



Universidad  
Tecnológica  
de Pereira

**DEFINICIÓN DE UNA ARQUITECTURA PARA LA TRANSFORMACIÓN  
DE SOFTWARE CENTRALIZADO A SOFTWARE BASADO EN  
MICROSERVICIOS EN EL ÁMBITO WEB**

**JUAN PABLO GÓMEZ GALLEG**

**UNIVERSIDAD TECNOLÓGICA DE PEREIRA**  
**FACULTAD DE INGENIERÍA**  
**MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**  
**PEREIRA**  
**2017**

**DEFINICIÓN DE UNA ARQUITECTURA PARA LA TRANSFORMACIÓN  
DE SOFTWARE CENTRALIZADO A SOFTWARE BASADO EN  
MICROSERVICIOS EN EL ÁMBITO WEB**

**JUAN PABLO GÓMEZ GALLEG**

Proyecto presentado como requisito parcial optar el título de:  
**Magister en ingeniería de sistemas y computación**

Director:

**Ing. JORGE IVÁN RIOS PATIÑO**

Phd en Ingenierías

Universidad Tecnológica de Pereira

**UNIVERSIDAD TECNOLÓGICA DE PEREIRA  
FACULTAD DE INGENIERÍA  
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN  
PEREIRA  
2017**

## AGRADECIMIENTOS

A mis padres, a mi esposa por su paciencia, a mi amigo ingeniero Gonzalo Muller por sembrarme la idea de llevar acabo este proyecto, a mis compañeros en Globant por resolver mis duda técnicas, al ingeniero Jorge Iván Ríos por su paciencia en revisar este documento y a mi amigo el ingeniero Gustavo Andrés Castro por sus aportes y motivación a terminar este iniciativa.

## **Índice de Contenido**

RESUMEN.....	1
INTRODUCCIÓN.....	2
PLANTEAMIENTO DEL PROBLEMA.....	4
JUSTIFICACIÓN .....	5
HIPÓTESIS .....	6
ESTADO DEL ARTE .....	7
MARCO TEÓRICO.....	9
Definición de microservicios.....	9
Ventajas de los microservicios .....	10
Desventajas de los microservicios .....	12
Modelado de los servicios.....	13
Desacoplamiento de artefactos.....	15
Fragmentación por base de datos .....	16
El proceso scrum .....	20
MARCO CONCEPTUAL.....	22
Planeamiento y factibilidad .....	22
Educación y análisis de requisitos .....	23
Especificación de requisitos .....	23
Validación de requisitos .....	24
Diseño y definición de microservicios .....	24
Desarrollo y pruebas de unidad .....	25
Pruebas de integración, contratos y punto a punto .....	25
Despliegue e integración continua .....	26

Diagrama de flujo de las etapas .....	27
<b>OBJETIVOS GENERAL Y ESPECIFICOS .....</b>	<b>28</b>
Objetivo General .....	28
Objetivos Específicos.....	28
<b>DESARROLLO .....</b>	<b>29</b>
Modelado De Arquitectura .....	29
Protocolo .....	30
Dominio .....	30
Persistencia.....	30
Externo .....	30
Ecosistema de microservicios .....	31
Fragmentación conceptual .....	31
Planeamiento y factibilidad .....	33
Determinar el nivel de madurez.....	33
Propiedad intelectual sobre el código fuente.....	34
Equipo de desarrollo calificado.....	34
Comenzar con artefactos de baja criticidad.....	34
Educación y análisis de requisitos .....	36
Definición de metodología ágil y técnicas de educación .....	36
Lista de requisitos.....	36
Especificación de requisitos .....	37
Construcción de historias de usuario, clasificación y priorización .....	37
Construcción de prototipos.....	37
Validación de requisitos .....	38
Validación de historias de usuario .....	38

Priorización y estimación.....	38
Diseño y definición de microservicios .....	39
Modelado de servicios .....	39
Fragmentación del empaquetamiento del monolito .....	41
Preparación de la capa de datos de los microservicios.....	42
Refactorizando el código del monolito.....	43
Desarrollo y pruebas de unidad .....	45
Registro de microservicios .....	45
Manejo De Tolerancia A Fallos .....	48
Configuración centralizada .....	50
Balanceo de carga y monitoreo de llamadas.....	52
Desacoplamiento de la comunicación entre los microservicios.....	54
Manejo de bitácora (Log).....	55
Pruebas de unidad .....	56
Pruebas de integración, contratos y punto a punto.....	57
Pruebas de integración.....	57
Pruebas de contratos .....	58
Pruebas punto a punto .....	58
Pruebas de producción.....	59
Despliegue e integración continua .....	60
Preparación entorno virtualización .....	60
Integración continua y entrega continua.....	61
VALIDACIÓN DE LA METODOLOGIA.....	65
Preparación del ambiente de desarollo.....	65
Aplicaciones y herramientas de desarollo necesarias: .....	65

Configuración y consideraciones de las herramientas a instalar.....	66
Descarga del código fuente, preparación de la base de datos y despliegue de la aplicación .....	67
Diseño Y Definición De Microservicios.....	71
Modelado de servicios.....	71
Fragmentación del monolito .....	74
Implementación de la fase I .....	76
Inspección del código fuente de autenticación en el monolito.....	76
Desprendimiento del código fuente del contexto de autenticación en el monolito.....	78
Creación el servicio de Autenticación.....	80
Inspección del código fuente de autores en el monolito .....	93
Desprendimiento del código fuente del contexto de propiedad intelectual en el monolito.....	94
Creación del servicio de autores .....	97
Creación del servicio y de los clientes del descubrimiento.....	110
Creación del servicio y de los clientes de la configuración .....	121
Creación del servicio y clientes de la bitácora.....	131
Implementación de patrón cortocircuito en los microservicios.....	139
Creación del servicio y clientes del monitoreo.....	141
Implementación de la fase II .....	146
Inspección del código fuente de autenticación en el monolito .....	146
Desprendimiento del código fuente del contexto de publicaciones en el monolito.....	147
Creación del servicio publicaciones.....	150
Implementación fase III .....	165

Inspección del código fuente de folcsonomia en el monolito .....	165
Desprendimiento del código fuente del contexto de publicaciones en el monolito.....	166
Creación del servicio categories.....	170
Integración de los servicios por medio de mensajería .....	184
Implementación de la fase IV .....	191
Creación de archivos Dockerfile para cada proyecto .....	191
Creación de archivos docker-compose.yml.....	191
Creación de nuevo perfil de configuración docker.....	191
Creación de archivos DockerFile por proyecto.....	191
Creación de archivo docker-compose.yml:.....	193
Creación de perfiles docker en archivos de configuración .....	197
Implementación de la fase V .....	201
Instalación de los artefactos en los entornos de despliegue .....	213
Anexos de la validación de la metodología .....	219
Guía para la instalación de apache tomcat .....	219
Guía para instalación de servidor y cliente Mysql .....	223
Guía para instalación de GitStack .....	227
Guía para instalación de RabbitMQ.....	230
Guía de instalación de Docker .....	234
Guía de instalación de Jenkins.....	238
Guía de instalación de Nexus.....	244
Guía de instalación de Rundeck.....	246
CONCLUSIONES .....	248
Trabajos futuros .....	250

BIBLIOGRAFÍA .....	251
--------------------	-----

# Índice de Ilustraciones

Ilustración 1. La productividad vs la complejidad en los microservicios. (Fowler 2015)	11
Ilustración 2 Ejemplo de bounded contexts. (Fowler, M. 2014) .....	13
Ilustración 3. Manejando la creación de un cliente por medio de orquestación. (Newman, 2015, Figura 4-3) .....	15
Ilustración 4 Manejando la creación del cliente por medio de coreografía. (Newman,2015, Figura 4-4) .....	15
Ilustración 5 Remoción de una llave foránea entre dos microservicios. (Newman,2015, Figura 5-2,Figura 5-3) .....	17
Ilustración 6 Creación de un nuevo bounded context para reutilización de datos. (Newman,2015,Figura 5-4) .....	18
Ilustración 7 Descomposición de tabla para desacoplamiento entre contextos. (Newman,2015,Figura5-5) .....	19
Ilustración 8 Etapas de la separación de un servicio. (Newman 2015,Figura 5-9) ....	19
Ilustración 9 Pirámide de pruebas de Cohn.(Newman 2015, Figura 7-2).....	26
Ilustración 10 Diagrama de flujo de actividades necesarias para la transformación de un monolito a microservicios .....	27
Ilustración 11 Anatomía de un microservicio (Clemson,2014) .....	29
Ilustración 12 Ecosistema de microservicios (Clemson,2014) .....	31
Ilustración 13 Refactorización de un monolito a microservicios. (Richardson,2016) .....	32
Ilustración 14 Adaptación de la escala de Cokcburn para determinar el esfuerzo de la transformación de un monolito a microservicios (Larman 2004, Figura 7.2).....	35
Ilustración 15 Caso de ejemplo de extracción de contextos .....	40
Ilustración 16 Resolución de ambigüedad de un modelo entre varios contextos (Lerman,2014).....	40
Ilustración 17 Particionamiento del empaquetamiento de la aplicación monolítica (Brown 2016) .....	42
Ilustración 18 Arquitectura de un servidor de registro de microservicios (Haddad,2012) .....	45

Ilustración 19 Enrutamiento de microservicios por medio de filtro a través de un proxy (Sastrasinh,2016) .....	47
Ilustración 20 Representación del patrón disyuntor .....	48
Ilustración 21 Arquitectura del servidor de configuración .....	50
Ilustración 22 Balanceador de carga centralizado con microservicios (Newman 2015, Figura 11-4) .....	52
Ilustración 23 Fragmentación de la pirámide de Cok de acuerdo a los niveles de granularidad de las pruebas (Clemson 2014) .....	57
Ilustración 24 Múltiples servicios ejecutándose en un mismo host (Newman 2015, Figura 6-10) .....	60
Ilustración 25 Proceso de integración continua para varios microservicios .....	62
Ilustración 26 Proceso de Integración continua y entrega continua de un solo microservicio.....	63
Ilustración 27 Ubicación del archivo .war desde el explorador de Ms Windows .....	69
Ilustración 28 Endpoint del microservicio que permite saber el estado de salud del monolito .....	69
Ilustración 29 Consulta de servicio autores en el monolito .....	70
Ilustración 30 Descripción general del proyecto restfukl-blog obtenida en https://github.com/benjsicam/restful-blog .....	71
Ilustración 31 Contexto existentes en el monolito .....	72
Ilustración 32 Desambigüación de los contexto del monolito .....	73
Ilustración 33 Fragmento clase AuthorController en restful-blog antes de refactorización .....	78
Ilustración 34 Fragmento clase AuthorService en restful-blog antes de refactorización .....	78
Ilustración 35 Fragmento UserDetailsServiceImpl en proyecto restful-blog antes de refactorizar .....	79
Ilustración 36 Refactorización de tablas para contexto de autenticación .....	79
Ilustración 37 Formulario de start.spring.io para la creación dek microservicio restful-blog-credentials .....	80
Ilustración 38 Contenido del archivo pom.xml del proyecto resftul-blog-credentials. ....	82

