# 

**Documento Arquetipos y Guías de Implementación**

**Elaborado para Coordinador Eléctrico Nacional**



Diciembre 2021

Tabla de Contenido

[**Prefacio**](#_1z6dsh353lkn) **3**

[Confidencialidad, derechos de autor y extensión de responsabilidad](#_a5yrox6sx5iu) 3

[Acerca de este documento](#_xy1y0go8efu9) 3

[Contactos del equipo de Red Hat para este proyecto](#_4qhibqapl10e) 3

[**Introducción**](#_tdvh0d4cw1hv) **4**

[Terminologías](#_ppkxh3aq94em) 4

[**Condiciones Técnicas**](#_nb41fdethl0e) **4**

[Estereotipos](#_8wl4de3th9mn) 5

[Lógica de utilización](#_do684nhq1al4) 5

[Proyección de Arquitectura CEN](#_umtif87e6oko) 6

[**Condiciones de construcción e implementación**](#_d1on0a903tsm) **7**

[Precondiciones](#_7rgkkfj8va91) 7

[Generación del Arquetipo](#_u0srwzj6twzm) 7

[**Arquetipos**](#_6p29zyg5gfii) **11**

[Arquetipo Proxy SOAP](#_x4ojcwvn9zv0) 11

[Ruta Camel](#_71xv3vlkpa7s) 13

[Utilización](#_s4icio458dt0) 14

[Arquetipo Proxy REST](#_z3eqbvyy7f9) 18

[Ruta Camel](#_qkaoylte74b) 20

[Utilización](#_no7gczo0yrv2) 21

[Registro del Arquetipo en el repositorio Nexus](#_mxg941ry6idt) 25

[Generación de Proyecto con Servicio desde Arquetipo en repositorio Nexus](#_cmveesek32yk) 28

[Incorporación de Jaeger](#_oodfuput521k) 28

[**Arquitectura de APIs**](#_3v24561lg0z) **29**

[Normar Base de Arquitectura](#_m82lr95zy5hw) 31

# Prefacio

## Confidencialidad, derechos de autor y extensión de responsabilidad

Se trata de un documento orientado al cliente entre Red Hat, Inc. y Coordinador Eléctrico Nacional

Copyright © 2021 Red Hat, Inc. Todos los derechos reservados. Ninguna parte del trabajo cubierto por los derechos de autor puede ser reproducida o utilizada de ninguna forma ni por ningún medio - gráfico, electrónico o mecánico, incluyendo fotocopiado, grabación o sistemas de almacenamiento y recuperación de información - sin el permiso por escrito de Red Hat, excepto cuando sea necesario para compartir esta información, tal y como se ha proporcionado a las partes confidenciales mencionadas anteriormente.

## Acerca de este documento

Este documento es el resultado de sesiones entre Red Hat y el Coordinador Eléctrico Nacional, donde se exponen las condiciones del proyecto desde el punto de vista de la migración y modernización de servicios. Este documento está dirigido para quienes deban comprender el contexto de este proyecto y que además les permita contar con una guía de migración y de modernización de servicios.

La versión de este documento referencia el alcance del proyecto el cual fue definido por los equipos de trabajo, tanto Red Hat como CEN(Coordinador Eléctrico Nacional) y de este modo generar una guía de trabajo para los equipos de consultoría.

## Contactos del equipo de Red Hat para este proyecto

Para obtener información adicional sobre la contenida en este documento, puede contactar con uno de los siguientes miembros del equipo de Red Hat:

| **Name** | **Title** | **Number** | **Email** |
| --- | --- | --- | --- |
| Cristian Quezada | Senior Consultant | +56 9 6340 9153 | cquezada@redhat.com |
| Hugo Cerda | Consultant | +56 9 5321 6795 | hcerdaja@redhat.com |
| Nathalia Patiño | Engagement Lead | +56 9 7518 1603 | npatinom@redhat.com |
| Daniele Stefenello | Project Manager | +56 9 7705 8527 | dstefene@redhat.com |

# 

# Introducción

Este documento busca dar a conocer lineamientos de arquitectura para el desarrollo y generación de servicios web basados en condiciones de la arquitectura middleware Red Hat OpenShift, esto permitirá resolver servicios en mayor medida dado los estándares establecidos como además la agilidad sobre la construcción del mismo.

## Terminologías

| **Término** | **Definición** |
| --- | --- |
| MVN | Comando de ejecución maven para la utilización de proyectos maven como arquetipos |
| SOAP | Simple Object Access Protocol, protocolo de servicio web basado en definición WSDL |
| REST | Representational State Transfer, servicio web basado en protocolo HTTP con definición sencilla de comunicación |
| JSON | JavaScript Object Notation, formato de texto ligero para la utilización de servicios REST principalmente. |

*Tabla 1: Terminologías*

# Condiciones Técnicas

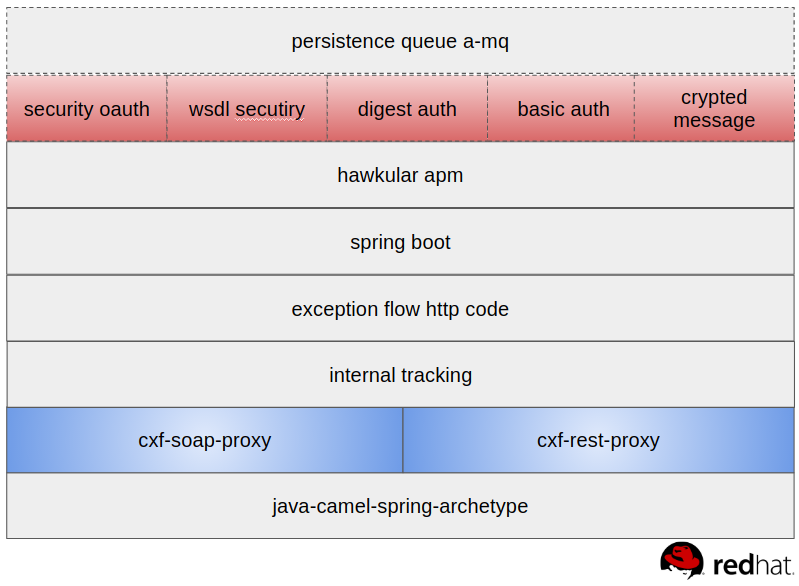
El proceso de arquetipos conlleva a la realización de proyectos bases que permiten reconocer una estructura predefinida. Como se puede ver en la siguiente imagen, el proceso comienza con la generación de una estructura de proyecto base donde se define los lineamientos de construcción la cual es posteriormente copiado a un proyecto de tipo arquetipo que será utilizado para generación de múltiples proyectos.

Cada estructura maven deberá generar un componente .jar que deberá ser “instalado” sobre algún repositorio corporativo maven, para este caso se entiende que El Coordinador dispone de un repositorio Nexus en el cual estos componentes serán desplegados.

