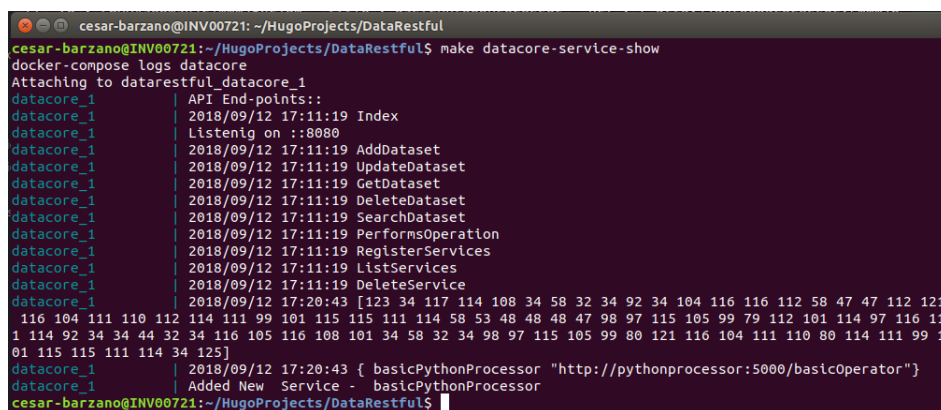
Si levantamos el servicio de procesamiento mediante **make processor-service-up** y el usuario vuelve a ejecutar **Datarestful listservices** desde el cli, esta vez si obtiene el servicio que acabamos de levantar, con la información necesaria para futuras operaciones:



```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful$ Datarestful listservices
[{"_id": "5b994aeb0a87c1000768c1b0", "title": "basicPythonProcessor", "url": "http://pythonprocessor:5000/basicOperator/"}]
```

Figura 3.9: Step 1: `http://$HOST:$PORT/Services/`

De manera adicional, podemos chequear los logs del servicio `datacore` para validar que efectivamente se ha registrado un nuevo servicio de procesamiento. **NOTA:** Estos logs corresponden a los servicios, un usuario nominal no dispondría de estas trazas.

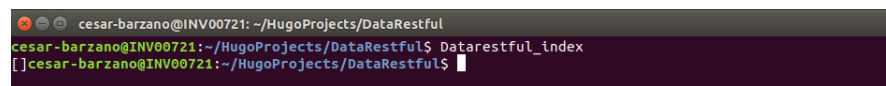


```
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make datacore-service-show
docker-compose logs datacore
Attaching to datarestful_datacore_1
datacore_1 | API End-points::
datacore_1 | 2018/09/12 17:11:19 Index
datacore_1 | Listening on ::8080
datacore_1 | 2018/09/12 17:11:19 AddDataset
datacore_1 | 2018/09/12 17:11:19 UpdateDataset
datacore_1 | 2018/09/12 17:11:19 GetDataset
datacore_1 | 2018/09/12 17:11:19 DeleteDataset
datacore_1 | 2018/09/12 17:11:19 SearchDataset
datacore_1 | 2018/09/12 17:11:19 PerformsOperation
datacore_1 | 2018/09/12 17:11:19 RegisterServices
datacore_1 | 2018/09/12 17:11:19 ListServices
datacore_1 | 2018/09/12 17:11:19 DeleteService
datacore_1 | 2018/09/12 17:20:43 [123 34 117 114 108 34 58 32 34 92 34 104 116 116 112 58 47 47 112 121
116 104 111 110 112 114 111 99 101 115 115 111 114 58 53 48 48 48 47 98 97 115 105 99 79 112 101 114 97 116 11
1 114 92 34 34 44 32 34 116 105 116 108 101 34 58 32 34 98 97 115 105 99 80 121 116 104 111 110 80 114 111 99 1
01 115 115 111 114 34 125]
datacore_1 | 2018/09/12 17:20:43 { basicPythonProcessor "http://pythonprocessor:5000/basicOperator"}
datacore_1 | Added New Service - basicPythonProcessor
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.10: `make datacore-service-show`

### 3.6.2. Step-2: Gestión de Datasets

Una vez que el usuario de Datarestful dispone de servicios de procesamiento, lo siguiente que ha de hacer es nutrir al sistema de datos sobre los que realizar procesamiento. Para ello, desde el cli puede obtener todas los datasets de los que dispone ejecutando **Datarestful index**