Ilustración 39 Clase RestfulBlogCredentialsApplication del proyecto restful-blog-credentials .....	84
Ilustración 40 Clase CredentialsService del proyecto restful-blog-credentials .....	85
Ilustración 41 Clase CredentialsRepository del proyecto restful-blog-credentials ....	85
Ilustración 42 Clase Credentials del proyecto restful-blog-credentials .....	86
Ilustración 43 Archivo bootstrap.yml del proyecto restful-blog-credentials.....	87
Ilustración 44 Archivo application.yml del proyecto restful-blog-credentials .....	87
Ilustración 45 Script de la tabla Credentials para la base de datos restful_blog_auth .	88
Ilustración 46 Script de la valores de la tabla Credentials para la base de datos restful_blog_auth .....	88
Ilustración 47 Obtención de las credenciales del usuario admin desde el microservicio credentials .....	89
Ilustración 48 Clase SecurityConfiguration del proyecto restful-blog-auth.....	90
Ilustración 49 Clase UserDetailsServiceImpl del proyecto restful-blog-auth .....	90
Ilustración 50 Clase CredentialsClient del proyecto restful-blog .....	91
Ilustración 51 Refactorización de la clase UserDetailsServiceImpl en el proyecto restful-blog .....	92
Ilustración 52 Clase AuthorController en el proyecto restful-blog .....	94
Ilustración 53 Clase AuthorService en el proyecto restful-blog .....	95
Ilustración 54 Tabla Author luego de refactorización del contexto credenciales .....	96
Ilustración 55 Creación de microservicio restful-blog-author .....	97
Ilustración 56 Archivo pom.xml del proyecto restful-blog-author .....	99
Ilustración 57 Clase RestfulBlogAuthorApplication del proyecto restful-blog-authors .....	101
Ilustración 58 Clase AuthorService del proyecto restful-blog-author.....	103
Ilustración 59 Fragmento de la clase Author del proyecto restful-blog-author .....	104
Ilustración 60 Clase AuthorRepository del proyecto restful-blog-author.....	104
Ilustración 61 Clase CredentialsClientService del proyecto restful-blog-author.....	104
Ilustración 62 Clase PostsClientService del proyecto restful-blog-author .....	105
Ilustración 63 Clase FeingClientConfiguration del proyecto restful-blog-author.....	105
Ilustración 64 Clase UserDetailsServiceImpl del proyecto restful-blog-author .....	106

Ilustración 65 Fragmento de la clase PostController del proyecto restful-blog.....	107
Ilustración 66 Archivo bootstrap.yml del proyecto restful-blog-author .....	107
Ilustración 67 Archivo application.yml del proyecto restful-blog-author .....	108
Ilustración 68 Clase AuthorClient del proyecto restful-blog .....	108
Ilustración 69 Script de la tabla Author para la base de datos restful-blog-author ...	109
Ilustración 70 Valores de inserción para la tabla Authors.....	109
Ilustración 71 Consulta de autores al microservicio Authors .....	110
Ilustración 72 Formulario de start.spring.io para la creación del proyecto restful-blog-eureka.....	111
Ilustración 73 Configuración de la clase RestfulBlogEurekaApplication .....	112
Ilustración 74 Modificación del archivo application.yml del proyecto restful-blog-eureka.....	112
Ilustración 75 Configuración de cliente eureka en los archivos application.yml.....	113
Ilustración 76 Dependencia de cliente de eureka necesaria en el pom.xml de cada microservicio.....	113
Ilustración 77 Reemplazo de urls por alias de eureka en la clase CredentialsClientService .....	113
Ilustración 78 Reemplazo de urls por alias de eureka en la clase PostsClientService	114
Ilustración 79 Uso de variables de sistema para nombres de microservicios en la clase AuthorClient .....	114
Ilustración 80 Uso de variables de sistema para nombres de microservicios en la clase CredentialClient.....	115
Ilustración 81 Formulario de start.spring.io para la creación del proyecto restful-blog-sidecar .....	116
Ilustración 82 Archivo pom.xml del proyecto restful-blog-sidecar .....	118
Ilustración 83 Clase RestfulBlogSidecarApplication del proyecto restful-blog-sidecar .....	118
Ilustración 84 Archivo bootstrap.yml del proyecto restful-blog-sidecar .....	119
Ilustración 85 Archivo application.yml del proyecto restful-blog-sidecar.....	119
Ilustración 86 Interfáz gráfica del servicio eureka .....	120

Ilustración 87 Formulario de start.spring.io para la creación del proyecto restful-blog-configuration .....	121
Ilustración 88 Clase RestfulBlogConfigurationApplication del proyecto restful-blog-configuration .....	122
Ilustración 89 Formulario de creación de usuarios de gitstack .....	122
Ilustración 90 Formulario de creación de repositorio de gitstack.....	123
Ilustración 91 Formulario de asignación de usuario a repositorios en gitstack .....	123
Ilustración 92 Contenidos de archivos de propiedades del proyecto restful-blog-configuration .....	124
Ilustración 93 Comandos git para hacer push en un reoositorio remoto .....	124
Ilustración 94 Archivo bootstrap.yml del proyecto restful-blog-configuration.....	125
Ilustración 95 Modifcación de clase UserDetailsServiceImpl para la utilización de propieades.....	125
Ilustración 96 Modifcación de clase FeingClientConfiguration para la utilización de propieades existentes en spring cloud config .....	126
Ilustración 97 Clase ServiceAuthUser del proyecto restful-blog .....	127
Ilustración 98 Clase SpringCloudConfigClient del proyecto restful-blog.....	128
Ilustración 99 Inyección de la clase SpringCloudConfigClient en la clase UserDetailsServiceImpl de restful-blog .....	128
Ilustración 100 Inyección de la clase SpringCloudConfigClient en la clase AuthorClient de restful-blog .....	129
Ilustración 101 Inyección de la clase SpringCloudConfigClient en la clase CredentialClient de restful-blog.....	130
Ilustración 102 Verificación de la exposición de las propiedades en el servicio de configuración .....	130
Ilustración 103 Dependencias lo4j2 para proyectos restful-blog-credentials y restflu-blog-author.....	132
Ilustración 104 Archivo de configuración de log4j2.xml para los proyectos restful-blog-credentials y restful-blog-authors .....	133
Ilustración 105 Formulario para la creación del proyecto restful-blog-log.....	134
Ilustración 106 Archivo pom.xml del proyecto restful-blog-log.....	135

Ilustración 107 Clase RestfulBlogLogApplication del proyecto restful-blog-log...	136
Ilustración 108 Receiver del proyecto restful-blog-log .....	137
Ilustración 109 Clase LogRepository del proyecto restful-blog-log.....	137
Ilustración 110 Script de tabla log para la base de datos restful_blog_log .....	138
Ilustración 111 Filas sql generadas en el momento en que se ejecuta el llamado al servicio authors .....	138
Ilustración 112 Dependencia a adicionar en el archivo pom.xml de microservicios credentials y authors para soportar cortocircuitajes .....	139
Ilustración 113 Modificación de la clase CredentialsService en el proyecto restful-blog-credentials para soporte de cortocircuitajes en el proyecto restful-blog-authors.....	139
Ilustración 114 Modificación de la clase AuthorService en el proyecto restful-blog-credentials para soporte de cortocircuitajes en proyecto restful-blog-authors .....	140
Ilustración 115 Formulario de start.spring.io para la creación del proyecto restful-blog-monitor .....	141
Ilustración 116 Clase RestfulBlogMonitorApplication del proyecto restful-blog-monitor .....	142
Ilustración 117 Archivo bootstrap.yml del proyecto restful-blog-monitor .....	142
Ilustración 118 Archivo application.yml del proyecto restful-blog-monitor.....	142
Ilustración 119 Página principal del microservicio hystrix .....	143
Ilustración 120 Estadísticas en hystrix del cluster CREDENTIALS .....	144
Ilustración 121 Estadísticas en hystrix del cluster AUTHOR.....	145
Ilustración 122 Clase PostService del proyecto restful-blog-posts.....	147
Ilustración 123 Clase PostController del proyecto restful-blog-posts.....	148
Ilustración 124 Clase PostRepository del proyecto restful-blog-posts.....	149
Ilustración 125 Formulario de start.spring.io para la creación dek microservicio restful-blog-posts .....	150
Ilustración 126 Archivo pom.xml del proyecto restful-blog-posts .....	153
Ilustración 127 Clase RestfulBlogPostsApplication del proyecto restful-blog-posts.....	155
Ilustración 128 Clase PostService del proyecto restful-blog-posts .....	157
Ilustración 129 Clase PostRepository del proyecto restful-blog-posts.....	157
Ilustración 130 Clase CategoryClientService del proyecto restful-blog-posts .....	158

Ilustración 131 Clase AuthorClientService del proyecto restful-blog-posts .....	158
Ilustración 132 Clase CredentialsClientService del proyecto restful-blog-posts.....	158
Ilustración 133 Archivo bootstrap.yml del proyecto restful-blog-posts .....	159
Ilustración 134 Archivo application.yml del proyecto restful-blog-posts .....	159
Ilustración 135 Archivo de configuración restful-blog-monitor incluyendo post....	160
Ilustración 136 Script sql de tabla post para base de datos restful_blog_post.....	160
Ilustración 137 Valores para la tabla post para base de datos restful_blog_post.....	161
Ilustración 138 Clase Category del proyecto restful-blog-monolith, remoción de etiqueta ManyToMany y adición de lista de objetos tipo PostCategory .....	161
Ilustración 139 Clase PostCategory del proyecto restful-blog .....	162
Ilustración 140 PostClient del proyecto restful-blog .....	163
Ilustración 141 CategoryService del proyecto restful-blog.....	164
Ilustración 142 CategoryController del proyecto restful-blog .....	164
Ilustración 143 Clase CategoryService existente en el monolito .....	166
Ilustración 144 Clase CategoryController del proyecto restful-blog .....	167
Ilustración 145 Clase Category existente en el proyecto restful-blog .....	168
Ilustración 146 Clase CategoryRepository del proyecto restful-blog .....	168
Ilustración 147 Formulario de start.spring.io para la creación del proyecto restful-blog- categories.....	170
Ilustración 148 Archivo pom.xml del proyecto restful-blog-categories.....	173
Ilustración 149 Clase RestfulBlogCategoriesApplication del proyecto restful-blog- categories.....	175
Ilustración 150 Clase CategoryService del proyecto restful-blog-category .....	176
Ilustración 151 Clase CategoryRepository del proyecto restful-blog-category .....	177
Ilustración 152 Clase PostCategory transcrita al proyecto restful-blog-categories...	178
Ilustración 153 Clase Category transcrita al proyecto restful-blog-categories .....	179
Ilustración 154 Cliente del microservicio credentials en el proyecto restful-blog- categories.....	180
Ilustración 155 Cliente del microservicio posts en el proyecto restful-blog-categories	180
Ilustración 156 Archivo bootstrap.yml del proyecto restful-blog-categories .....	181

