

## **Estructura base de un microservicio - Springboot**

---

**Estándares.**



**23 Enero/2018**

**Preparado por:**

**Luis Maridueña**

**Arquitectura y Soporte de TI**

**Banco Bolivariano**



 															
Estructura base de un microservicio - Springboot															
Área: Arquitectura IT															
Fecha: 14/12/2015															

© 2011 BANCO BOLIVARIANO  
 TODOS LOS DERECHOS RESERVADOS

Queda reservado el derecho de propiedad de este documento, con la facultad de disponer de él, publicarlo, traducirlo o autorizar su traducción, así como reproducirlo total o parcialmente, por cualquier sistema o medio.

No se permite la reproducción total o parcial de este documento, ni su incorporación a un sistema informático, ni su locación, ni su transmisión en cualquier forma o por cualquier medio, sea éste escrito o electrónico, mecánico, por fotocopia, por grabación u otros métodos, sin el permiso previo y escrito de los titulares de los derechos y del copyright.



FOTOCOPIAR ES CONSIDERADO UN DELITO.



															 <b>Banco Bolivariano</b> <small>El Banco con Visión</small>
Estructura base de un microservicio - Springboot															Área: Arquitectura IT
															Fecha: 14/12/2015

## Tabla de contenido

### Contenido

1	Objetivo .....	5
2	Generalidades .....	5
2.1	¿Qué es un micro servicio? .....	5
2.2	Ventajas de usar una arquitectura basada en microservicios .....	6
2.3	¿Qué es spring boot? .....	6
2.4	¿Que es maven? .....	7
3	Estructura base de microservicio .....	7
3.1	Estructura de directorios .....	8
3.2	Nomenclatura de paquetes .....	8
3.3	Archivos de configuración .....	9
3.4	Configuración de archivos de log .....	10
3.5	Banners .....	10
3.6	Construcción de binarios .....	11
3.7	Estructura del archivo POM .....	11

  <b>Banco Bolivariano</b> <small>El Banco con Visión</small>															
Estructura base de un microservicio - Springboot															
Área: Arquitectura IT															
Fecha: 14/12/2015															

																
Estructura base de un microservicio - Springboot															Área: Arquitectura IT	
															Fecha: 14/12/2015	

## 1 Objetivo

El siguiente documento tiene el objetivo de servir como referencia al momento de desarrollar un micro servicio usando maven y springboot, ya que es el estándar usado actualmente en el banco.

## 2 Generalidades

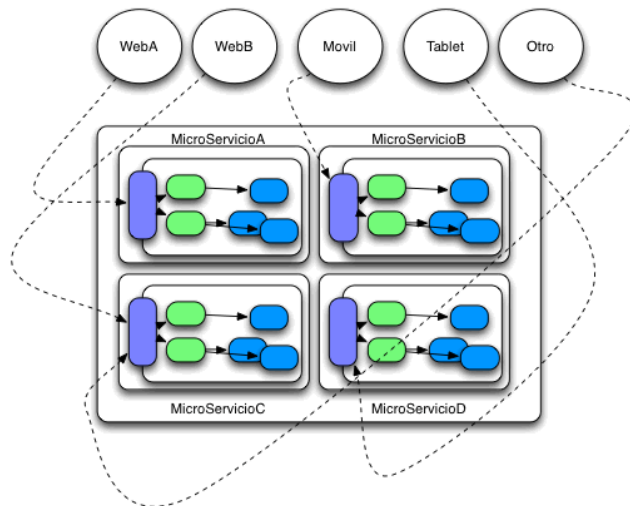
### 2.1 ¿Qué es un micro servicio?



Una “arquitectura de micro servicios” es un enfoque para desarrollar una aplicación software como una serie de pequeños servicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí, por ejemplo, a través de peticiones HTTP a sus API.

Normalmente hay un número mínimo de servicios que gestionan cosas comunes para los demás (como el acceso a base de datos), pero cada micro servicio es pequeño y corresponde a un área de negocio de la aplicación.