## Estereotipos

Se ha incorporado hasta el momento una serie de estereotipos que permitirán a El Coordinador establecer lineamientos generales para la construcción de servicios web sobre la estructura middleware Red Hat OpenShift. Esto teniendo en cuenta las situaciones críticas de construcción de servicios los cuales quedan resueltos por:

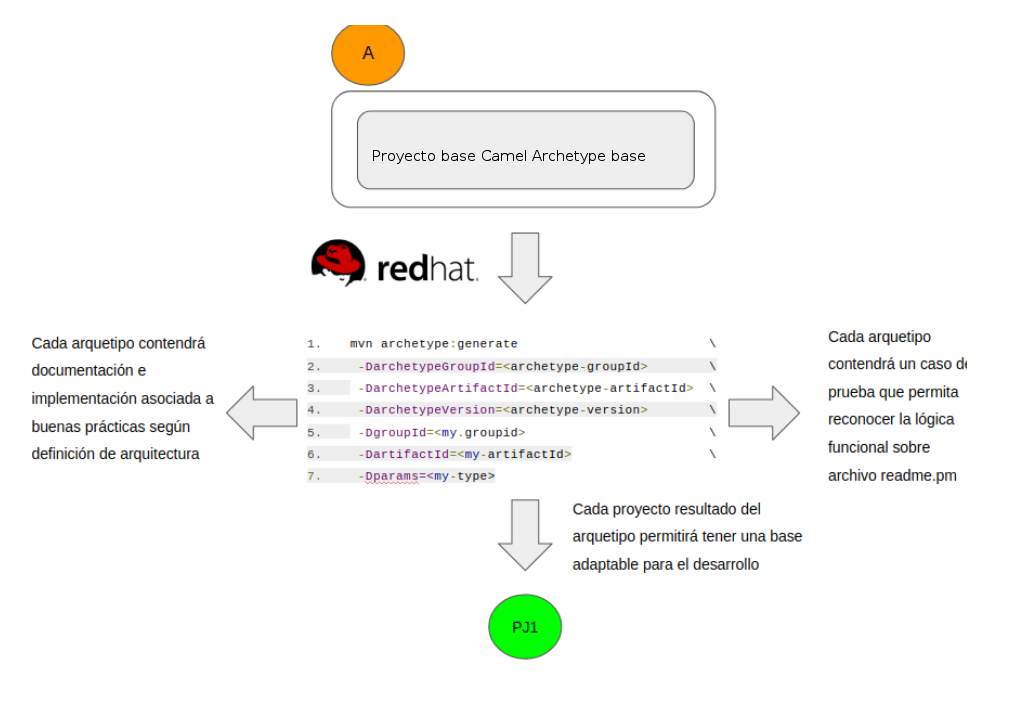
* Arquetipo Proxy Soap: Arquetipos para la construcción de servicios SOAP el cual contendrá las condiciones base de proyecto
* Arquetipo Proxy Rest: Arquetipo para la generación de arquetipos REST



## Lógica de utilización

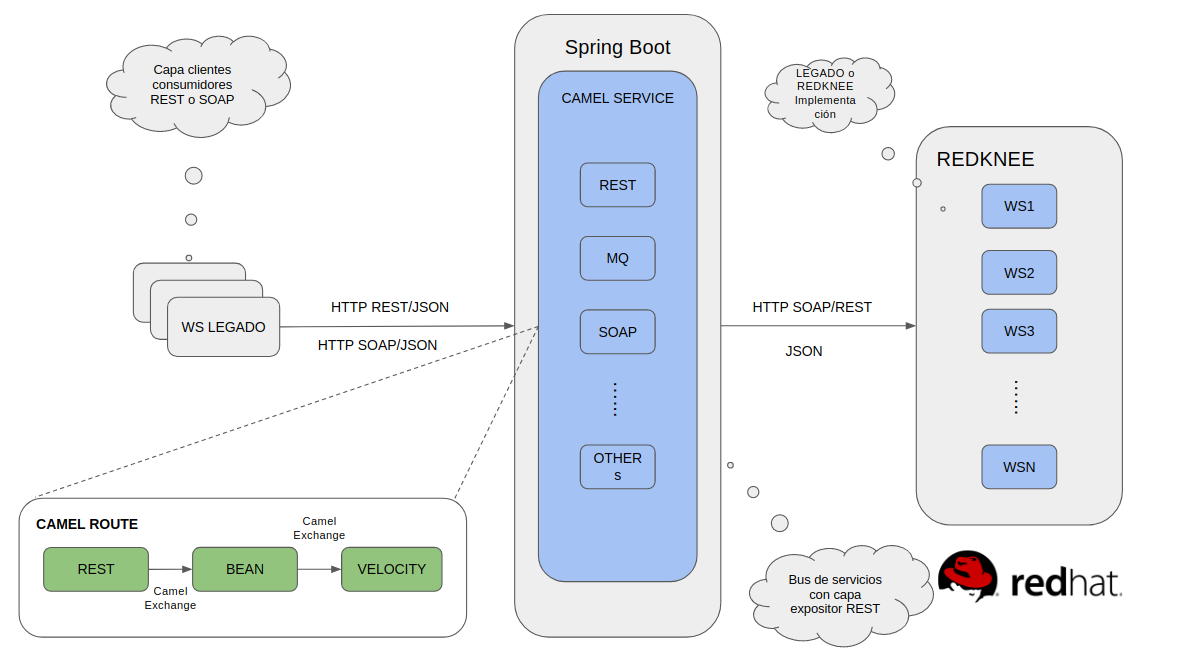
La lógica de utilización de arquetipos tiene como objetivo resolver un proyecto tipo sobre condiciones previamente definidas sobre estructura de arquetipo, esto permite definir en base a un comando la utilización del arquetipo que finalizará en un proyecto tipo según parametrización de entrada establecida.

A continuación la lógica de utilización del arquetipo:



## Proyección de Arquitectura CEN

La principal razón para la utilización de arquetipos está dada por la definición de arquitectura que El Coordinador establece para sus soluciones de integración, con esto se puede entender que la lógica de integración estará dada por un bus conceptual de servicios basado en Spring Boot y que definirá una suerte de expositores y consumidores, esto se refiere a Web Services Legado que consumen servicios RedKnee a través del bus de servicio o arquitectura de Servicios Red Hat OpenShift.

En la siguiente imagen se ve una representación general de arquitectura donde se establecen los actores principales.

# Condiciones de construcción e implementación

Los arquetipos han sido construidos en base a la definición declarada previamente, con lo cual a continuación se comenzará a detallar las condiciones de construcción e implementación.