Ilustración 157 Archivo application.yml del proyecto restful-blog-categories .....	181
Ilustración 158 Archivo application.yml del proyecto restful-blog-monitor .....	182
Ilustración 159 Scripts tablas category y post_category del microservicio categories .....	182
Ilustración 160 Scripts valores tablas category y post_category del microservicio categories.....	183
Ilustración 161 Clase RabbitMQMessage como modelo para utilizar la comunicación de mensajes entre microservicios .....	184
Ilustración 162 Fragmento de la clase AuthorService que incluye llamados por mensajería al servidor RabbitMQ .....	186
Ilustración 163 Clase RabbitMQClient en el proyecto restful-blog-credentials .....	187
Ilustración 164 Clase Receiver del proyecto restful-blog-credentials.....	188
Ilustración 165 Fragmento de la clase PostService del proyecto restful-blog-post... 188	188
Ilustración 166 Clase RabbitMQClient del proyecto restful-blog-categories .....	189
Ilustración 167 Clase Receiver del proyecto restftul-blog-categories.....	190
Ilustración 168 Plantilla de archivo DockerFile para los microservicios .....	192
Ilustración 169 Archivo docker-compose.yml para la ejecución de los microservicios .....	196
Ilustración 170 Inclusión de perfil docker en el archivo application.yml del proyecto restful-blog-eureka .....	197
Ilustración 171 Perfil docker en archivo application.yml del proyecto restful-blog-configuration .....	198
Ilustración 172 Inclusión de perfil docker en el archivo application.yml para los microservicios log, credential,author, post y category .....	198
Ilustración 173 Inclusión de perfil docker en el archivo bootstrap.yml para los microservicios log, credential,author, post y category .....	199
Ilustración 174 Comandos utilizados para ejecutar bases de datos por medio de docker-compose.....	199
Ilustración 175 Comandos git usados para la inicialización de un repositorio remoto .....	203
Ilustración 176 Repositorios dev,test,qa y prod en nexus .....	204

Ilustración 177 Creación de tarea en jenkins tipo build : definición de nombre.....	205
Ilustración 178 Creación de tarea en jenkins tipo build: asociación de repositorio git .....	205
Ilustración 179 Creación de tarea en jenkins tipo build: disparador de ejecuciones de repositorio git .....	206
Ilustración 180 Creación de tarea en jenkins tipo build: Ejecución de tareas tipo maven .....	206
Ilustración 181 Creación de tarea en jenkins tipo build: Administrador de repositorios nexus .....	207
Ilustración 182 Creación de tarea en jenkins tipo promoción: Definición de nombre de tarea .....	208
Ilustración 183 Creación de tarea en jenkins tipo promoción: Definición de lista desplegable de entornos .....	208
Ilustración 184 Creación de tarea en jenkins tipo promoción: Definición de propiedades de contenido .....	209
Ilustración 185 Tarea jenkins tipo promoción: Script de promoción de artefactos ...	209
Ilustración 186 Tarea restful-blog-author-build antes de ser ejecutada.....	210
Ilustración 187 Tarea restful-blog-author-build después de ser ejecutada.....	210
Ilustración 188 Artefacto restful-blog-author con versión 1.0.9 en el repositorio dev de nexus .....	211
Ilustración 189 Artefacto restful-blog-author con versión 1.0.9 en el repositorio test de nexus .....	211
Ilustración 190 Artefacto restful-blog-author con versión 1.0.9 en el repositorio qa de nexus .....	212
Ilustración 191 Artefacto restful-blog-author con versión 1.0.9 en el repositorio prod de nexus .....	212
Ilustración 192 Creación de nuevo proyecto en rundeck .....	213
Ilustración 193 Navegación para crear nuevo trabajo en rundeck .....	214
Ilustración 194 Asignación de nombre al nuevo trabajo .....	215
Ilustración 195 Creación del parámetro versión y asignación de url de servicio rest trabajo restful-blog-deploy en rundeck .....	215

Ilustración 196 Creación de parámetro application el el trabajo restful-blog-deploy en rundeck.....	216
Ilustración 197 Creación de parámetro enviroment el el trabajo restful-blog-deploy en rundeck.....	216
Ilustración 198 Referencia del script restful-blog-sh para ser ejecutado .....	217
Ilustración 199 Contenido del script restful-blog.sh .....	217
Ilustración 200 Carpetas utilizadas por restful-blog.sh para la simulación de los despliegues.....	218
Ilustración 201 Carpeta base para instalación de apache tomcat.....	219
Ilustración 202 Archivo de configuración de servidor tomcat, tomcat-users.xml ....	220
Ilustración 203 Autenticación de interfaz administrativa de apache tomcat .....	220
Ilustración 204 Interfaz administrativa de apache tomcat .....	221
Ilustración 205 Ejemplo de despliegue en apache tomcat de restful-blog .....	221
Ilustración 206 Prueba de ejecución de aplicación restful-blog desplegada en apache tomcat .....	222
Ilustración 207 Asistente de instalación mysql: Escojer el tipo de instalación desarrollador .....	223
Ilustración 208 Asistente de instalación mysql: Componentes a instalar para el perfil desarrollador .....	223
Ilustración 209 Asistente de instalación mysql: Lista de chequeo después de la instalación de mysql.....	224
Ilustración 210 Asistente de instalación mysql: Configuración de red del servidor mysql .....	224
Ilustración 211 Asistente de instalación mysql: Asignación de contraseña de administrador .....	225
Ilustración 212 Asistente de instalación mysql: Configuración de servicio web de windows .....	225
Ilustración 213 Configuración de conexión a servidor mysql desde wokbench .....	226
Ilustración 214 Asistente de instalación de gitstack: escojencia de la ruta .....	227
Ilustración 215 Asistente de instalación de gitstack: programas a instalar .....	227
Ilustración 216 Ventana de inicio de sesión de gitstack .....	228

Ilustración 217 Creación de usuarios en gitstack .....	229
Ilustración 218 Asistente de instalación erlang: Componentes a instalar .....	230
Ilustración 219 Asistente de instalación erlang: escojencia runta instalación .....	231
Ilustración 220 Asistente de instalación rabbitMQ: componentes a instalar .....	232
Ilustración 221 Asistente de instalación rabbitMQ: ruta de instalación .....	232
Ilustración 222 Comandos necesarios para configurar rabbitMQ .....	233
Ilustración 223 Ventana de inicio de sesión de rabbitMQ .....	233
Ilustración 224 Panel de control de RabbitMQ .....	234
Ilustración 225 Asistente de instalación de docker toolbox .....	235
Ilustración 226 Consola de docker usando docker toolbox .....	235
Ilustración 227 Panel de control de Oracle virtual box para docker .....	236
Ilustración 228 Ajuste de la memoria ram del Oracle Virtualbox para docker .....	237
Ilustración 229 Comando para la ejecución de jenkins desde línea de comandos ....	238
Ilustración 230 Pantalla inicial para configurar jenkins por primera vez .....	238
Ilustración 231 Escojencia de plugins para instalar en jenkins .....	239
Ilustración 232 Instalación de plugins en jenkins .....	239
Ilustración 233 Creación de primer usuario de jenkins.....	240
Ilustración 234 Pantalla principal de jenkins .....	240
Ilustración 235 Creación de credenciales de nexus en jenkins .....	241
Ilustración 236 Creación de credenciales de git para jenkins .....	241
Ilustración 237 Instalación de plugin inyector de propiedades y nexus desde el administrador de jenkins .....	242
Ilustración 238 Parametrización de variables globales de nexus en jenkins .....	242
Ilustración 239 Configuración de plugin nexus jenkins.....	243
Ilustración 240 Ventana de inicio de sesión de nexus; <b>Error! Marcador no definido.</b>	
Ilustración 241 Explorador de repositorio de nexus .....	244
Ilustración 242 Verificación de plugines instalados en nexus .....	245
Ilustración 243 Comando para ejecutar rundeck desde líne de comandos.....	246
Ilustración 244 Ventana de inicio de sesión de rundeck .....	246
Ilustración 245 Ventana principal de rundeck .....	247

## **Índice de Tablas**

Tabla 1 Resultados de la inspección del código fuente del contexto credenciales en el monolito .....	77
Tabla 2 Resultados de la inspección del código fuente del contexto author en el monolito .....	93
Tabla 3 Resultados de inspección de código del contexto publicaciones .....	146
Tabla 4 Resultados de la inspección del código fuente del contexto folcsonomía en el monolito .....	165
Tabla 5 Ambientes de despliegue de la aplicación.....	202
Tabla 6 Repositorios por proyectos necesarios para la entrega continua .....	202

## **RESUMEN**

La presente investigación plantea la creación de una metodología para la transformación de aplicaciones monolíticas a una arquitectura distribuida basada en microservicios; de forma resumida la metodología se organiza en 3 grandes bloques:

- Análisis del modelo del negocio de la aplicación monolítica a transformar.
- Fraccionamiento del monolito y construcción de los microservicios.
- Integración y entrega continua de los microservicios.

La investigación también incluye un caso de prueba que transforma una aplicación monolítica, paso a paso, de tal manera que sirva de ejemplo para llevar acabo transformaciones de aplicaciones monolíticas a una arquitectura basada en microservicios.

## **INTRODUCCIÓN**

El desarrollo ágil de software tiene como propósito promover un proceso de construcción más flexible y adaptativo a las necesidades del cliente, orientado a entregas las cuales están sujetas a ajustes durante cada fase. Desde la declaración del manifiesto ágil han surgido diversas metodologías que se han popularizado en los últimos años tales como SCRUM y XP (Lainez,2014).

El patrón de arquitectura por microservicios es un paradigma surgido recientemente el cual propone que los sistemas de software se deben fragmentar de tal manera que mejoren su disponibilidad, faciliten el mantenimiento, las modificaciones y la escalabilidad cuando se necesiten aumentar los recursos para un mejor desempeño, acercándose así a la filosofía de las metodologías ágiles .

Se entiende como sistema de software monolítico, aquel programa informático que tiene todos sus componentes ejecutándose en un mismo contenedor de aplicaciones y su código fuente se encuentra almacenado en una misma base de código (Rosa, 2016). Las aplicaciones monolíticas de gran tamaño sufren de inconvenientes de disponibilidad, el mantenimiento y la escalabilidad se dificultan a medida que dicho sistema monolito crece. Así mismo las actualizaciones se vuelven más complejas porque a medida que aumentan la cantidad de líneas de código fuente de toda la aplicación los programadores van a necesitar de un mayor esfuerzo para nuevos desarrollos, de igual manera la escalabilidad se hace más costosa debido a que si la aplicación requiere crecer se deben de replicar todos los componentes de la aplicación por cada nueva instancia de software necesitada. Por lo tanto la arquitectura basada en microservicios propone romper una aplicación monolítica en pequeñas aplicaciones con propósitos o funcionalidades específicas de tal manera que lo que se necesite replicar sea lo necesario y que las aplicaciones sean más fáciles de mantener. Otra ventaja de la arquitectura por microservicios es su adaptabilidad y facilidad de respuesta rápida a las necesidades del cliente lo cual es un principio del manifiesto ágil.