Además cada uno es independiente y su código debe poder ser desplegado sin afectar a los demás. Incluso cada uno de ellos puede escribirse en un lenguaje de programación diferente, ya que solo exponen la API (una interfaz común, a la que le da igual el lenguaje de programación en la que el micro servicio esté programado por debajo) al resto de micro servicios.

No hay reglas sobre qué tamaño tiene que tener cada micro servicio, ni sobre cómo dividir la aplicación en micro servicios, pero algunos autores como Jon Eaves caracterizan un micro servicio como algo que a nivel de código podría ser reescrito en dos semanas.



																
Estructura base de un microservicio - Springboot										Área: Arquitectura IT						
										Fecha: 14/12/2015						

## 2.2 Ventajas de usar una arquitectura basada en microservicios

- **Combinar los servicios como nos interesen.** Incluso, reutilizarlos para distintos usos dentro de la empresa. Como piezas de un rompecabezas podemos crear aplicaciones que usen una misma lógica de negocio sin estar cautiva en una aplicación monolítica.
- **Escalar a nivel de micro servicios.** Cada uno de ellos expone una funcionalidad, así podemos distribuirlo según nuestras necesidades pensando en la demanda y el balanceo de carga de cada aplicación. Esto contraponen la idea poco eficiente de replicar aplicaciones enteras cargadas de funcionalidad por unas pocas que represente la mayor carga.
- **Simplificamos el mantenimiento.** Cumplen a la perfección el principio SOLID. Podemos desechar componentes que no reutilizando o extenderlos en otros para engancharlo en nuestra aplicación. Es más fácil tirar algo que no usemos a la basura que mantenerlo como código legacy dentro de una aplicación monolítica.
- **Su fallo no arrastra a todo el sistema.** Si un componente no funciona correctamente no afecta al resto. Se puede aislar y manejar el error, desplegando por separado.
- **El despliegue puede ser progresivo sin parar todo de golpe.** Empezamos a ver más luz a la integración continúa alcanzando mayores cifras de disponibilidad. Aunque recuerda que esto implica que ya no tienes un único contenedor, sino que hay que estar pendiente de  $n$  contenedores de aplicaciones.

















**El camino hacia los microservicios no es trivial.** Al igual que mencionamos que todo trabaja en conjunto, llegar ahí necesita portar el código legacy de la aplicación monolítica y parcelarlo. No te vuelvas loco en convertir en microservicios tu actual aplicación. Comienza **poco a poco explorando este tipo de arquitectura con las nuevas funcionalidades/servicios** que crees.

Para entender más en detalle lo qué son los microservicios os recomiendo a un clásico como Martin Fowler. Él aporta todo ese halo filosófico y teórico, sin importar el lenguaje o la plataforma, en su artículo sobre la arquitectura de Microservicios.

## 2.3 ¿Qué es spring boot?

Tradicionalmente las aplicaciones Java web han sido instaladas en un contenedor de *servlets* como Tomcat o Jetty y Wildfly, JBoss o Weblogic si necesita más servicios que son ofrecidos por la plataforma Java EE completa como JMS, JPA, JTA o EJB. Aunque las aplicaciones se ejecutan independientemente unas de otras comparten el entorno de ejecución del servidor de aplicaciones, algunas aplicaciones no necesitarán todos los servicios que ofrecen los servidores de aplicaciones en su implementación del perfil completo Java EE y algunas nuevas aplicaciones pueden necesitar hacer uso de una nueva versión de un servicio como JMS con funcionalidades mejoradas. En el primer caso algunos servicios son innecesarios y en el segundo la actualización del servidor de aplicaciones se ha de producir para todas las aplicaciones que en él se ejecuten o tener varias versiones del mismo servidor de aplicaciones e ir instalando las aplicaciones en la versión del servidor según las versiones de los servicios para las que se desarrolló la aplicación.

Los microservicios proponen una aproximación diferente al despliegue de las aplicaciones prefiriendo entre otros aspectos que sean autocontenidos de tal forma que puedan evolucionar independientemente unas de otras. Se puede ejecutar una aplicación web Java de forma autocontenida con la versión embebida de Tomcat, Jetty también ofrece una versión embebible que puede usarse de forma similar de tal modo que ya no necesitemos instalar previamente además del JDK la versión del servidor de aplicaciones que necesite.