```
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful index
[cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.11: Step 2: `http://$HOST:$PORT/`

Puesto que aun no se ha dado de alta ningun dataset, la consulta devuelve una lista vacia. Para dar de alta un dataset generico, desde el cli, el

usuario puede ejecutar **Datarestful\_addDataset step2** enviado una petición POST sobre el end-point **http://\$HOST:\$PORT/AddDataset** y de nuevo **Datarestful\_index** devolviendo ahora el dataset estático que se ha definido en el cli:

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_addDataset step2
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id":"5b99556b8fc0430006db1a76","title":"step2","data":["1","2","3"]}]cesar-barzano@INV00721:~/HugoProjects/
DataRestful$
```

Figura 3.12: Step 2: **http://\$HOST:\$PORT/**

El usuario ha realizado varias llamadas a **Datarestful\_addDataset** con distintos títulos por lo que si consulta todas las datasets mediante **Datarestful\_index** obtendrá:

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id":"5b99556b8fc0430006db1a76","title":"step2","data":["1","2","3"]},{ "_id":"5b99563e8fc0430006db1a77","tit
le":"step2_another","data":["1","2","3"]},{ "_id":"5b9956428fc0430006db1a78","title":"step2_another_one","data":
["1","2","3"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.13: Step 2: **http://\$HOST:\$PORT/**

El usuario puede filtrar los datasets disponibles mediante ID usando **Datarestful\_getDataset ID**

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id":"5b99556b8fc0430006db1a76","title":"step2","data":["1","2","3"]},{ "_id":"5b99563e8fc0430006db1a77","title":"
step2_another","data":["1","2","3"]},{ "_id":"5b9956428fc0430006db1a78","title":"step2_another_one","data":["1","2","
3"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_getDataset 5b99563e8fc0430006db1a77
{"_id":"5b99563e8fc0430006db1a77","title":"step2_another","data":["1","2","3"]}]cesar-barzano@INV00721:~/HugoPr
jects/DataRestful$
```

Figura 3.14: Step 2: **http://\$HOST:\$PORT/datasets/\$ID**

El usuario puede filtrar los datasets disponibles mediante su título usando **Datarestful\_getDatasetByTitle title**

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id":"5b99556b8fc0430006db1a76","title":"step2","data":["1","2","3"]},{ "_id":"5b99563e8fc0430006db1a77","title":"
step2_another","data":["1","2","3"]},{ "_id":"5b9956428fc0430006db1a78","title":"step2_another_one","data":["1","2","
3"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_getDataset 5b99563e8fc0430006db1a77
{"_id":"5b99563e8fc0430006db1a77","title":"step2_another","data":["1","2","3"]}]cesar-barzano@INV00721:~/HugoPr
jects/DataRestful$ Datarestful_getDatasetByTitle step2_another_one
[{"_id":"5b9956428fc0430006db1a78","title":"step2_another_one","data":["1","2","3"]}]cesar-barzano@INV00721:~/HugoPr
jects/DataRestful$
```

Figura 3.15: Step 2: **http://\$HOST:\$PORT/Search/\$T**

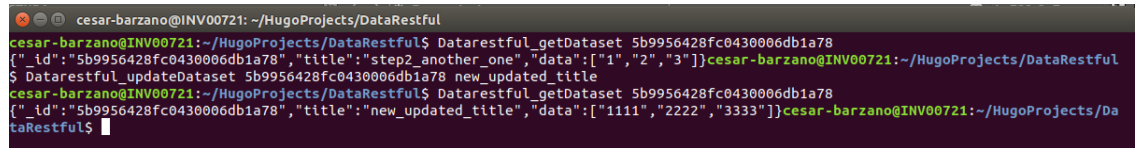
El usuario puede actualizar los datasets disponibles mediante **Datarestful\_updateDataset \$ID \$TITLE** donde \$ID corresponde a la `_id` del dataset

que será actualizado, y \$TITLE al nuevo titulo. En esta operación y por simplificar, los datos con los que se actualizará la dataset en cuestión son estaticos, pero bastaría con componer una PUT request con la siguiente estructura en su body:

Listing 3.5: body