El uso de malas prácticas en la construcción de software presenta diferentes inconvenientes tales como poca comunicación, rigidez a las cambiantes necesidades de los clientes, incumplimientos en el cronograma y sobrecostos durante la ejecución. El resultado es un producto que puede ser tardío, costoso, de baja calidad o no funcional.

## **PLANTEAMIENTO DEL PROBLEMA**

Pregunta de investigación: ¿De qué manera la transformación de una arquitectura monolítica a microservicios puede ayudar el desarrollo de la aplicación web?

La arquitectura monolítica ha sido el diseño tradicionalmente utilizado para el desarrollo de aplicaciones web desde sus principios, la cual es construida por uno o varios equipos de desarrollos, encargados de trabajar un diseño centralizado, de rápido crecimiento y expuesto a cambios permanentes. A medida que las aplicaciones monolíticas crecen están son susceptibles al desarrollo de anti patrones de diseño que la convierten en sistemas complejos y difíciles de mantener, incrementando no solo los tiempos de mantenimiento sino también los costos de escalabilidad que representan la duplicación de instancias del monolito.

Por lo anterior, se propone construir una metodología que sirva de guía para que los equipos de desarrollo o mantenimiento de aplicaciones monolíticas puedan hacer uso de recomendaciones y buenas prácticas que permitan llevar acabo transformaciones evolutivas y planificadas de las aplicaciones web.

## **JUSTIFICACIÓN**

Una metodología que permita transformar aplicaciones monolíticas a microservicios ofrece un marco de trabajo para un equipo de desarrollo donde se puede estructurar, planificar y controlar el proceso de transformación hacia una arquitectura distribuida. Una metodología con este fin deberá estar en capacidad de ayudar a evaluar la factibilidad de una transformación a microservicios y de analizar la fragmentación del código para hacerlo de manera incremental y confiable, otro fin importante de la metodología es ayudar a lograr que el monolito pueda ser escalable desde el aspecto técnico por medio de la fragmentación. De igual manera la metodología permitirá que los coordinadores de los equipos de desarrollo puedan realizar un mejor seguimiento de la integración continua y los despliegues en producción de los microservicio al igual que lograr ser más asertivos en la asignación de actividades de acuerdo a las experticias y trabajo previo de cada persona.

## HIPÓTESIS

La transformación de una arquitectura monolítica a microservicios puede ayudar el desarrollo de una aplicación web de la siguiente manera:

- Semejanza entre la estructura de la organización y la aplicación web con base en la definición de contextos de negocio.

**Indicador:** Cada microservicio debe tener un contexto de negocio equivalente

- Fragmentando el código fuente legado haciéndolo más legible a los desarrolladores.

**Indicador:** El código fuente debe de fragmentarse en diferentes proyectos de código y diferentes bases de código fuente .

- Disminuyendo el acoplamiento entre los componentes de la aplicación, facilitando la escalabilidad de cada uno manera independiente.

**Indicador:** Cada microservicio debe tener un funcionamiento autónomo y debe poderse desplegar independientemente de los demás componentes de software.

- Ofreciendo tareas automatizadas de integración y de entrega continua.

**Indicador:** Las tareas de compilación, pruebas y despliegues deben de hacerse automáticamente ante cualquier cambio en las bases de código.

- Previniendo fallos de sistema no manejados y seguimiento de errores por medio de estadísticas de monitoreo en cada microservicio

**Indicador:** Cada microservicio deberá incluir lógica propia que permita manejar errores de ejecución en sus llamados al igual que estadísticas de red sobre su utilización remota.

## ESTADO DEL ARTE

Para comenzar a hablar sobre el origen de los microservicios hay que remontarse a los inicios de SOA en el año de 1994 cuando Alexander Pasik un antiguo analista de Gartner acuñó el término *SOA* antes de que los servicios *XML* y los servicios web fueran inventados, Pasik creó este nuevo concepto para referirse a un nuevo tipo de arquitectura cliente servidor que había perdido su sentido clásico para referirse a un nuevo tipo de computación distribuida (Josuttis,2007). De acuerdo con (Wolff ,2016) el término servicio en *SOA* debería de tener las siguientes características:

- Debería de poder implementarse como una pieza individual del dominio
- Debería poderse usar independientemente
- Debería estar disponible en la red
- Cada servicio debería de tener una interfaz, acceder a la interfaz debería ser suficiente para utilizar el servicio.
- El servicio debería poder ser utilizado en diferentes lenguajes de programación y plataformas.
- Para facilitar su uso, el servicio debe de estar registrado en un directorio. Para localizarlo y usar el servicio, los clientes buscan este directorio en tiempo de ejecución
- El servicio debería de ser de grano fino, de tal manera que permita reducir las dependencias. Los servicios pequeños pueden implementar funcionalidad de gran utilidad solamente en conjunción con otros servicios. Sin embargo SOA tradicionalmente se enfoca en grandes servicios

Fue a partir de los preceptos de *SOA* que grupos de arquitectos de software y varias compañías que decidieron realizar sus propias implementaciones de *SOA*. Un caso muy conocido es Amazon que decidieron cerca del año 2000 convertir toda su infraestructura interna a servicios distribuidos (Miller, 2016) hasta ofrecer en el año 2006 su famoso *Amazon Elastic Compute Cloud*. Netflix por su parte era una compañía compuesta por 100 ingenieros

que trabajaban sobre una aplicación monolítica de rentaba *DVD* a llegar a transformar a una arquitectura microservicios compuesta de grupos pequeños y que trabajan en conjunto para ofrecer el servicio de streaming a millones de clientes cada día (Mauro,2015), posteriormente otras compañías como uber, airbnb, orbitz, ebay, gilt, twitter, nike (RV,2016) siguieron el ejemplo transformando sus monolitos a una arquitectura basada en microservicios.

Sin embargo (Krueger,2015) plantea que mientras *SOA* se enfoca en integrar sistemas, microservicios se enfoca en un sistema individualmente por medio de coreografía, cada servicio es autónomo en su comportamiento, y no en orquestación como lo planteaba inicialmente *SOA*, mientras *SOA* proveía un sistema inteligente, *ESB*<sup>1</sup>, para conectar aplicaciones, microservicios se basa en diseñar sistemas inteligentes y desacoplados entre ellos. De acuerdo con (Stenberg,2014) durante la conferencia GOTO Berlín Martin Fowler planteó las características más comunes encontradas en los sistemas microservicios como una extensión de los principios planteados por *SOA*:

- Componentización, la habilidad de remplazar partes del sistema, haciendo un símil con los dispositivos con sonido estéreo donde cada pieza puede ser reemplazada independientemente de las otras.
- Organización con base en núcleo del negocio y no basada en la tecnología.
- Inteligentes y comunicados por conectores simples, explícitamente rechazan el uso de un ESB.
- Manejo de base de datos descentralizado con una sola base de datos para cada servicio en vez de una sola para toda la compañía.
- Automatización de infraestructura con entrega continua es mandatoria

<sup>1</sup> Enterprise Service Bus: Bus de servicio empresarial

## MARCO TEÓRICO

### Definición de microservicios

De acuerdo con James Lewis y Martin Fowler (Fowler ,2014): “*el termino microservicios fue discutido en un taller de arquitectos de software cerca a Venecia en Mayo del 2011 para describir un nuevo estilo de arquitectura común que ellos habían recientemente explorado. En mayo del 2012 el mismo grupo decidió escoger microservicios como el nombre más apropiado para esta naciente arquitectura.*”

Según (Newman, 2015) los microservicios son componentes de software que permiten modularizar un sistema, cada uno con un propósito diferente y específico (p.2). deben ser lo suficientemente pequeños de tal manera que sean fáciles de mantener . Newman sugiere que los microservicios deben de ser guiados por los límites del dominio haciendo uso del patrón Model Driven Design el cual fue acuñado por (Evans, 2004) y que se define como un enfoque, con necesidades complejas, que se fundamenta entre una profunda conexión entre la implementación del software y el modelo del núcleo del negocio, el cual provee prácticas y terminologías para la toma de decisiones que tienen como propósito acelerar proyectos de software con modelos complejos (Evans & Gitlevich, 2007)

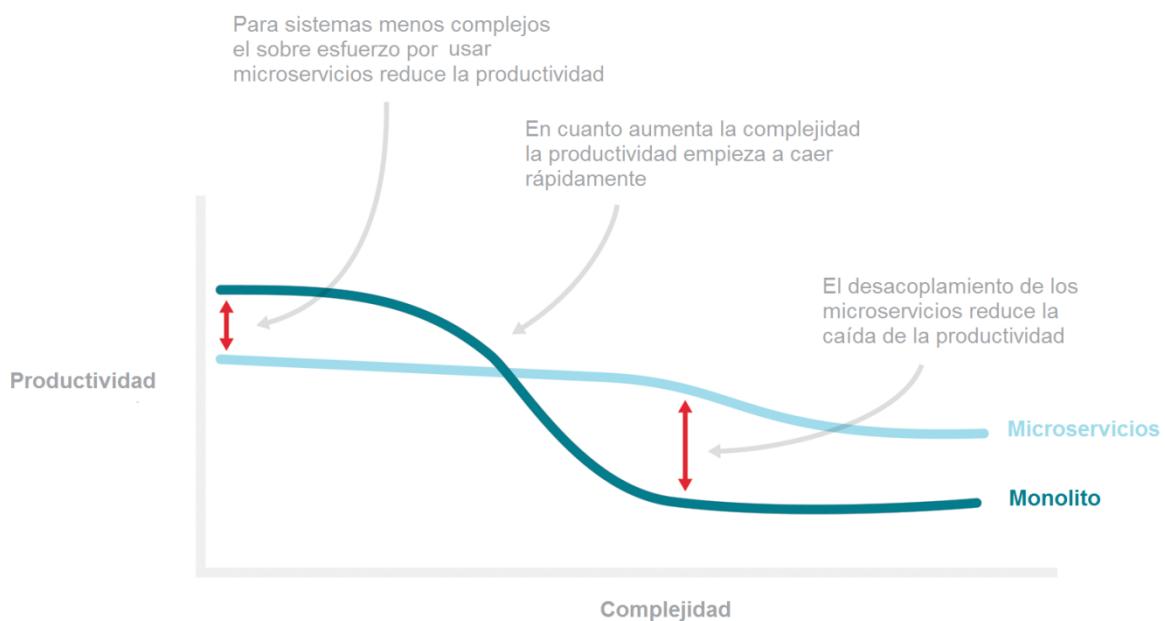
Los microservicios deben de funcionar como entidades separadas y autónomas, su diseño debe seguir los lineamientos de alta cohesión y bajo acoplamiento, para ello un microservicio no deberá de exponer toda su funcionamiento ya que aumenta la tendencia al acoplamiento entre los consumidores dificultando el mantenimiento de la aplicación cuando se desea hacer una actualización (Newman, 2015,p.3)

### **Ventajas de los microservicios**

- La gobernanza centralizada establece un estándar para construir una plataforma con la misma tecnología, pero la arquitectura microservicios permite hacer uso de una gobernanza descentralizada facilitando el libre uso de tecnologías para la construcción de cada servicio (p.4)
- Capacidad de auto controlar fallos del sistema, a diferencia de los monolitos que necesitan instancias completas de la aplicación (p.5) el sistema basado en microservicios se puede mantener disponible sin presentar un fallo general.
- Son pequeños y fáciles de mantener ,el escalamiento de los microservicios se hace con los servicios que se necesitan sobre la demanda reduciendo costos de disponibilidad (p.5)
- El despliegue es menos tedioso que el de una aplicación monolítica ya que se hacen de manera aislada, y en caso de fallo el *rollback*<sup>2</sup>es más simple de hacer reduciendo el impacto negativo que tendría una aplicación tradicional mal desplegada (p.6)

<sup>2</sup> Rollback: Reversión

- El uso de microservicios permite alinear mejor la arquitectura con la organización ayudando a optimizar la cantidad de personas por módulo y la productividad (p.7) , en la siguiente ilustración se puede encontrar un comparativo entre un software monolito y otro basado en microservicios comparando las variables productividad y complejidad, deduciéndose que un monolito se vuelve menos productivo de mantener en la medida que se hace más complejo, mientras que uno basado en microservicios es más productivo de mantener en la medida que se aumente su complejidad.