             															 		<b>Banco Bolivariano</b> <i>El Banco con Visión</i>	
Estructura base de un microservicio - Springboot															Área: Arquitectura IT			
															Fecha: 14/12/2015			

Otra forma de poder hacer la aplicación autocontenida es con Spring Boot, internamente usa una versión embebible del servidor de aplicaciones de la misma forma que lo podemos usar directamente, la ventaja al usar Spring Boot es que soporta Tomcat, Jetty o Undertow y pasar de usar uno a otro es muy sencillo y prácticamente transparente para la aplicación, además proporciona algunas características adicionales como inicializar el contenedor IoC de Spring, configuración, perfiles para diferentes entornos (desarrollo, pruebas, producción), monitorización y métricas del servidor de aplicaciones y soporte para la herramienta de automatización Gradle entre algunas más. En el siguiente ejemplo mostraré como ejecutar una aplicación Java y una aplicación web Java con Spring Boot que usa jOOQ como alternativa a Hibernate, Apache Tapestry como *framework* web, Liquibase para crear el esquema y tablas de la base de datos y por simplicidad H2 como base de datos.

## 2.4 ¿Que es maven?

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant (y en menor medida a PEAR de PHP y CPAN de Perl), pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.



Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta. Maven provee soporte no solo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje. En realidad, el soporte y uso de lenguajes distintos de Java es mínimo.

## 3 Estructura base de microservicio

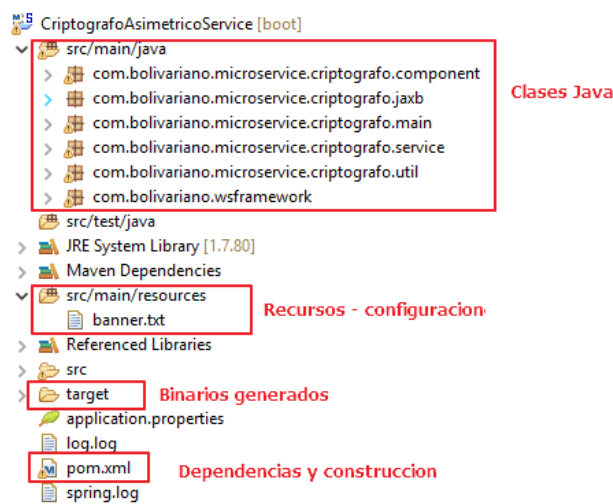
En esta sección se mostrarán las consideraciones básicas que todo micro servicio debe tener al momento de ser construido tales como:

- Estructura de directorios
- Nomenclatura de paquetes
- Archivos de configuración
- Configuración de archivos de log
- Banners
- Construcción de binarios
- Estructura del archivo POM

										
Estructura base de un microservicio - Springboot										Área: Arquitectura IT
										Fecha: 14/12/2015

### 3.1 Estructura de directorios

Como parte de la estructura del microservicio springboot hemos tratado de seguir la nomenclatura básica de un proyecto generado con maven, a continuación, detallamos cada uno de los directorios más importantes:



Nota: El IDE estándar para el desarrollo java usado en el banco es eclipse, por tal motivo la imagen muestra la estructura de directorios de un proyecto en dicho IDE.

**Clases java.** - En esta ruta debemos colocar todas las clases java que necesitemos crear para nuestro micro servicio, como se muestra en la imagen la estructura de paquetes tiene una nomenclatura estándar que será explicado más adelante.

**Recursos – configuración.** - En esta ruta se encontrarán los recursos spring que se empaquetarán con el micro servicio, tales como banner y cualquier otro archivo de configuración que no necesite ser modificado.

**Binarios generados.** - En esta ruta encontraremos el jar resultante de la construcción de nuestro proyecto usando maven.

**Dependencias y construcción.** - Este es el archivo de mayor importancia para la construcción de nuestro proyecto ya que en el agregamos muchos detalles necesarios como dependencias, exclusiones, detalles para generación de binarios, despliegues automáticos, etc.