## Precondiciones

Se deberá tener presente que las condiciones técnicas para la construcción de arquetipos quedarán dadas por una versión java estándar y maven, se deberá tener un repositorio maven local habilitado para este propósito.

| Apache Maven 3.3.9  Maven home: /usr/share/maven  Java version: 1.8.0\_131, vendor: Oracle Corporation  Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre  Default locale: en\_US, platform encoding: UTF-8  OS name: "linux", version: "4.10.0-32-generic", arch: "amd64", family: "unix" |
| --- |

## Generación del Arquetipo

Se debe comprender que para poder generar un arquetipo, lo primero a realizar es tomar un proyecto maven del tipo arquetipo para su construcción, o sea, la construcción de un proyecto arquetipo de tipo arquetipo maven.

| mvn archetype:generate \  -DgroupId=[your project's group id] \  -DartifactId=[your project's artifact id] \  -DarchetypeArtifactId=maven-archetype-archetype |
| --- |

A continuación se indica cómo se genera un arquetipo base:

| #ARQUETIPO SPRING-BOOT REST  mvn archetype:generate -DarchetypeGroupId=cl.coordinador.arquetipos \  -DarchetypeArtifactId=archetype-rest-microservice-crud \  -DarchetypeVersion=1.0.0 \  -DgroupId=cl.coordinador.personas \  -DartifactId=api-personas-rrhh-personas |
| --- |

| #ARQUETIPO SPRING-BOOT REST APACHE CAMEL  mvn archetype:generate \  -DarchetypeGroupId=cl.coordinador.arquetipos \  -DarchetypeArtifactId=archetype-spring-boot-proxy-rest-soap \  -DarchetypeVersion=1.0.0 \  -DgroupId=cl.coordinador.gerencia.departamento \  -DartifactId=coordinador-proxy-soap-spring-boot-camel |
| --- |

Lo anterior permite generar un proyecto maven de tipo arquetipo basada en un arquetipo. Se puede visualizar que el proyecto de tipo arquetipo contendrá principalmente las condiciones de groupID, artifactId y versión, con lo anterior tendremos un proyecto con las siguiente estructura.

| [INFO] Generating project in Interactive mode  [WARNING] Archetype not found in any catalog. Falling back to central repository.  [WARNING] Add a repository with id 'archetype' in your settings.xml if archetype's repository is elsewhere.  [INFO] Using property: groupId = cl.coordinador.personas  [INFO] Using property: artifactId = api\_personas\_rrhh\_personas  Define value for property 'version' 1.0-SNAPSHOT: : [INFO] Using property: package = cl.coordinador.personas  Confirm properties configuration:  groupId: cl.coordinador.personas  artifactId: api\_personas\_rrhh\_personas  version: 1.0-SNAPSHOT  package: cl.coordinador.personas  Y: :  [INFO] ----------------------------------------------------------------------------  [INFO] Using following parameters for creating project from Archetype: archetype-rest-microservice-crud:1.0.0  [INFO] ----------------------------------------------------------------------------  [INFO] Parameter: groupId, Value: cl.coordinador.personas  [INFO] Parameter: artifactId, Value: api\_personas\_rrhh\_personas  [INFO] Parameter: version, Value: 1.0-SNAPSHOT  [INFO] Parameter: package, Value: cl.coordinador.personas  [INFO] Parameter: packageInPathFormat, Value: cl/coordinador/personas  [INFO] Parameter: package, Value: cl.coordinador.personas  [INFO] Parameter: groupId, Value: cl.coordinador.personas  [INFO] Parameter: artifactId, Value: api\_personas\_rrhh\_personas  [INFO] Parameter: version, Value: 1.0-SNAPSHOT  [WARNING] Don't override file /home/hai/Desktop/arquetipos-coordinador/tmp-rest-crud/api\_personas\_rrhh\_personas/src/main/java  [INFO] Project created from Archetype in dir: /home/hai/Desktop/arquetipos-coordinador/tmp-rest-crud/api\_personas\_rrhh\_personas  [INFO] ------------------------------------------------------------------------  [INFO] BUILD SUCCESS  [INFO] ------------------------------------------------------------------------  [INFO] Total time: 46.018 s  [INFO] Finished at: 2021-12-28T13:42:32-03:00  [INFO] ------------------------------------------------------------------------ |
| --- |

Se ha procesado el arquetipo el cual existirán las condiciones para comenzar a ensamblar un nuevo arquetipo, la estructura que ha sido generada se declara a continuación:

| #~/git/coordinador/tmp$ tree middleware-soap-proxy-archetype/  └── api\_personas\_rrhh\_personas  ├── pom.xml  ├── README.md  ├── src  │ └── main  │ ├── java  │ │ └── cl  │ │ └── coordinador  │ │ └── gerencia  │ │ └── departamento  │ │ ├── controllers  │ │ │ └── PersonaController.java  │ │ ├── CreateServiceApplication.java  │ │ ├── exceptions  │ │ │ ├── ApiExceptionHandler.java  │ │ │ ├── BadRequestException.java  │ │ │ ├── ErrorMessage.java  │ │ │ ├── InternalException.java  │ │ │ └── NotFoundException.java  │ │ └── models  │ │ ├── dao  │ │ │ └── PersonaDao.java  │ │ ├── entity  │ │ │ └── Persona.java  │ │ └── services  │ │ ├── IPersonaService.java  │ │ └── PersonaServiceImp.java  │ └── resources  │ ├── application.properties  │ ├── META-INF  │ │ └── additional-spring-configuration-metadata.json  │ ├── routes  │ ├── velocity  │ └── wsdl  └── target  ├── api\_personas\_rrhh\_personas-1.0-SNAPSHOT.jar  ├── api\_personas\_rrhh\_personas-1.0-SNAPSHOT.jar.original  ├── classes  │ ├── application.properties  │ ├── cl  │ │ └── coordinador  │ │ └── gerencia  │ │ └── departamento  │ │ ├── controllers  │ │ │ └── PersonaController.class  │ │ ├── CreateServiceApplication.class  │ │ ├── exceptions  │ │ │ ├── ApiExceptionHandler.class  │ │ │ ├── BadRequestException.class  │ │ │ ├── ErrorMessage.class  │ │ │ ├── InternalException.class  │ │ │ └── NotFoundException.class  │ │ └── models  │ │ ├── dao  │ │ │ └── PersonaDao.class  │ │ ├── entity  │ │ │ └── Persona.class  │ │ └── services  │ │ ├── IPersonaService.class  │ │ └── PersonaServiceImp.class  │ └── META-INF  │ └── additional-spring-configuration-metadata.json  ├── generated-sources  │ └── annotations  ├── maven-archiver  │ └── pom.properties  └── maven-status  └── maven-compiler-plugin  └── compile  └── default-compile  ├── createdFiles.lst  └── inputFiles.lst |
| --- |

Se destaca principalmente que dentro del arquetipo se ha definido un directorio llamado “archetype-resources” que corresponde al directorio donde se ensamblará el proyecto base considerado para la elaboración del arquetipo. Dado lo anterior como ejemplo, se estructuran algunos archivos y carpetas que deberán ser removidos de ser necesario, adicionalmente se declara el archivo “archetype.xml” en el cual se establecen las condiciones de configuración del arquetipo como los io del proyecto que deberán ser considerados, excluidos o los archivos que deberán ser representados paramétricamente dado el comportamiento dinámico que se le desea dar al arquetipo.