**Ilustración 1. La productividad vs la complejidad en los microservicios. (Fowler 2015)**

- La sustitución se ve optimizada puesto que los sistemas legados pueden ser muy costosos de mantener o borrar, en tanto que los sistemas basados en microservicios son más económicos de actualizar o inclusive borrar ya que su modificación es pequeña y aislada. (p.8)

### **Desventajas de los microservicios**

- Microservicios no es una solución recomendada a ser implementable en todos los sistemas monolíticos ya que la construcción de los servicios incluyen toda la complejidad de los sistemas distribuidos y son difíciles de implementar (p.11)
- Los micro servicios ofrecen cierta capacidad de autonomía y de decisión a los trabajadores, en caso que la organización tenga estructuras rígidas el uso de microservicios no debería ser una opción (p.24)

En tanto los sistemas monolíticos presentan un alta acoplamiento en sus componentes lo cual produce alta dependencia tecnológica para realización de mejoras, baja sensibilidad a fallos, altos costos de escalabilidad, despliegues más riesgosos y dificulta labores de mantenimiento a los técnicos. (p.4)

## Modelado de los servicios

Para la creación de una correcta definición de un servicio (Newman, 2015) sugiere enfocarse en dos conceptos: Alta cohesión y bajo acoplamiento. Un servicio con bajo acoplamiento es aquel que puede ser modificado y desplegado sin necesidad de cambiar ninguna otra parte en el sistema. Un servicio con alta cohesión es aquel que tiene un comportamiento único en el sistema.(p.30)

(Evans, 2004) planteó por primera vez el concepto de *bounded context*<sup>3</sup> los cuales son agrupaciones dentro un dominio que contienen modelos, cada *bounded context* tiene una interfaz explícita donde se especifica que modelos se pueden compartir y cuales no (p.69)

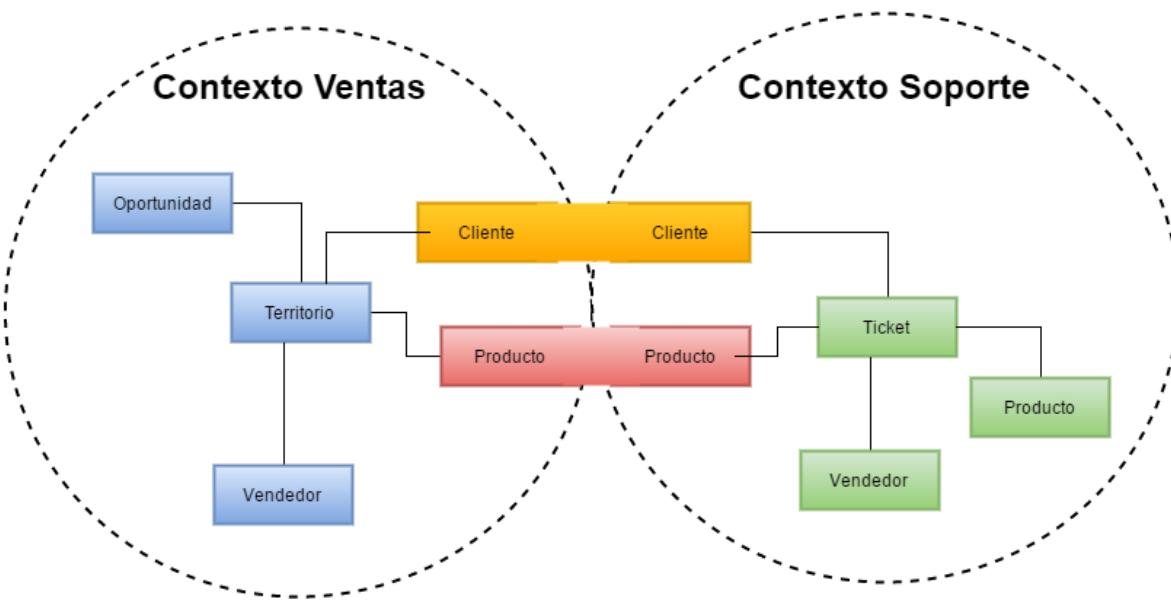


Ilustración 2 Ejemplo de bounded contexts. (Fowler, M. 2014)

De acuerdo con (Newman, 2015) es importante identificar los *bounded context* de el dominio, definir adecuadamente las interfaces para evitar acoplamientos no deseados. Los *bounded context* deberán ser modelados en el código como módulos los cuales a su vez se califican como candidatos a ser definidos como microservicios.

<sup>3</sup> Bounded context: Fragmento de modelo de negocio delimitado, consistente e implementable

Antes de iniciar una aplicación basada en microservicios el autor recomienda tener clara la información del dominio, definir los *bounded context* e interfaces para evitar una inadecuada reutilización de un módulo que no haya sido lo suficientemente generalizado necesitando que volverlo monolito para redefinir los límites de cada dominio (p.34)

## Desacoplamiento de artefactos

(Newman, 2015) define que los microservicios deben ocultar implementación interna y ofrecer un API agnóstico, es decir independientes a cualquier proceso de negocio concreto ,de tal manera que se disminuya el acoplamiento y se simplifique la comunicación con los consumidores . Debido a un sistema microservicios tiene una estructura distribuida la comunicación asíncrona permite bajar la latencia en la comunicación y evita el uso de canales dedicados mientras se realiza una transacción y la comunicación por eventos permite a los consumidores comunicarse a los microservicios por medio de suscripción sin necesidad que los clientes se enteren permitiendo adicionar microservicios dinámicamente.(p.43) A continuación se muestra un ejemplo de un microservicio que se comunica de manera síncrona (Ver **¡Error! La autoreferencia al marcador no es válida.**) y de manera asíncrona (Ver Ilustración 4).

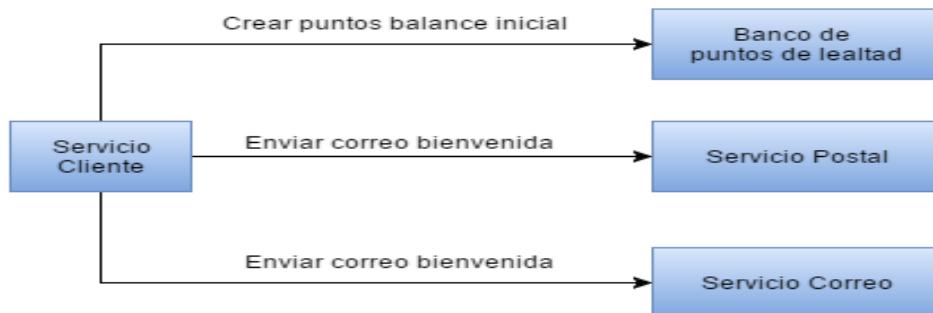
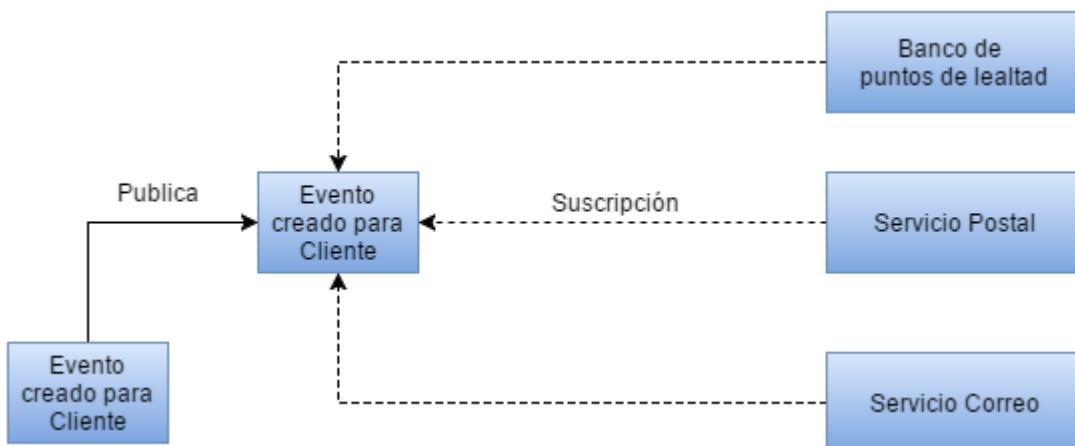


Ilustración 3. Manejando la creación de un cliente por medio de orquestación. (Newman, 2015, Figura 4-3)



**Ilustración 4 Manejando la creación del cliente por medio de coreografía. (Newman,2015,**

**Figura 4-4)**

### **Fragmentación por base de datos**

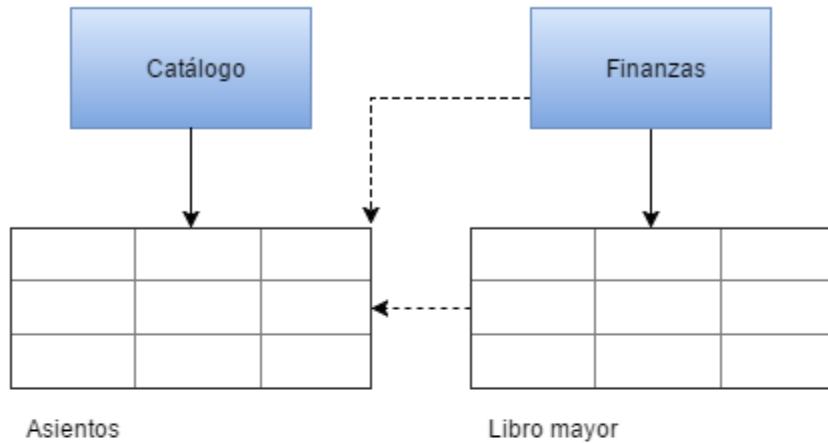
Luego de haber subdividido el dominio por medio de contextos delimitados o *bounded context*, se deben de crear paquetes en el código, representando los contextos y empezar a mover el código existente de manera incremental, por medio de un *IDE*<sup>4</sup> que facilite esta labor. En caso que quede código suelto en este proceso debe de ser analizado para empaquetarlo dentro de algún contexto.(p.80)