### 3.2 Nomenclatura de paquetes



Como estándar de nombramiento de paquetes en aplicaciones java dentro del banco se definió la siguiente nomenclatura

**com.bolivariano.[tipo de servicio].[nombre abreviado de servicio].\***

El tipo de servicio indica si es un web service, microservicio, tcp, archivos, etc por ejemplo:

com.bolivariano.microservice.criptografo.component



										
Estructura base de un microservicio - Springboot										Área: Arquitectura IT
										Fecha: 14/12/2015

El nombre abreviado del servicio indica un nombre corto con el que se pueda asociar el paquete a que servicio pertenece, por ejemplo:

com.bolivariano.microservice.criptografo.component

En estos ejemplos se puede apreciar que el paquete pertenece al microservicio del componente llamado "Criptografo Asimetrico"

Consideraciones generales

- El nombre de los paquetes debe estar en minúsculas
- Se recomienda tener un máximo de 6 niveles de divisiones en el nombre
- El nombre del servicio debe estar en minúsculas
- Dentro de la estructura interna del servicio se deben seguir los siguientes lineamientos para agrupar las clases según su uso:
  - **bean:** En este paquete se deben colocar todas las clases usadas como POJO
  - **configuration:** En este paquete se deben colocar las clases relacionadas a la configuración de Spring.
  - **dao:** En este paquete se deben colocar las clases relacionadas al acceso a base de datos.
  - **service:** En este paquete se deben colocar las clases de relacionadas con los servicios spring encargados del acceso a las fuentes de información.
  - **cx:** En este paquete se deben colocar las clases relacionadas a la configuración de apache cxf
  - **controller:** En este paquete se deben colocar todas las clases REST controller.
  - **utils:** En este paquete se deben colocar todas las clases de naturalezas utilitarias









**Comentado [LMN1]:** 1er ítem de la lista de cambios solicitados

**Comentado [LMN2]:** 2do ítem de la lista de cambios solicitados

**Comentado [LMN3]:** 8avo ítem de la lista de cambios

Ejemplo:

```

v  usuarioms
  v  src/main/java
    >  com.bolivariano.microservice.usuarioms
    >  com.bolivariano.microservice.usuarioms.configuration
    >  com.bolivariano.microservice.usuarioms.cxf
    >  com.bolivariano.microservice.usuarioms.dao
    >  com.bolivariano.microservice.usuarioms.service
    >  com.bolivariano.microservice.usuarioms.utils
  
```

### 3.3 Archivos de configuración



Para que nuestro micro servicio funcione adecuadamente es necesario realizar ciertas configuraciones las cuales se replican en todos los componentes desarrollados, esta configuración debe estar fuera del binario generado para que pueda ser modificada sin necesidad de recompilar y generar ejecutable. El archivo común se llama **application.properties**, este archivo tiene una estructura similar a un archivo de propiedades normalmente usado por las aplicaciones, a continuación se detalla las configuraciones que todo servicio debe tener.

**#configuracion de acceso**

**server.port** - Puerto tcp que usa el micro servicio para atender requerimientos.

**server.tomcat.max-threads** - Cantidad de hilos disponibles para atender requerimientos.

**cx.path** - Contexto base usado para exponer el servicio web, esto se concatena a los endpoints (esta propiedad aplica para servicios SOAP)

										
Estructura base de un microservicio - Springboot										Área: Arquitectura IT
										Fecha: 14/12/2015

#### #informacion del servicio

**info.version.-** Versión del microservicio.

**info.date.-** Fecha de liberación de la versión.

**info.stage.-** Ambiente en el que se ejecuta el servicio.

#### #configuracion de logs

**logging.config.-** Ruta del archivo de configuración para logeo del servicio.

#### #configuracion de monitoreo

**spring.boot.admin.url.** - URL del servidor de monitoreo para el reporte de estado.

**spring.application.name.** - Nombre del servicio que se mostrara en aplicación de monitoreo

Se debe tomar en cuenta que este archivo puede contener configuraciones personalizadas o propias de spring, dentro del programa se puede acceder a cualquiera de ellas por medio de su nombre.