```

1 {
2     "_id": "dataset_id_to_update",
3     "title": "new_title",
4     "data": ["9999", "888", "77", "6"]
5 }
```

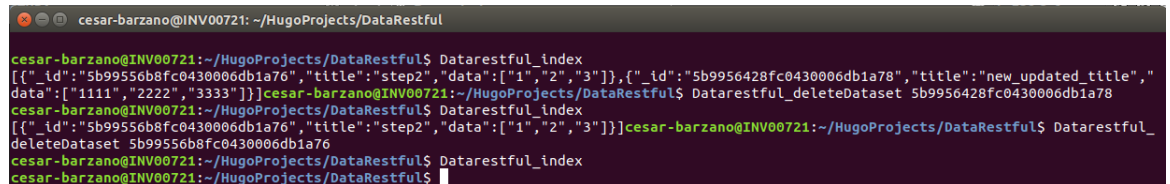


```

cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_getDataset 5b9956428fc0430006db1a78
{"_id":"5b9956428fc0430006db1a78","title":"step2_another_one","data":["1","2","3"]}cesar-barzano@INV00721:~/HugoProjects/DataRestful
$ Datarestful_updateDataset 5b9956428fc0430006db1a78 new_updated_title
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_getDataset 5b9956428fc0430006db1a78
{"_id":"5b9956428fc0430006db1a78","title":"new_updated_title","data":["1111","2222","3333"]}cesar-barzano@INV00721:~/HugoProjects/Da
taRestful$
```

Figura 3.16: Step 2: http://\$HOST:\$PORT/UpdateDataset

El usuario puede eliminar aquellos datasets que considere oportunos mediante la operación **Datarestful\_deleteDataset \$ID** donde \$ID corresponde al identificador del dataset en cuestión.



```

cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id":"5b9956428fc0430006db1a76","title":"step2","data":["1","2","3"]}, {"_id":"5b9956428fc0430006db1a78","title":"new_updated_title","data":["1111","2222","3333"]}cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_deleteDataset 5b9956428fc0430006db1a78
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id":"5b9956428fc0430006db1a76","title":"step2","data":["1","2","3"]}cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_deleteDataset 5b9956428fc0430006db1a76
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.17: Step 2: http://\$HOST:\$PORT/deleteDataset/\$ID

### 3.6.3. Step-3: Operaciones

El siguiente paso es solicitar alguna operación aritmética sobre un dataset, para ello, se puede hacer uso de la función **Datarestful\_operation \$ID \$VALUE \$OPE \$URL** donde:

1. **ID** identificador del dataset a operar
2. **VALUE** valor utilizado en la operacion aritmética
3. **OPE** Operador representativo de la operación
4. **URL** dirección del servicio al que delegaremos la operación

Para realizar una operación, el usuario ha de tener algún conjunto de datos declarado en el sistema, debe de haber algún servicio de procesamiento disponible y solicitar una operación, en el siguiente caso, la suma del valor 2 a todo el conjunto de datos:

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_addDataset step3
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id": "5b997eabbd93a4000733b4f8", "title": "step3", "data": ["1", "2", "3"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_listservices
[{"_id": "5b997e91bd93a4000733b4f7", "title": "basicPythonProcessor", "url": "\http://pythonprocessor:5000/basicOperator\"}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_operation 5b997eabbd93a4000733b4f8 2 + http://pythonprocessor:5000/basicOperator
{"title": "step3+_20180912_210213", "data": ["3.0", "4.0", "5.0"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id": "5b997eabbd93a4000733b4f8", "title": "step3", "data": ["1", "2", "3"]}, {"_id": "5b997ed5bd93a4000733b4f9", "title": "step3+_20180912_210213", "data": ["3.0", "4.0", "5.0"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