La base de datos también está sujeta a particionamiento debido al fuerte acoplamiento que tiene en un sistema monolítico, el código responsable de hacer el mapeo deberá de ser organizado de igual manera en los *bounded context*, y los esquemas de las base datos deberán ser analizado para verificar el acoplamiento entre las tablas y delimitar los límites de cada servicio.(p.82)

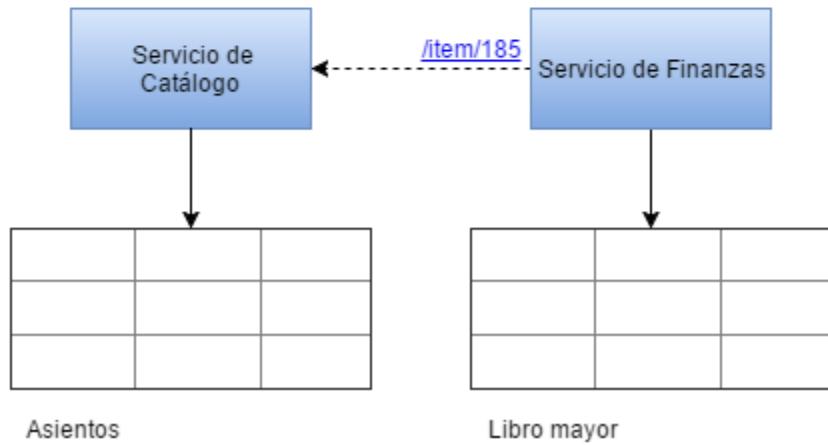
Cada *bounded context* deberá de tener su propio esquema de base de datos asociado, en caso que un microservicio necesite acceder una tabla fuera de sus límites esta deberá ser consultada a través de otro microservicio, en la Ilustración 5 se puede ver un ejemplo de una llave foránea en una aplicación monolítica y su equivalente en microservicios. (p.84)

<sup>4</sup> IDE: Integrated Development Environment

## Aplicación Monolítica

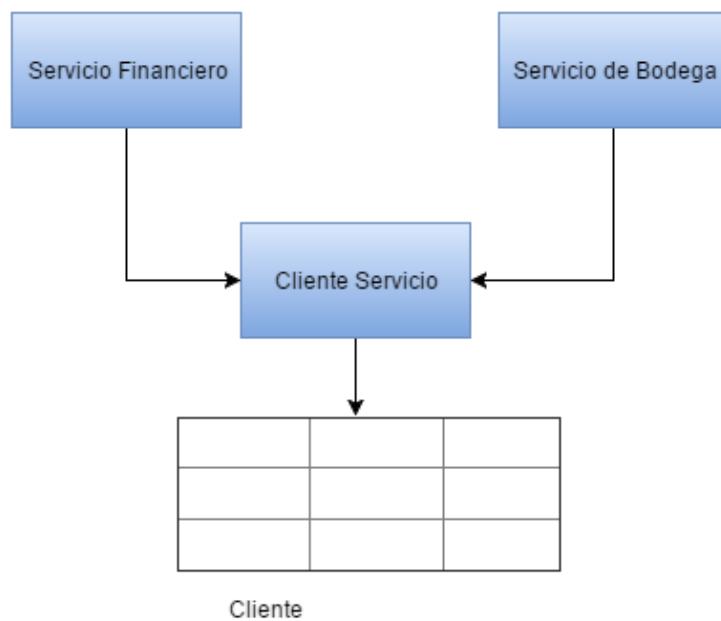
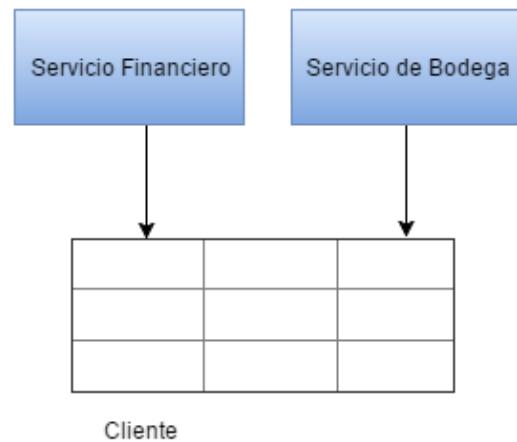


## Aplicación Microservicios



**Ilustración 5 Remoción de una llave foránea entre dos microservicios. (Newman,2015, Figura 5-2,Figura 5-3)**

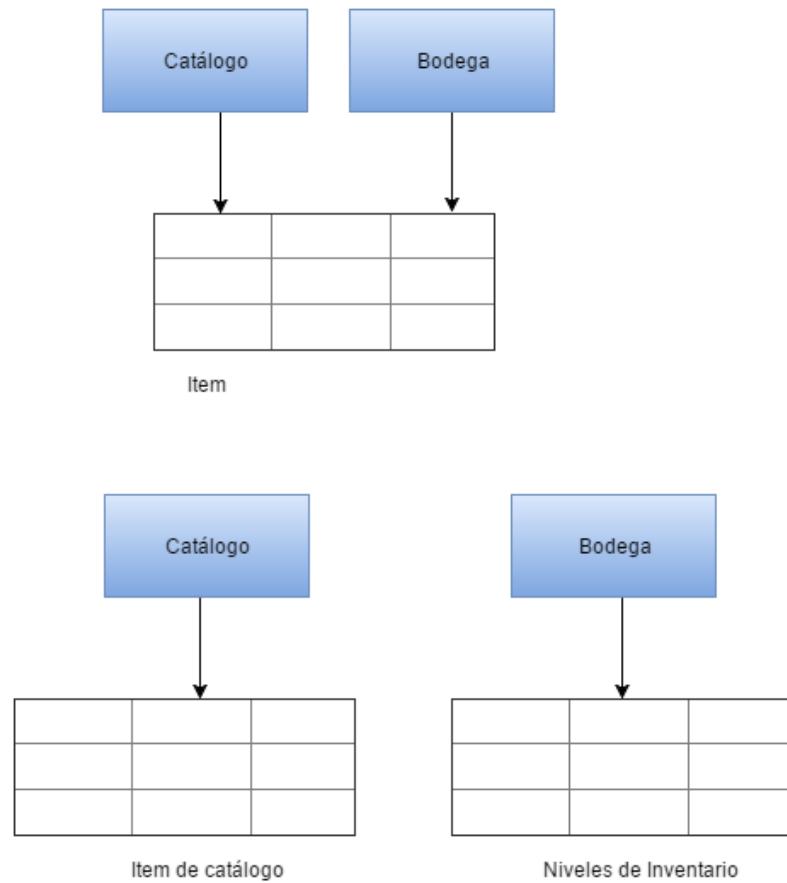
También se puede encontrar otro tipo de acoplamiento de datos donde dos componentes de un aplicativo monolítico intentan acceder a una tabla para escribir y leer, para solucionar este acoplamiento en un sistema microservicios será necesario crear un nuevo microservicio con el propósito de exponer la tabla al uso de múltiples microservicios ver Ilustración 6 (p.87)



**Ilustración 6 Creación de un nuevo bounded context para reutilización de datos.**

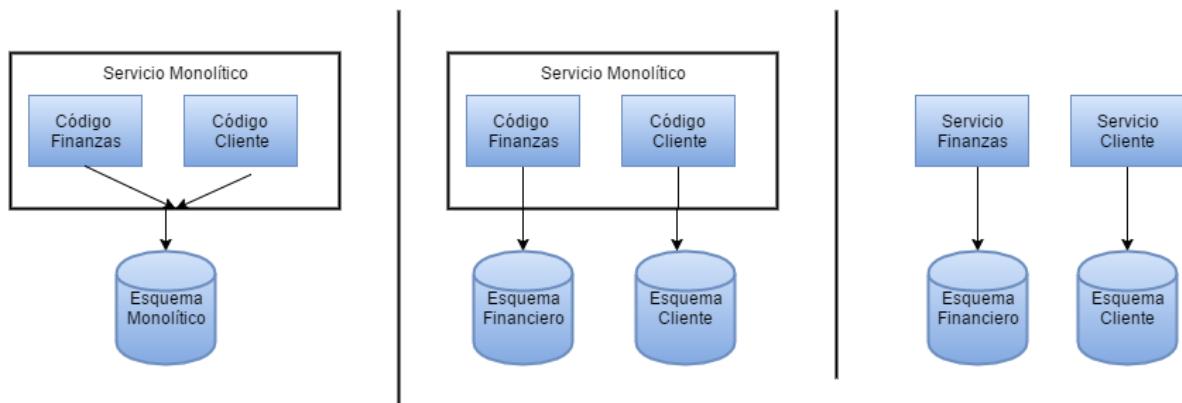
(Newman,2015,Figura 5-4)

Otro tipo de acoplamiento es por tabla compartida donde dos servicios intentan acceder a la misma tabla pero a escribir o consultar columnas diferentes para diferentes contextos, la solución para este acoplamiento consiste en separar la tabla en dos nuevas tablas de tal manera que cada una sea accedida por el microservicio correspondiente. (p.88)



**Ilustración 7 Descomposición de tabla para desacoplamiento entre contextos.**  
 (Newman,2015,Figura5-5)

Recomienda Newman que antes de iniciar la transformación de un servicio monolítico se aconseja comenzar con la división de los esquemas de base de datos antes dividir el servicio en microservicios (p.89) tal como se muestra en la Ilustración 8



**Ilustración 8 Etapas de la separación de un servicio.** (Newman 2015,Figura 5-9)

## El proceso scrum

De acuerdo con (Albaladejo,2008) *Scrum* es un marco de trabajo que permite trabajar colaborativamente y de manera productiva, permite realizar entregar parciales del producto al cliente permitiendo entregar resultados prontos y ofreciendo flexibilidad ante cualquier necesidad cambiante. *Scrum* ofrece capacidad de reacción frente a la competencia y da soluciones prontas a problemas en las entregas permitiendo así un desarrollo ágil.

Su ciclo de vida, el cual se denomina *sprint*, tiene una duración de 2 o 3 semanas por lo regular, cada iteración debe ofrecer un incremento al producto final, durante cada *sprint* se llevan acabo las siguientes actividades:

**Selección de historias de usuario:** El cliente presenta al equipo de desarrollo la lista de historia de usuario priorizada del producto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración.

**Planificación del sprint:** El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido, se hacen sesiones de estimación del esfuerzo entre los miembros del equipo y el cliente y se auto asignan las tareas.

**Reuniones diarias.** Cada día el equipo realiza una reunión corta con el propósito de que cada miembro del equipo se entere del trabajo que el resto está realizando el otro para servir de apoyo o enterarse de dependencias, tareas u obstáculos. Cada integrante del equipo debe dar un reporte diario respondiendo las siguientes preguntas:

Qué hice ayer?

Que voy a hacer en este momento?

Tengo algún bloqueo o impedimento para realizar mis labores?