### 3.4 Configuración de archivos de log

La configuración del logeo se encuentra en un archivo independiente el cual se indica en la propiedad **loggin.config**, tal como lo vimos en el punto anterior. Dentro de este archivo vamos a colocar la configuración de nuestro archivo de log, a continuación, se muestra un archivo de configuración ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="SAVE-TO-FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>./logs/entrust.log</file>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <Pattern>%d{dd-MM-yyyy HH:mm:ss} [%thread] %-5level %logger{36}.%M - %msg%n</Pattern>
    </encoder>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>./logs/entrust.log.%d{dd-MM-yyyy}</fileNamePattern>
      <maxHistory>100</maxHistory>
      <totalSizeCap>300MB</totalSizeCap>
    </rollingPolicy>
  </appender>
  <root level="info">
    <appender-ref ref="SAVE-TO-FILE" />
  </root>
</configuration>
```

En los recuadros de la imagen vemos las partes específicas que se deben modificar por cada micro servicio:



**<file>** En este dato debemos colocar la ruta donde se crearán nuestros archivos de logs, en el ejemplo se configuró para que se guarden en una carpeta llamada logs dentro del mismo directorio.

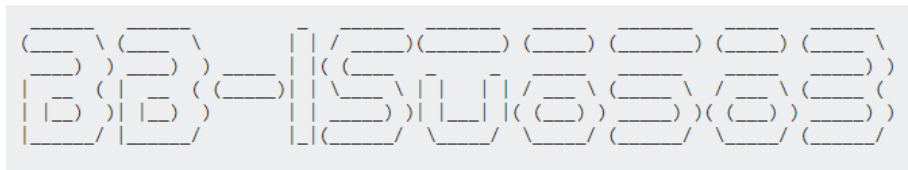
**<fileNamePattern>** En este dato debemos colocar el nombre de los archivos históricos que se crearán dentro de la ruta configurada, el patrón estándar es **ruta\_logs/nombre\_log.log.día-mes-año**

**<totalSizeCap>** En este dato debemos colocar el tamaño máximo del archivo de log, este tamaño dependerá de la cantidad de información que registre el servicio, para el ejemplo se usa un valor de 300MB lo cual indica que el archivo de log crecerá máximo ese límite y luego creará un nuevo archivo con un secuencial seguido del formato de fecha configurado.

### 3.5 Banners

Como parte de una mejora visual se define como un estándar que todos los servicios usen un banner para que se muestre al momento de iniciar. A continuación, observemos un ejemplo de banner:

															 <b>Banco Bolivariano</b> <i>El Banco con Visión</i>			
Estructura base de un microservicio - Springboot										Área: Arquitectura IT								
										Fecha: 14/12/2015								

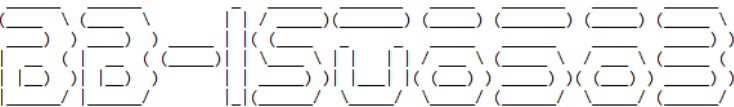


Para generar el texto del banner es necesario acceder a la página donde se genera el texto:  
<https://devops.datenkollektiv.de/banner.txt/index.html> , en la página debe colocar el nombre abreviado del servicio y usar la fuente rounded. Tomar en cuenta que el nombre a colocar en el cuadro de texto debe contener el prefijo "BB".  
 Una vez generado el texto se debe colocarlo dentro del archivo banner.txt tal como muestra la imagen a continuación.

```

1 ${Ansi.GREEN}
2 ${Ansi.GREEN}
3 ${Ansi.GREEN}
4 ${Ansi.GREEN}
5 ${Ansi.GREEN}
6 ${Ansi.GREEN}
7 ${Ansi.GREEN}=====

```



### 3.6 Construcción de binarios

Para la generación de los ejecutables finales haremos uso de maven, con esta herramienta podemos hacer una construcción automática registrando las dependencias y configuraciones específicas.  
 A continuación, se muestra sección del archivo pom.xml para configurar la construcción del archivo

```

















<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
  <finalName>TransformadorISOMicroservice-${version}</finalName>
</build>