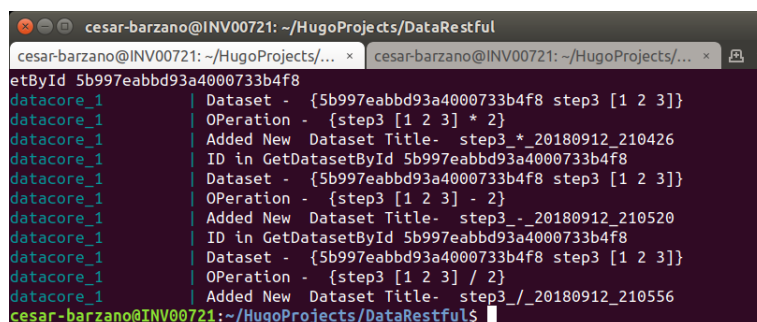
Figura 3.18: Step 3: `http://$HOST:$PORT/Operation/`

El procesador básico soporta multiplicación, resta y división:

```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_operation 5b997eabbd93a4000733b4f8 2 * http://pythonprocessor:5000/basicOperator
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id": "5b997eabbd93a4000733b4f8", "title": "step3", "data": ["1", "2", "3"]}, {"_id": "5b997ed5bd93a4000733b4f9", "title": "step3+_20180912_210213", "data": ["3.0", "4.0", "5.0"]}, {"_id": "5b997f5abd93a4000733b4fa", "title": "step3*_20180912_210426", "data": ["2.0", "4.0", "6.0"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_operation 5b997eabbd93a4000733b4f8 2 - http://pythonprocessor:5000/basicOperator
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ Datarestful_index
[{"_id": "5b997eabbd93a4000733b4f8", "title": "step3", "data": ["1", "2", "3"]}, {"_id": "5b997ed5bd93a4000733b4f9", "title": "step3+_20180912_210213", "data": ["3.0", "4.0", "5.0"]}, {"_id": "5b997f5abd93a4000733b4fa", "title": "step3*_20180912_210426", "data": ["2.0", "4.0", "6.0"]}, {"_id": "5b997f90bd93a4000733b4fb", "title": "step3-_20180912_210520", "data": ["-1.0", "0.0", "1.0"]}, {"_id": "5b997fb4bd93a4000733b4fc", "title": "step3/_20180912_210556", "data": ["0.5", "1.0", "1.5"]}]cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.19: Step 3: `http://$HOST:$PORT/Operation/`

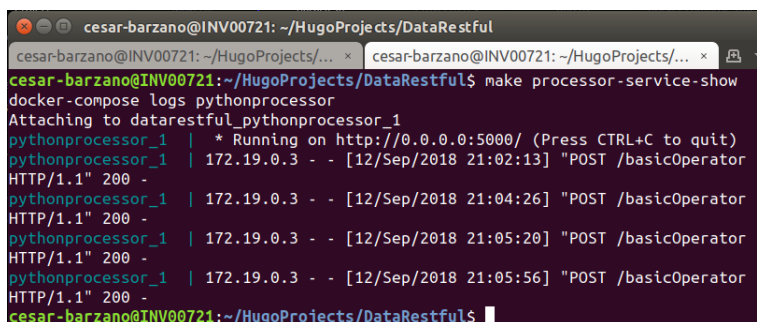
Chequeando los logs del servicio datacore se pueden observar las operaciones solicitadas:



```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721: ~/HugoProjects/... x cesar-barzano@INV00721: ~/HugoProjects/... x
etById 5b997eabbd93a4000733b4f8
datacore_1 | Dataset - {5b997eabbd93a4000733b4f8 step3 [1 2 3]}
datacore_1 | Operation - {step3 [1 2 3] * 2}
datacore_1 | Added New Dataset Title- step3 *_20180912_210426
datacore_1 | ID in GetDatasetById 5b997eabbd93a4000733b4f8
datacore_1 | Dataset - {5b997eabbd93a4000733b4f8 step3 [1 2 3]}
datacore_1 | Operation - {step3 [1 2 3] - 2}
datacore_1 | Added New Dataset Title- step3 _20180912_210520
datacore_1 | ID in GetDatasetById 5b997eabbd93a4000733b4f8
datacore_1 | Dataset - {5b997eabbd93a4000733b4f8 step3 [1 2 3]}
datacore_1 | Operation - {step3 [1 2 3] / 2}
datacore_1 | Added New Dataset Title- step3 _/20180912_210556
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.20: Step 3: make datacore-service-show