El Facilitador de la reunión el (*Scrum Master*) se encarga de que el equipo pueda cumplir su compromiso y de que no se merme su productividad, además ayudar a eliminar los

obstáculos que el equipo no puede resolver por sí mismo y protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

**Demostración:** Al final del *sprint* el equipo presenta al cliente los requisitos completados en la iteración y tomará en cuenta sus observaciones para la siguiente fase.

**Retrospectiva:** El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad.

## **MARCO CONCEPTUAL**

A continuación se describe los conceptos producto del análisis del marco teórico que serán base para desarrollar la investigación

### **Planeamiento y factibilidad**

Se deberá por medio de reuniones con el cliente establecer el propósito del proyecto, el problema a solucionar, el alcance, contribución a los objetivos organizacionales, factibilidad tecnológica, presupuesto disponible, riesgos y fechas límite del proyecto. También se deberá establecer procedimientos de negociación ante eventuales cambios solicitados por el cliente (Cruz Cristian, Castro Gustavo, 2015).

En caso que el sistema necesite ser integrado a sistemas legados se deberá realizar un levantamiento de información de sistemas existentes tales como documentos de diseño, especificaciones de servicios web o protocolos de comunicación, diagramas de base de datos, manuales de usuario etc. De igual manera es importante conocer los integrantes del equipo del proyecto, ya que sus especialidades y experticias son importantes para saber que actividades en el proyecto son factibles de realizar.

Finalmente el cliente deberá recibir al final de esta etapa una presentación de una solución al problema planteado, ajustada al presupuesto del cliente el cual deberá incluir información de entregables, cronograma, costos detallados, recursos disponibles, metodologías de desarrollo a seguir, tecnologías y herramientas de desarrollo a utilizar.

## **Educción y análisis de requisitos**

Es necesario escoger cuáles técnicas para la captura de requerimientos son las más adecuadas a usar, involucra interacción con el cliente para un entendimiento del dominio de la aplicación, los servicios que el aplicativo debe proveer y las restricciones del mismo. Las técnicas son seleccionadas de acuerdo a las necesidades del cliente, tipos de *stakeholder*<sup>5</sup>, recursos y tiempo disponible para desarrollar la solución. Existen múltiples técnicas de educación tales como cuestionarios, entrevistas, talleres de trabajo, observación natural, estudio de documentación, etnografía, lluvia de ideas entre muchas más.

La aplicación de las técnicas de educación producirá una lista de requisitos los cuales serán administrados por medio de una matriz de trazabilidad e independencia la cual permitirá justificarlos de acuerdo a las necesidades origen. De igual manera los requisitos deberán ser clasificados de acuerdo a su naturaleza funcional o no funcional, podrán ser priorizados según el grado de importancia que tenga los *stakeholders*.<sup>6</sup>

## **Especificación de requisitos**

Los requisitos son representados en historias de usuario que describen de manera detallada las entradas, salidas, flujo, precondiciones, pos condiciones y restricciones del mismo.

Las historias de usuario pueden adjuntar diagramas para mejorar la comprensión del flujo del requisito, sus restricciones y cómo implementarlo más fácilmente. Para ello se dispone de 3 tipos de diagramas: el diagrama de clases permitirá representar relaciones que existen entre las clases del sistema, la estructura de datos y las operaciones existente en cada uno, el diagrama de actividad ofrece una panorámica del paso a pasos que debe ejecutar la historia de usuario y el diagrama de secuencia permitirá tener una perspectiva de cómo se

<sup>5</sup> Stakeholders: Persona interesadas en el software

comunica cronológicamente el requisito con el resto del sistema permitiendo entender como es el flujo ideal y los posibles errores que pueden ocurrir en caso de un fallo del sistema.

En caso que el requisito esté asociado a una ventana gráfica o requiera de usabilidad estos pueden incluir un prototipo gráfico que describa la manera como debe ser presentar la información, la navegación entre ventanas, la operatividad y comprensión del sistema.

### **Validación de requisitos**

Es necesario validar las definiciones del sistema frente a las expectativas del cliente, de esta manera se evitarán resultados indeseados en las entregas y sobrecostos en el proyecto. Para ello se debe realizar validaciones de los criterios de aceptabilidad, verificación de consistencia de la lógica, de completitud funciones y restricciones, validación de viabilidad técnica y de verificación de tal manera que los resultados de los requisitos sean verificables.

Siguiendo las anteriores validaciones se verificará que el sistema cumpla con las expectativas del cliente y una trazabilidad adecuada entre los resultados y las especificaciones solicitadas

### **Diseño y definición de microservicios**

Las historias de usuario deben ser descritos técnicamente en documentos de diseño donde se detalle el diseño de la base de datos, despliegue de las aplicaciones, protocolos de comunicación, recursos de red, valores de configuración entre otros. También es necesario definir cada uno de las interfaces de los microservicios de tal manera que la comunicación quede estandarizada entre ellos.

Los documentos de diseño deben de tener diagramas relacionales, de despliegue, y de clase. Para la realización de diagramas y descripción de los diseños se deberán tener en cuenta criterios como la separación de los servicios, autonomía, alta cohesión, bajo acoplamiento, servicios con estado propio, comunicación asincrónica y movilidad en la red

## **Desarrollo y pruebas de unidad**

Los programadores deben de codificar la lógica de negocio de cada requerimiento en el lenguaje de programación seleccionado y las pruebas de unidad de los microservicios donde se garantice que cumplen con los criterios de aceptación de cada requerimiento.

## **Pruebas de integración, contratos y punto a punto**

Los microservicios se distribuyen en contenedores los cuales deben de ser probados en diferentes por diferentes niveles de granularidad de tal manera que se garantice el correcto funcionamiento interno y externo del servicio. Diferentes tipos de pruebas pueden realizarse en un servicio que van desde las pruebas de unidad, el protocolos de comunicación, el esquemas de base de datos, los contratos entre las interfaces, las pruebas punto a punto y pruebas de rendimiento. (Cohn, 2010) plantea en un modelo piramidal que el mecanismo de pruebas debe estar mayormente compuesto de pruebas de unidad caracterizadas por su granularidad fina, rapidez y bajo costo además de otras en menor cantidad, pero no menos importantes, de pruebas de servicio y de interfaz gráfica que son de granularidad gruesa, pero que ofrecen mayor nivel de cobertura y más confiabilidad en el buen funcionamiento del sistema. A continuación se muestra una imagen que describe el modelo piramidal.



**Ilustración 9 Pirámide de pruebas de Cohn.(Newman 2015, Figura 7-2)**

### **Despliegue e integración continua**

Si todas las pruebas de desarrollo en los microservicios han pasado se procede a hacer despliegues en otros tipos de entornos para que tanto testers como clientes pueda verificar la funcionalidad de la misma hasta finalmente realizar el despliegue a producción. En producción se deberán de realizar pruebas de estabilidad continuas para producir fallos aleatorios en los componentes del sistema. Las interfaces gráficas serán monitoreadas de tal manera que se implementen futuros cambios en las mismas que disminuyan la cantidad de errores humanos en el uso del aplicativo.

## Diagrama de flujo de la etapas

A continuación se muestra un diagrama de flujo donde se resume el paso a paso de la cada una de las actividades anteriormente mencionadas necesarias para llevar a cabo una transformación de monolito a microservicios

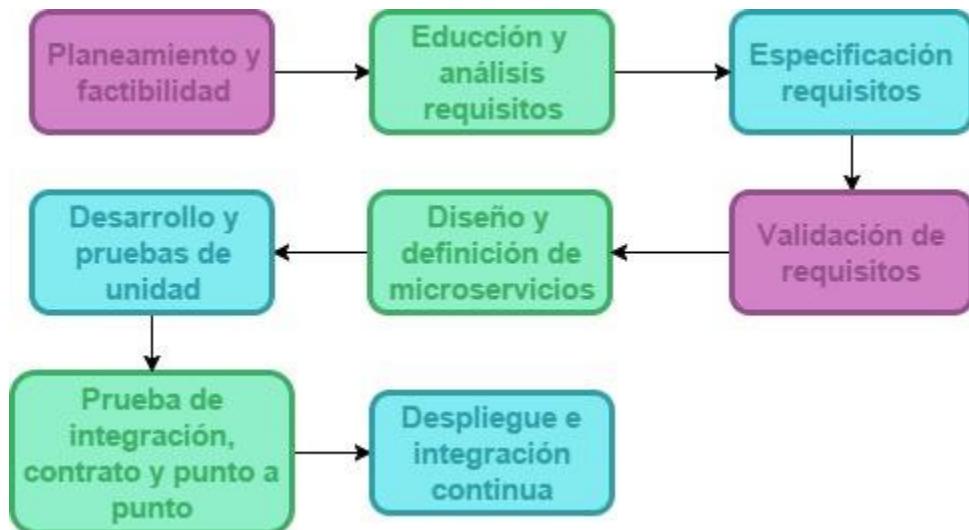


Ilustración 10 Diagrama de flujo de actividades necesarias para la transformación de un monolito a microservicios

## **OBJETIVOS GENERAL Y ESPECIFICOS**

### **Objetivo General**

Definir una arquitectura que sirva de guía a un equipo de desarrollo para transformar una aplicación monolítica a componentes de tipo microservicio.

### **Objetivos Específicos**

- Modelar diagramas que describan los componentes de la arquitectura al igual que sus conexiones e interacciones.
- Establecer una guía para la migración de un aplicativo monolítico a su equivalente en microservicios para la disminución de costos de actualización y tiempos de mantenimiento para las empresas.
- Proponer una escala de referencia que sirva para saber si un aplicativo monolítico es elegible o no para ser transformado a un sistema basado en microservicios.
- Validación de la arquitectura en un aplicativo monolítico transformado a microservicios usando tecnologías java y spring cloud.