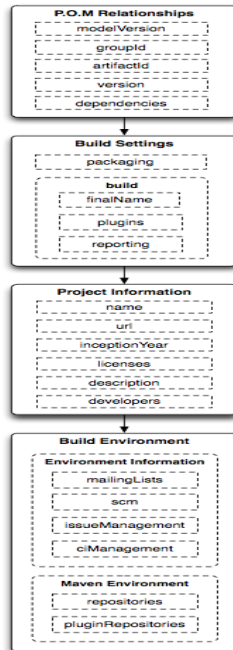
```

En este caso se configura el plugin de generación de springboot para maven, el cual genera un jar con todas las dependencias y librerías necesarias para que se ejecute el servicio de manera independiente.

### 3.7 Estructura del archivo POM

Sus siglas en ingles significan Project Object Model, en este xml de configuración se configuran todas las dependencias, atributos y detalles de la construcción automática, a continuación, describiremos un poco las secciones que usaremos para nuestros micro servicios, tomar en cuenta que el archivo tiene muchas más opciones de configuración, pero las que documentaremos aquí serán las que debe usar como base un micro servicio desarrollado en Banco Bolivariano.

              															 <b>Banco Bolivariano</b> <small>El Banco con Visión</small>	
Estructura base de un microservicio - Springboot															Área: Arquitectura IT	
															Fecha: 14/12/2015	



Para los servicios implementados en Banco Bolivariano debemos cumplir con los siguientes lineamientos para poder agruparlos y nombrarlos de una mejor manera.

- El groupId definido en el archivo pom debe llamarse **com.bolivariano.microservice**.
- El artifactId definido en el archivo pom debe ser el nombre del proyecto en minúsculas.
- La versión inicial del microservicio debe empezar en **1.0.0** y debe tener 3 niveles de versionamiento.
- Si el nombre del servicio posee más de 1 palabra se debe usar el carácter "-" como separador, por ejemplo: **consultas-cuentas**

**Comentado [LMN4]:** 3er ítem de la lista de cambios

**Comentado [LMN5]:** 4to ítem de la lista de cambios

**Comentado [LMN6]:** 5to ítem de la lista de cambios



Ejemplo:

Artifact	
Group Id:	com.bolivariano.microservice
Artifact Id:	* usuarioms
Version:	1.0.0
Packaging:	jar

## 4 Monitoreo

En esta sección se agregan datos necesarios para poder monitorear los microservicios por medio de springboot admin.

Para configurar el monitoreo en cada microservicio se deben seguir los siguientes pasos:

																								
Estructura base de un microservicio - Springboot															Área: Arquitectura IT									
															Fecha: 14/12/2015									


1. Agregar las siguientes dependencias al pom.xml del proyecto

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-starter-client</artifactId>
  <version>1.5.4</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

2. Luego de agregar las dependencias debemos agregar las siguientes propiedades al archivo **application.properties** del proyecto.

```
spring.boot.admin.url=http://172.16.30.152:8080
spring.boot.admin.client.enabled=true
spring.boot.admin.connect-timeout=2000
spring.boot.admin.read-timeout=10000
spring.application.name=Avisos24MS
```

En la primera línea se indica la url donde se encuentra disponible el spring-boot admin y en la segunda línea hace referencia al nombre del microservicio. Una vez que el microservicio se conecte al administrador, este podrá ser monitoreado desde la consola, tal como lo muestra la siguiente imagen.

spring 		APPLICATIONS	JOURNAL	ABOUT
Spring Boot applications				
Filter				
Application	URL	Version	Info	Status
Avisos24MS (ae947104)	http://esbsswsd1.bolivariano.fin.ec:8090/			UP
Avisos24MS (70163cfe)	http://CSPCOMMDSE2.bolivariano.fin.ec:8090/			UP
CriptografoAsimetricoMS (13ea8b47)	http://CSPCOMMDSE2.bolivariano.fin.ec:8096/			UP

## CONTROL DE CAMBIOS DEL DOCUMENTO

VERSIÓN	DESCRIPCIÓN DEL CAMBIO	FECHA DEL CAMBIO	ELABORADO POR	APROBADO POR
1.0.0	Versión inicial	23 de enero 2018	Luis Maridueña	
1.0.1	Agregar datos de monitoreo y estructura interna de paquetes	4 de junio del 2018	Luis Maridueña	