En las siguientes secciones se detalla la modificación de este arquetipo recién generado para convertirse en los arquetipos Proxy SOAP y Proxy Rest.

# Arquetipos

## Arquetipo Proxy SOAP

A continuación se despliega la estructura correspondiente al proyecto arquetipo que permite comprender el detalle de las condiciones de proyecto que será desplegado para servicios de tipo proxy SOAP.

| middleware-soap-proxy-archetype/  ├── pom.xml  ├── README.md  ├── src  │ └── main  │ └── resources  │ ├── archetype-resources  │ │ ├── pom.xml  │ │ └── src  │ │ └── main  │ │ ├── java  │ │ │ └── cl  │ │ │ └── coordinador  │ │ │ └── middleware  │ │ │ └── proxy  │ │ │ └── SpringBootSoapProxyApplication.java  │ │ └── resources  │ │ ├── application.properties  │ │ ├── META-INF  │ │ ├── spring-camel-context.xml  │ │ ├── velocity  │ │ │ └── sqlException-soap.vm  │ │ └── wsdl  │ │ └── baseExample.wsdl  │ └── META-INF  │ └── maven  │ └── archetype-metadata.xml |
| --- |

| **ARCHIVO** | **DEFINICIÓN** |
| --- | --- |
| SpringBootSoapProxyApplication.java | Archivo que es parte de la implementación de Spring Boot y será utilizado para el arranque del servicio como punto inicial. |
| application.properties | Archivo de propiedades del proyecto base, en este archivo se deberá incorporar cualquier información paramétrica del proyecto, no se considera bajo ningún orden depender de un archivo externo para el procesamiento. Los parámetros por defecto incluidos para la generación de proyectos son:  camel.springboot.main-run-controller, camel.timeout.redelivery-delay, camel.timeout.max-delivery  camel.soap.basicauth.user, camel.soap.basicauth.pass  soap.service.host, soap.service.port, soap.service.resource. |
| spring-camel-context.xml | Archivo principal del proyecto y es donde se establece toda configuración de las rutas camel basado en spring DSL. |
| sqlException-soap.vm | Archivo de estructura de respuesta del servicio, este archivo será utilizado en situaciones de excepción entregando una respuesta según el servicio que haya sido utilizado, se deberá reemplazar este archivo sobre la implementación según tipo de servicio implementado. |
| baseExample.wsdl | Archivo WSDL utilizado para la elaboración del endpoint inicial, este archivo deberá ser reemplazado según implementación a realizar. |

### 

### Ruta Camel

En la ruta camel se declaran varias condiciones de utilización, aunque las principales tienen relación a los atributos “**maximumRedeliveries**” y “**redeliveryDelay**” los cuales son utilizados para la definición de reintentos. Estos casos serán ejecutados al momento de que no exista la capacidad de conexión al punto de contacto del servicio, lo cual generará una situación de reintentos, estos parámetros serán configuradas en el archivo de propiedades “application.properties” detallado anteriormente.

| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:cxf="http://camel.apache.org/schema/cxf"  xsi:schemaLocation="  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd  http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd">    <!-- Proyecto apache camel que permite realizar un proxy entre servicios del tipo SOAP -->  <camelContext id="s03-proxy" xmlns="http://camel.apache.org/schema/spring" >    <propertyPlaceholder id="placeholder" location="classpath:application.properties"/>  <onException>  <exception>java.lang.Exception</exception>  <redeliveryPolicy maximumRedeliveries="{{camel.timeout.max-delivery}}" redeliveryDelay="{{camel.timeout.redelivery-delay}}"/>  </onException>  <!-- Ruta camel que expone un servicio SOAP basado en una publicación del mismo tipo-->  <route streamCache="true">    <!-- TODO: queda pendiente cambiar el namespace y probar, ver readme.pm-->  <from uri="cxf:http://0.0.0.0:${portProxy}/service.asmx?wsdlURL=classpath:wsdl/baseExample.wsdl&amp;serviceName={http://www.webserviceX.NET}GlobalWeather&amp;portName={http://www.webserviceX.NET}GlobalWeatherSoap&amp;dataFormat=MESSAGE" />  <to uri="cxf:http://{{soap.service.host}}:{{soap.service.port}}{{soap.service.resource}}?dataFormat=MESSAGE" />    <log message="pasando por servicio proxy"></log>  </route>  </camelContext>  </beans> |
| --- |

La implementación de la ruta está basada en apache [CXF](http://cxf.apache.org) la cual establecerá el endpoint de exposición además del endpoint al cual producirá el mensaje. Para este caso se está considerando un “dataformat=MESSAGE” lo que corresponde a una definición SOAP Request que será traspasada directamente desde el servicio consumidor al productor.

### Utilización

La utilización del proceso queda dado por la definición declarada del servicio de tipo arquetipo según Readme.md la cual contiene la siguiente información.

| Coordinador - SOAP Proxy Apache Camel - Spring Boot  =====================================  Permite la generación de servicios Proxy SOAP los cuales serán desplegados sobre un entorno autocontenido con Spring Boot.  ### Consideraciones:  \* El arquetipo no generará rutinas de validación o transformación intermedias, éstas deberán ser ajustadas por el programador al momento de generar el servicio.  \* El arquetipo siempre responderá a una situación de proxy.  \* El arquetipo generará un servicio basado en el modelo "pass through".  \* El contrato del servicio deberá ser adecuado dependiendo de las necesidades de interpretación, tanto de transformación como de invocación hacia y desde servicio de destino.  ### Requerimientos:  \* Spring Boot  \* Apache Maven  \* Java SE 8  Construcción  --------  Para construir e instalar el arquetipo en el repositorio maven se deberá ejecutar la siguiente línea dentro del proyecto.  mvn clean install  Nota: Con la anterior línea el arquetipo quedará instalado en el repositorio local el cual deberá ser versionado sobre Repositorio Maven correspondiente.  Ejecución desde línea de comando  -----------------------------  Después de instalado el arquetipo en el repositorio local, se deberá ejecutar la siguiente línea  para la generación del proyecto:  mvn archetype:generate -DarchetypeGroupId=cl.coordinador.arquetypes -DarchetypeArtifactId=middleware-soap-proxy-archetype -DarchetypeVersion=1.0.0 -DgroupId=cl.coordinador.services -DartifactId=SRV40011-UnifiedSalesSupportService -Dversion=1.0.0 -DportProxy=8888 -DresourceService=/salesSupport/uss -DserviceToHost=www.webservicex.net -DserviceToPort=80 -DserviceToResource=/globalweather.asmx -DinteractiveMode=false -DbasicAuth=false    \* -Dbusiness: atributo que define el nombre de la ruta camel, que servirá para diferenciarla de las demás rutas existentes.  \* -DserviceToHost: nombre o ip del host servicio al cual se dirigirá la petición.  \* -DserviceToPort: puerto del host servicio al cual se dirigirá la petición.  \* -DserviceToresource: nombre del recurso que quedará expuesto.  \* -DresourceService: nombre del recurso que se le dará al servicio expuesto el cual deberá definirse posteriormente si se trata de post o get.  El anterior ejercicio generará un nuevo proyecto llamado "middleware-soap-proxy-base", nombre que se le dió al artifactId previamente.  Finalmente, ejecutar el proyecto de la siguiente forma:  mvn spring-boot:run |
| --- |