Chequeando los logs del servicio pythonprocessor se pueden observar las operaciones atendidas:



```
cesar-barzano@INV00721: ~/HugoProjects/DataRestful
cesar-barzano@INV00721: ~/HugoProjects/... x cesar-barzano@INV00721: ~/HugoProjects/... x
cesar-barzano@INV00721:~/HugoProjects/DataRestful$ make processor-service-show
docker-compose logs pythonprocessor
Attaching to datarestful_pythonprocessor_1
pythonprocessor_1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
pythonprocessor_1 | 172.19.0.3 - - [12/Sep/2018 21:02:13] "POST /basicOperator
HTTP/1.1" 200 -
pythonprocessor_1 | 172.19.0.3 - - [12/Sep/2018 21:04:26] "POST /basicOperator
HTTP/1.1" 200 -
pythonprocessor_1 | 172.19.0.3 - - [12/Sep/2018 21:05:20] "POST /basicOperator
HTTP/1.1" 200 -
pythonprocessor_1 | 172.19.0.3 - - [12/Sep/2018 21:05:56] "POST /basicOperator
HTTP/1.1" 200 -
cesar-barzano@INV00721:~/HugoProjects/DataRestful$
```

Figura 3.21: Step 3: make processor-service-show

### 3.7. Conclusiones

El prototipo del sistema resultante cumple con las indicaciones que cliente solicito ya que es facilmente escalable pues soporta tantas operaciones como servicios de procesamiento se registren en el servicio de gestión datacore. La arquitectura orientada a servicios que se ha implementado permite que distintos procesadores, implementados con distintas tecnologías sean capaces de integrarse como servicio al sistema lo que facilita la actualización de sus capacidades. La base de datos MongoDB esta pensada para grandes cantidades de datos y en el futuro, cuando el rendimiento del sistema se mida en proporciones globales podrá escalarse facilmente mediante una configuración de replica-set donde un conjunto de MongoDBs actuan como una sola.

El uso de contenedores para la estructuración y pruebas del sistema facilita a los desarrolladores la distribución del sistema ya que no necesitan instalar las dependencias, solo disponer de docker en su sistema, ademas facilita las pruebas y las tareas debug.

## Capítulo 4

# Entrega

La entrega contiene los siguientes directorios:

1. **basicPythonProcessor** código fuente del servicio de procesamiento.
2. **datacore:** código fuente del servicio principal de gestión de datos.
3. **DOC** Ficheros fuente de esta documentación redactada con LaTeX
4. **resources** Recursos adicionales como el cli.sh

De manera adicional, el directorio raíz contiene este documento en formato PDF a modo de memoria formal, licencia GPL3, herramienta de construcción y especificación de arquitectura.





# Bibliografía

- [1] GOOGLE, *Google Cloud* <https://cloud.google.com/?hl=es>
- [2] AWS, *Amazon Web Services* [https://aws.amazon.com/es/?nc2=h\\_lg](https://aws.amazon.com/es/?nc2=h_lg)
- [3] AZURE, *Microsoft Azure* <https://azure.microsoft.com/es-es/>
- [4] MLAB, *mLab: Database-as-a-Service for MongoDB* <https://mlab.com/>
- [5] YAHOO, *Yahoo* <https://es.yahoo.com/>
- [6] ECS, *Ether Cloud Services* <http://www.ticbeat.com/innovacion/fintech/inaki-bernal-cto-del-bbva-somos-una-compania-de-software-bastante-mala/>
- [7] API REST, *Representational State Transfer* <https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos>
- [8] HTTP, *Hypertext Transfer Protocol – HTTP* <https://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [9] PYTHON, *Programing Language* <https://www.python.org/>
- [10] FLASK, *Python Microframework* <http://flask.pocoo.org/>
- [11] GO, *The Go Programming Language* <https://golang.org/>
- [12] MONGODB, *noSQL Mongo Data Base* <https://www.mongodb.com/>
- [12] DOCKER, *Build, Manage and Secure Your Apps Anywhere. Your Way.* <https://www.docker.com/>