Se debe modificar el archivo spring-camel-context.xml para ajustar los parámetros relativos a la wsdl, esta configuración se encuentra descrita en el tag: <from uri="cxf:...”. Los parámetros a modificar son: wsdlURL, serviceName y portName, a continuación podemos ver un ejemplo:

| <from uri="cxf:http://0.0.0.0:8888/RedkneeSoap\_v3\_0/services/SubscriptionService/?**wsdlURL**=classpath:wsdl/SubscriptionService\_1\_1.wsdl&amp;**serviceName**={http://soap.crmapi.util.redknee.com/subscriptions/Subscriptions-v2.0}SubscriptionService&amp;**portName**={http://soap.crmapi.util.redknee.com/subscriptions/Subscriptions-v2.0}SubscriptionsSOAP11port\_http&amp;dataFormat=MESSAGE" /> |
| --- |

El último paso correspondiente a Spring Boot, dejará un entorno de ejecución independiente por servicio, al ejecutar el comando “mvn spring-boot:run”. El anterior comando permitirá ejecutar el entorno de ejecución según el siguiente ejemplo para este proyecto.

| Coordinador - SOAP Proxy Apache Camel - Spring Boot  =====================================  Permite la generación de servicios Proxy SOAP los cuales serán desplegados sobre un entorno autocontenido con Spring Boot.  ### Consideraciones:  \* El arquetipo no generará rutinas de validación o transformación intermedias, éstas deberán ser ajustadas por el programador al momento de generar el servicio.  \* El arquetipo siempre responderá a una situación de proxy.  \* El arquetipo generará un servicio basado en el modelo "pass through".  \* El contrato del servicio deberá ser adecuado dependiendo de las necesidades de interpretación, tanto de transformación como de invocación hacia y desde servicio de destino.  ### Requerimientos:  . \_\_\_\_ \_ \_\_ \_ \_  /\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \  ( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  \\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /  =========|\_|==============|\_\_\_/=/\_/\_/\_/  :: Spring Boot :: (v1.4.1.RELEASE)  2017-09-03 21:49:14.441 INFO 16377 --- [ main] c.w.m.p.SpringBootRestProxyApplication : Starting SpringBootRestProxyApplication on usera-Lenovo-Y720-15IKB with PID 16377 (/home/user/git/coordinador/coordinador-billing-integration/middleware-rest-proxy-base/target/classes started by user in /home/user/git/coordinador/coordinador-billing-integration/middleware-rest-proxy-base)  2017-09-03 21:49:14.443 INFO 16377 --- [ main] c.w.m.p.SpringBootRestProxyApplication : No active profile set, falling back to default profiles: default  2017-09-03 21:49:14.472 INFO 16377 --- [ main] s.c.a.AnnotationConfigApplicationContext : Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@285be679: startup date [Sun Sep 03 21:49:14 CLST 2017]; root of context hierarchy  2017-09-03 21:49:14.743 INFO 16377 --- [ main] o.s.b.f.xml.XmlBeanDefinitionReader : Loading XML bean definitions from class path resource [spring-camel-context.xml]  2017-09-03 21:49:15.634 INFO 16377 --- [ main] trationDelegate$BeanPostProcessorChecker : Bean 'org.apache.camel.spring.boot.CamelAutoConfiguration' of type [class org.apache.camel.spring.boot.CamelAutoConfiguration$$EnhancerBySpringCGLIB$$9e439522] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)  2017-09-03 21:49:16.054 INFO 16377 --- [ main] o.a.c.i.converter.DefaultTypeConverter : Loaded 205 type converters  2017-09-03 21:49:16.451 INFO 16377 --- [ main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup  2017-09-03 21:49:16.455 INFO 16377 --- [ main] o.s.c.support.DefaultLifecycleProcessor : Starting beans in phase 2147483647  2017-09-03 21:49:16.462 INFO 16377 --- [ main] o.a.camel.spring.boot.RoutesCollector : Loading additional Camel XML routes from: classpath:camel/\*.xml  2017-09-03 21:49:16.462 INFO 16377 --- [ main] o.a.camel.spring.boot.RoutesCollector : Loading additional Camel XML rests from: classpath:camel-rest/\*.xml  2017-09-03 21:49:16.462 INFO 16377 --- [ main] o.a.camel.spring.boot.RoutesCollector : Starting CamelMainRunController to ensure the main thread keeps running  2017-09-03 21:49:16.465 INFO 16377 --- [inRunController] o.a.camel.spring.SpringCamelContext : Apache Camel 2.18.1 (CamelContext: camel-1) is starting  2017-09-03 21:49:16.465 INFO 16377 --- [inRunController] o.a.c.m.ManagedManagementStrategy : JMX is enabled  2017-09-03 21:49:16.467 INFO 16377 --- [ main] c.w.m.p.SpringBootRestProxyApplication : Started SpringBootRestProxyApplication in 2.188 seconds (JVM running for 4.577)  2017-09-03 21:49:16.526 INFO 16377 --- [inRunController] o.a.c.i.DefaultRuntimeEndpointRegistry : Runtime endpoint registry is in extended mode gathering usage statistics of all incoming and outgoing endpoints (cache limit: 1000)  2017-09-03 21:49:16.688 INFO 16377 --- [inRunController] o.a.camel.spring.SpringCamelContext : StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more details at http://camel.apache.org/stream-caching.html  2017-09-03 21:49:16.809 INFO 16377 --- [inRunController] org.restlet : Starting the Apache HTTP client  2017-09-03 21:49:16.821 INFO 16377 --- [inRunController] org.restlet.Component.Server : Starting the internal [HTTP/1.1] server on port 8888  2017-09-03 21:49:16.824 INFO 16377 --- [inRunController] o.a.camel.spring.SpringCamelContext : Route: route1 started and consuming from: **http://0.0.0.0:8888/saldo/cliente/1?restletMethods=GET**  2017-09-03 21:49:16.824 INFO 16377 --- [inRunController] o.a.camel.spring.SpringCamelContext : Total 1 routes, of which 1 are started.  2017-09-03 21:49:16.825 INFO 16377 --- [inRunController] o.a.camel.spring.SpringCamelContext : Apache Camel 2.18.1 (CamelContext: camel-1) started in 0.360 seconds |
| --- |

## Arquetipo Proxy REST

A continuación se despliega la estructura correspondiente al proyecto arquetipo que permite comprender el detalle de las condiciones de proyecto que será desplegado para servicios de tipo proxy REST.

| └── api\_personas\_rrhh\_personas  ├── template-prueba.yaml  ├── pom.xml  ├── README.md  ├── src  │ └── main  │ ├── java  │ │ └── cl  │ │ └── coordinador  │ │ └── gerencia  │ │ └── departamento  │ │ ├── controllers  │ │ │ └── PersonaController.java  │ │ ├── CreateServiceApplication.java  │ │ ├── exceptions  │ │ │ ├── ApiExceptionHandler.java  │ │ │ ├── BadRequestException.java  │ │ │ ├── ErrorMessage.java  │ │ │ ├── InternalException.java  │ │ │ └── NotFoundException.java  │ │ └── models  │ │ ├── dao  │ │ │ └── PersonaDao.java  │ │ ├── entity  │ │ │ └── Persona.java  │ │ └── services  │ │ ├── IPersonaService.java  │ │ └── PersonaServiceImp.java  │ └── resources  │ ├── application.properties  │ ├── META-INF  │ │ └── additional-spring-configuration-metadata.json  │ ├── routes  │ ├── velocity  │ └── wsdl  └── target  ├── api-personas-rrhh-personas-1.0-SNAPSHOT.jar  ├──api-personas-rrhh-personas-1.0-SNAPSHOT.jar.original  ├── classes  │ ├── application.properties  │ ├── cl  │ │ └── coordinador  │ │ └── gerencia  │ │ └── departamento  │ │ ├── controllers  │ │ │ └── PersonaController.class  │ │ ├── CreateServiceApplication.class  │ │ ├── exceptions  │ │ │ ├── ApiExceptionHandler.class  │ │ │ ├── BadRequestException.class  │ │ │ ├── ErrorMessage.class  │ │ │ ├── InternalException.class  │ │ │ └── NotFoundException.class  │ │ └── models  │ │ ├── dao  │ │ │ └── PersonaDao.class  │ │ ├── entity  │ │ │ └── Persona.class  │ │ └── services  │ │ ├── IPersonaService.class  │ │ └── PersonaServiceImp.class  │ └── META-INF  │ └── additional-spring-configuration-metadata.json  ├── generated-sources  │ └── annotations  ├── maven-archiver  │ └── pom.properties  └── maven-status  └── maven-compiler-plugin  └── compile  └── default-compile  ├── createdFiles.lst  └── inputFiles.lst |
| --- |

| **ARCHIVO** | **DEFINICIÓN** |
| --- | --- |
| SpringBootSoapProxyApplication.java | Archivo que es parte de la implementación de Spring Boot y será utilizado para el arranque del servicio como punto inicial. |
| application.properties | Archivo de propiedades del proyecto base, en este archivo se deberá incorporar cualquier información paramétrica del proyecto, no se considera bajo ningún orden depender de un archivo externo para el procesamiento. Los parámetros incluidos por defecto para la generación de proyectos son:  camel.springboot.main-run-controller,  camel.timeout.redelivery-delay, camel.timeout.max-delivery,  camel.rest.basicauth.user, camel.rest.basicauth.pass |
| spring-camel-context.xml | Archivo principal del proyecto y es donde se establece toda configuración de las rutas camel basado en spring DSL. |
| sqlException-json.vm | Archivo de estructura de respuesta del servicio, este archivo será utilizado en situaciones de excepción entregando una respuesta según el servicio que haya sido utilizado, se deberá reemplazar este archivo sobre la implementación según tipo de servicio implementado. |

### Ruta Camel

En la ruta camel se declaran varias condiciones de utilización, aunque las principales tienen relación a los atributos “**maximumRedeliveries**” y “**redeliveryDelay**” los cuales son utilizados para la definición de reintentos. Estos casos serán ejecutados al momento de que no exista la capacidad de conexión al punto de contacto del servicio, lo cual generará una situación de reintentos, estos parámetros serán configuradas en el archivo de propiedades “application.properties” detallado anteriormente.

| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:camel="http://camel.apache.org/schema/spring"  xsi:schemaLocation="  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">    <camelContext xmlns="http://camel.apache.org/schema/spring" >  <propertyPlaceholder id="placeholder" location="classpath:application.properties"/>  <onException>  <exception>java.lang.Exception</exception>  <redeliveryPolicy maximumRedeliveries="{{camel.timeout.max-delivery}}" redeliveryDelay="{{camel.timeout.redelivery-delay}}"/>  <log message="error al enviar el mensaje"></log>  <to uri="velocity:velocity/sqlException-json.vm"/>  </onException>  <intercept>  <transform><simple>{"request":{"headers"{"http":"sdasdasd","http":"sdasdasd","http":"sdasdasd","http":"sdasdasd",}}}</simple></transform>  <log message="mensaje: ${body}"></log>  </intercept>    <route>  <from uri="restlet:http://0.0.0.0:${portProxy}/${resourceService}?restletMethod=get" />  #if ( $basicAuth == true)  <to uri="restlet:${serviceTo}?restletMethod=get&amp;CamelRestletLogin={{camel.rest.basicauth.user}}&amp;CamelRestletPassword={{camel.rest.basicauth.pass}}" />  #end    #if ( $basicAuth == false)  <to uri="restlet:${serviceTo}?restletMethod=get" />  #end  </route>    </camelContext>  </beans> |
| --- |

El último paso correspondiente a Spring Boot, dejará un entorno de ejecución independiente por servicio, para realizar esto ejecutaremos el comando “**mvn spring-boot:run**”.

### Utilización

La utilización del proceso queda dado por la definición declarada del servicio de tipo arquetipo según Readme.md la cual contiene la siguiente información.

| # Spring Boot - Proyecto Base Arquitectura - Microservicio  Este componente corresponde a un proyecto base que permite crear una estructura basado en microservicios, este componente tiene como objetivo generar la base de implementación para que pueda ser ajustada respectivamente  Este proyecto pertenece a la especificación de liberación de documentación \*"[CEN] - Documento Lineamientos Técnicos APIs de Servicios - v1.0"\*    ## Pre Requisitos  \* Instalación de Postgresql: se recomienda que esta habilitación sea realizada para el sistema operativo correspondiente para la versión compatible con el driver https://www.postgresql.org/download/  \* Contar con OpenJDK 11 instalado entorno local  ## Pre configuración  Una vez que la base de datos sea creada, se deberá configurar el usuario y crear base de datos de test con los siguientes comandos respectivamente  ```  CREATE USER postgres SUPERUSER;  CREATE DATABASE test\_db WITH OWNER postgres;  ```  ## Ejecución del proyecto base sobre mvn archetype  Para iniciar se deberá crear el proyecto base mediante el arquetipo, para esto se deberá utilizar la siguiente instrucción  ```  mvn archetype:generate -DarchetypeGroupId=cl.coordinador.arquetipos \  -DarchetypeArtifactId=archetype-rest-microservice-crud \  -DarchetypeVersion=1.0.0 \  -DgroupId=cl.coordinador.personas \  -DartifactId=api\_personas\_rrhh\_personas  ```  ## Compilación y ejecución  La primera parte corresponderá a la compilación la cual deberá ser realizada mediante el comando:  ```  mvn clean install  ```  Posteriormente, para ejecutar el servicio deberá realizarlo mediante el comando:  ```  mvn spring-boot:run  ```  ## Pruebas  Las pruebas podrán ser realizadas mediante postman, se adjunta proyecto para que pueda ser importado y testeado respectivamente, adicionalmente se establecen los comandos curl respectivamente  ###### Crear Registro  ```  curl --location --request POST 'http://localhost:8082/v1/gerencia/departamento/funcionalidad/create' \  --header 'Content-Type: application/json' \  --data-raw '{  "name":"testName",  "rut": 11111  }'  ```  ###### Buscar todos los registros  ```  curl --location --request GET 'http://localhost:8082/v1/gerencia/departamento/funcionalidad/findAll'  ```  ###### Buscar registro por ID  ```  curl --location --request GET 'http://localhost:8082/v1/gerencia/departamento/funcionalidad/findById/1'  ```  ###### Buscar registro por Rut o por atributo custom  ```  curl --location --request GET 'http://localhost:8082/v1/gerencia/departamento/funcionalidad/findByRut/11111'  ```  ###### Eliminar registro por ID  ```  curl --location --request DELETE 'http://localhost:8082/v1/gerencia/departamento/funcionalidad/delete/8'  ```  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Coordinador - REST Proxy Apache Camel - Spring Boot  ===========================================  Permite generar proyectos del tipo API Rest los cuales serán expuestos basados en una situación de proxy, internamente si el cuerpo cambiará entre el servicio consumido  y producido, se deberá generar un proceso de transformación eventualmente y todas las actividades adicionales que sean necesarias.  ### Consideraciones:  \* El arquetipo no generará rutinas de validación o transformación intermedias, éstas deberán ser ajustadas por el programador al momento de generar el servicio.  \* El arquetipo siempre responderá a una situación de proxy.  \* El arquetipo generará un servicio que siempre consumirá un servicio rest con método get.  \* El arquetipo generará un servicio basado en el modelo "pass through".  \* El contrato del servicio deberá ser adecuado dependiendo de las necesidades de interpretación, tanto de transformación como de invocación hacia y desde servicio de destino.  ### Requerimientos:  \* Spring Boot  \* Apache Maven  \* Java SE 8  Construcción  ------------  Para construir e instalar el arquetipo en el repositorio maven se deberá ejecutar la siguiente línea dentro del proyecto  mvn clean install  Nota: Con la anterior línea el arquetipo quedará instalado en el repositorio local el cual deberá ser versionado sobre Repositorio Maven correspondiente.  Ejecución desde línea de comando  --------------------------------  Después de instalado el arquetipo en el repositorio local, se deberá ejecutar la siguiente línea  para la generación del proyecto.  mvn archetype:generate -DarchetypeGroupId=cl.coordinador.middleware.archetype -DarchetypeArtifactId=middleware-rest-proxy-archetype -DarchetypeVersion=1.0-SNAPSHOT -DgroupId=cl.coordinador.middleware.proxy -DartifactId=SRV40011-UnifiedSalesSupportService -Dversion=1.0.0 -DresourceService=saldo/cliente/1 -DportProxy=8888 -DserviceTo=http://jsonplaceholder.typicode.com/posts/1 -DinteractiveMode=false -DbasicAuth=false  mvn archetype:generate \  -DarchetypeGroupId=cl.coordinador.middleware.archetype \  -DarchetypeArtifactId=middleware-rest-proxy-archetype \  -DarchetypeVersion=1.0-SNAPSHOT \  -DgroupId=cl.coordinador.middleware.proxy \  -DartifactId=test    \* -DresourceService: atributo que define el nombre de la ruta camel, que servirá para diferenciarla de las demás rutas existentes.  \* -DportProxy: puerto en el cual se expondrá el servicio.  \* -DserviceTo: endpoint al cual se producirá los mensajes del servicio.  \* -DbasicAuth: aplica seguridad básica para el servicio, true con seguridad, false sin seguridad.  El anterior ejercicio generará un nuevo proyecto llamado "SRV40011-UnifiedSalesSupportService", nombre que se le dió al artifactId previamente.  Finalmente, ejecutar el proyecto de la siguiente forma:  mvn spring-boot:run  ### Referencias:  https://github.com/tmforum/RESTGUIDELINESV2/blob/master/TMF630\_REST\_API\_Design\_Guidelines\_Part\_1\_R14.5.0-5.pdf1\_R14.5.0-5.pdf |
| --- |

## Registro del Arquetipo en el repositorio Nexus

Para registrar el arquetipo en el repositorio maven de El Coordinador, se deben ejecutar los siguientes pasos:

1. Agregar al archivo pom.xml del proyecto la información correspondiente al repositorio nexus.

| <distributionManagement>  <repository>  <id>coordinador-local-repository</id>  <url>http://maven.coordinador.cl:8081/repository/coordinador-local-repository/</url>  </repository>  </distributionManagement> |
| --- |

2. Configurar en el pom.xml plugins, recursos y extensiones.

| <build>  <extensions>  <extension>  <groupId>org.apache.maven.archetype</groupId>  <artifactId>archetype-packaging</artifactId>  <version>2.2</version>  </extension>  </extensions>  <pluginManagement>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-archetype-plugin</artifactId>  <version>2.2</version>  </plugin>  </plugins>  </pluginManagement>  <resources>  <resource>  <directory>src/main/resources</directory>  <filtering>false</filtering>  </resource>  <resource>  <directory>src/main/resources-filtered</directory>  <filtering>true</filtering>  <includes>  <include>\*\*/archetype-metadata.xml</include>  </includes>  </resource>  </resources>  </build> |
| --- |

3. En el archivo settings.xml ubicado por defecto en la ruta “<HomeUsuario>/.m2/” se debe configurar agregando la siguiente información: usuario, contraseña e id del repositorio nexus configurado anteriormente.

| <?xml version="1.0" encoding="UTF-8" standalone="no"?>  <settings>  <servers>  <server>  <id>coordinador-local-repository</id>  <username>admin</username>  <password>admin</password>  </server>  </servers>  <profiles>  <!-- Nexus Coordinador-->  <profile>  <id>coordinador-local-repository</id>  <repositories>  <repository>  <id>coordinador-local-repository</id>  <name>coordinador local repository</name>  <url>http://maven.coordinador.cl:8081/repository/coordinador-local-repository/</url>  <releases>  <enabled>true</enabled>  <updatePolicy>never</updatePolicy>  </releases>  <snapshots>  <enabled>false</enabled>  <updatePolicy>daily</updatePolicy>  </snapshots>  </repository>  </repositories>  <pluginRepositories>  <pluginRepository>  <id>coordinador-local-repository</id>  <name>coordinador local repository</name>  <url>http://maven.coordinador.cl:8081/repository/coordinador-local-repository/</url>  <releases>  <enabled>true</enabled>  <updatePolicy>never</updatePolicy>  </releases>  <snapshots>  <enabled>false</enabled>  <updatePolicy>daily</updatePolicy>  </snapshots>  </pluginRepository>  </pluginRepositories>  </profile>  </profiles>  <activeProfiles>  <activeProfile>coordinador-local-repository</activeProfile>  </activeProfiles>  </settings> |
| --- |

4. Para instalar en el repositorio nexus, en la terminal ejecutamos el siguiente comando:

| mvn clean deploy |
| --- |

## Generación de Proyecto con Servicio desde Arquetipo en repositorio Nexus

Para generar un proyecto a partir del arquetipo de soap desde el repositorio maven se deben ejecutar los siguientes pasos:

1. A continuación se indica cómo generar el arquetipo soap desde nexus.

| mvn archetype:generate -DarchetypeGroupId=cl.coordinador.arquetypes -DarchetypeArtifactId=middleware-soap-proxy-archetype -DarchetypeVersion=1.0.0 -DgroupId=cl.coordinador.services -DartifactId=SRV40011-UnifiedSalesSupportService -Dversion=1.0.0 -DportProxy=8888 -DresourceService=/salesSupport/uss -DserviceToHost=10.120.145.160 -DserviceToPort=9080 -DserviceToResource=/salesSupport/uss -DinteractiveMode=false -DbasicAuth=false |
| --- |

## 

## Incorporación de Jaeger

Para lograr incorporar Jaeger en el arquetipo, se agregó la dependencia a el archivo pom.xml y luego se creó un Tracer dentro de el archivo JaegerConfig.java del arquetipo.

pom.xml:

| <dependency>  <groupId>io.opentracing.contrib</groupId>  <artifactId>opentracing-spring-jaeger-web-starter</artifactId>  <version>3.3.1</version>  </dependency> |
| --- |

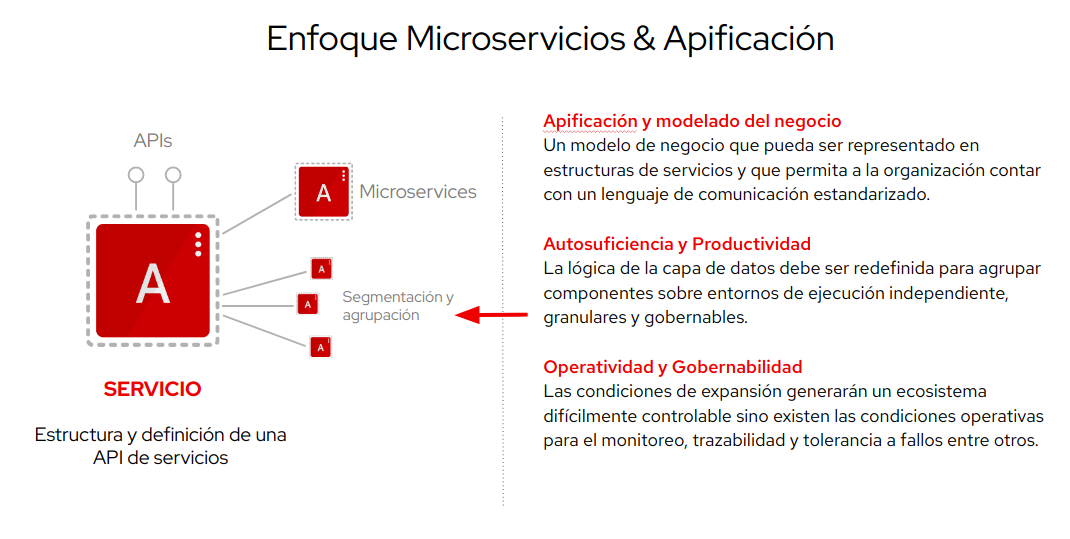
JaegerConfig.java

| package cl.coordinador.gerencia.departamento.config;  import io.jaegertracing.internal.JaegerTracer;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  @Configuration  public class JaegerConfig {  @Bean  public JaegerTracer jaegerTracer() {  io.jaegertracing.Configuration.SamplerConfiguration samplerConfig = io.jaegertracing.Configuration.SamplerConfiguration  .fromEnv().withType("const").withParam(1);  io.jaegertracing.Configuration.ReporterConfiguration reporterConfig = io.jaegertracing.Configuration.ReporterConfiguration  .fromEnv().withLogSpans(true);  io.jaegertracing.Configuration.SenderConfiguration senderConfig = io.jaegertracing.Configuration.SenderConfiguration  .fromEnv()  .withEndpoint(  "http://jaeger-all-in-one-inmemory-collector:14268/api/traces");  io.jaegertracing.Configuration config = new io.jaegertracing.Configuration("api-personas-rrhh-personas-traces")  .withSampler(samplerConfig).withReporter(reporterConfig.withSender(senderConfig));  return config.getTracer();  }  } |
| --- |

Esta integración de Jaeger, permite que se trace todas las consultas realizadas al arquetipo.

# Arquitectura de APIs

Cuando nos referimos al entendimiento es para poder conocer cuánto nos falta para lograr un estado final adecuado para nuestra organización y sobre la realidad de TI, esto implica que por más que el mundo tenga una referencia tecnológica y metodológica, posiblemente no es lo más adecuado para nuestra organización y esto se debe a que todas las compañías son diferentes y nuestros objetivos idealistas no podrán ser cubiertos en el corto plazo, sin embargo esto no impide que podamos proyectar nuestra solución de valor en fases o segmentos iterativos e incrementales.



Tres aspectos fundamentales al momento de comenzar a generar soluciones de APIs son la capacidad de segmentación del negocio que llamaremos “Apificación”, la capacidad de maximizar los ciclos de desarrollo de software o APIs de servicios que definiremos como Autosuficiencia o productividad y finalmente la operación o gobernabilidad de nuestros productos de APIs de servicio.

Cada uno de estos aspectos son clave para no sólo hablar un único lenguaje con el negocio, sino que hacerlo de forma eficiente, por ejemplo, si el negocio comprender qué APIs tiene a su disposición y sabe muy bien lo que hacen y lo que no, podrían pedirnos con mayor precisión un cambio para ajustar su comportamiento y si contamos con un ciclo de desarrollo de software automatizado debe implicar que poseemos mayor control sobre estos procesos y que podemos precisar con mayor exactitud al cliente el tiempo necesario para lograr su liberación en un corto plazo. Además una vez la API de servicio haya finalizado, el cliente interno podrá ver cómo ésta se comporta a través de herramientas claras de operación y gobernabilidad.

Todo lo anterior apunta hacia un modelo de agilidad que involucra a los procesos y habilidades de las personas que deberán ir madurando además durante este proceso.

## Normar Base de Arquitectura

Antes de hablar de un modelo de arquitectura específico, se debe tener en cuenta que todo debe iniciar por una base fundamental que permitirá organizar al TI en torno a las principales capas las cuales se recomienda lo que la siguiente gráfica representa.

El coordinador asociará su modelo de arquitectura de aplicaciones para efectos de integración e interoperabilidad basado en capas generales como se puede expresar en la gráfica.