## DESARROLLO

### Modelado De Arquitectura

Los microservicios tienen una estructura interna similar el cual se compone de unas capas internas que se muestran en la siguiente figura:

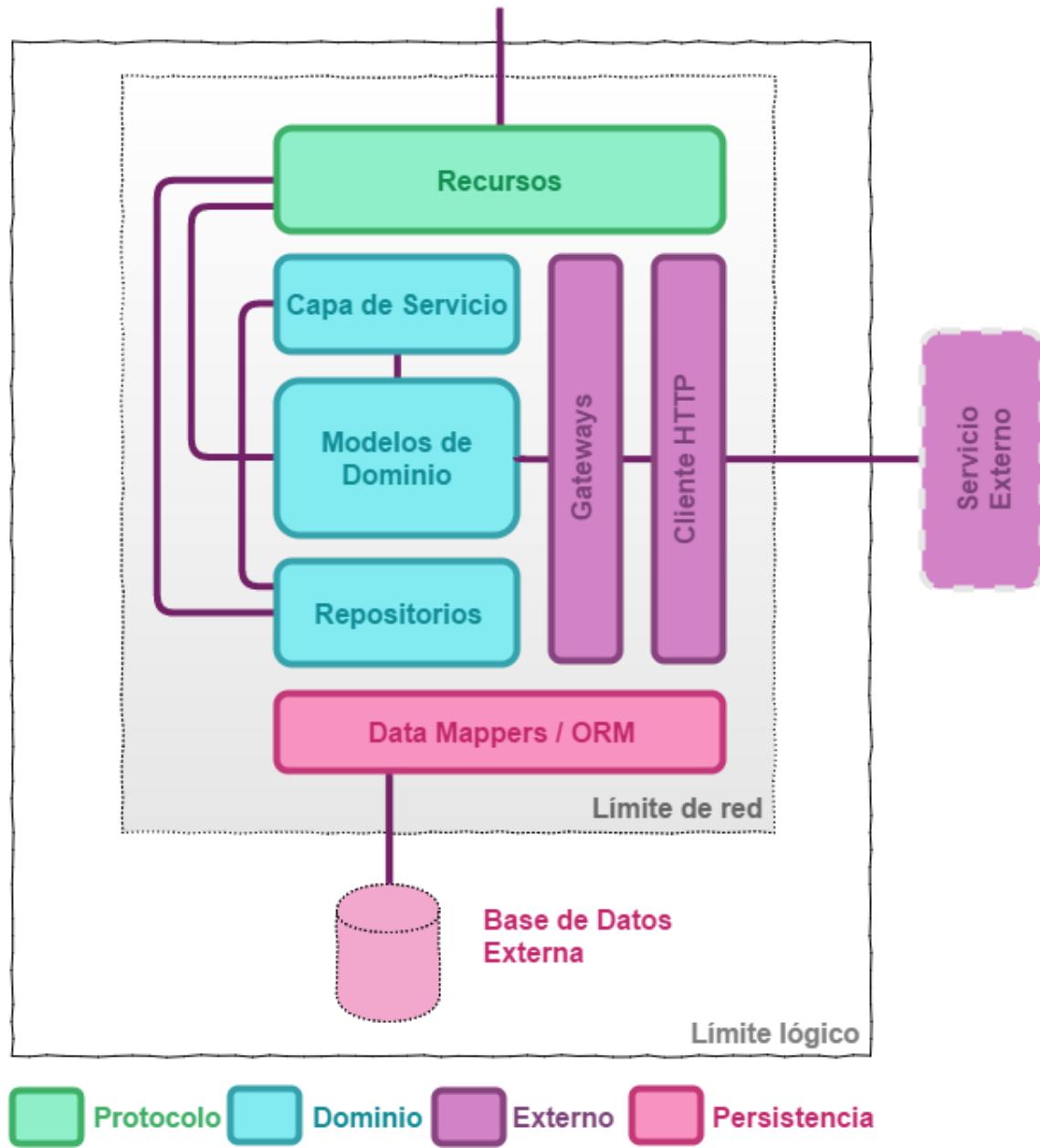


Ilustración 11 Anatomía de un microservicio (Clemson,2014)

## **Protocolo**

Los recursos actúan como mapeadores entre el protocolo expuesto por el servicio y los a los objetos representados por el dominio. Típicamente los recursos son livianos y tienen la responsabilidad de verificar las peticiones de red y proveer una respuesta específica al protocolo de acuerdo al resultado de la transacción del negocio. Un recurso recibe una petición que una vez validada, llama a la capa del servicio quien comienza a procesar la petición.

## **Dominio**

La capa del servicio es responsable de la lógica del negocio , los modelos del dominio contendrá las entidades asociadas al servicio u el repositorio se hará cargo de almacenar en memoria las listas de los modelos del dominio que necesitan ser persistidos en base de datos.

## **Persistencia**

Un microservicio podrá persistir los objetos del dominio entre las peticiones. Usualmente esto se logra usando mapeo objeto relacional o mapeos más livianos dependiendo de la complejidad de los requerimientos de persistencia. La base de datos e acoplada al sistema sin embargo debido a que está por fuera de los límites de la red pueden presentarse problemas de latencia o riesgo de desconexión.

## **Externo**

Si un servicio tiene otro servicio como colaborador, será necesaria la definición de una lógica que permita la comunicación con el servicio externo. Una puerta de enlace encapsula un mensaje para pasarlo a un servicio remoto, transformando peticiones y respuestas con los para los objetos de dominio. Un cliente se encargaría de entender el protocolo manejado en el ciclo de solicitudes y respuestas. Las conexiones a servicios externos requieren atención especial debido a que cruzan el límite de la red. El sistema debería ser resiliente a fallos de componentes remotos. Las puertas de enlace contienen la lógica necesaria para manejar los casos de error.

Para disminuir el acoplamiento entre un microservicio y los servicios externos se puede utilizar comunicación por mensajes con el fin de haciendo uso de llamados asíncronos del tipo consumidor suscriptor que son menos acoplados que mejor que los llamados síncronos.

## Ecosistema de microservicios

Debido a que los componentes se encuentran aislados por límites de red, es importante considerar los posibles modos de fallo en el sistema. Técnicas como tiempos de espera o interruptores de circuito ayudan a mantener el tiempo de funcionamiento general del sistema a pesar de una interrupción del componente.

En la siguiente ilustración se puede apreciar la manera como los microservicios interactúan entre ellos por medio del protocolo definido en la aplicación y con otros servicios externos que tienen un protocolo de comunicación propio que los microservicios necesitan soportar.

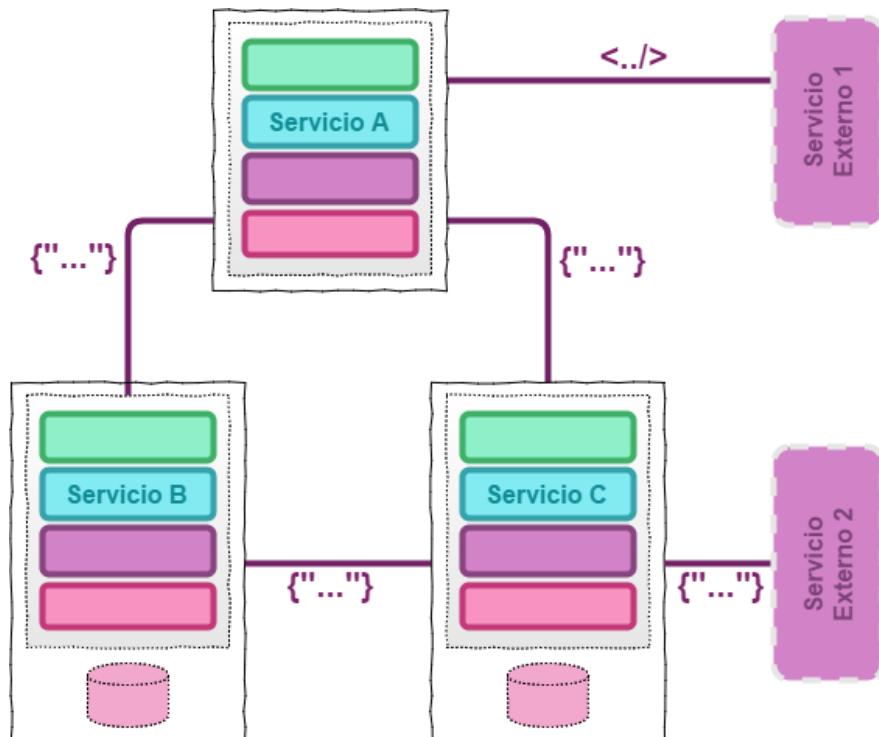


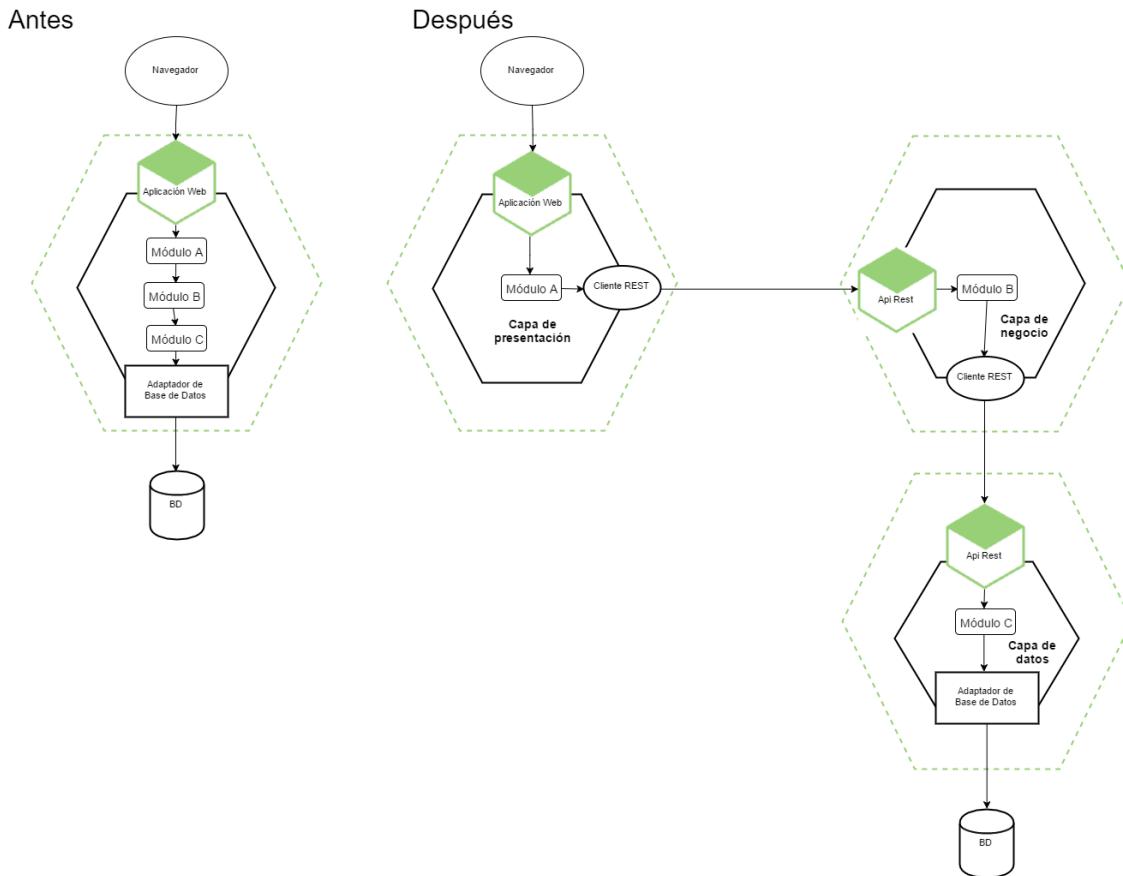
Ilustración 12 Ecosistema de microservicios (Clemson,2014)

## Fragmentación conceptual

Las aplicaciones monolíticas a medida que crecen son susceptibles al desarrollo de anti patrones de diseño que la convierten en sistemas complejos y difíciles de mantener para los desarrolladores., una posible solución a este problema es detener el crecimiento de la arquitectura monolítica de la aplicación y escalarla usando una arquitectura basada en microservicios que soporte las futuras actualizaciones de la lógica del negocio Los

microservicios deberá estar en capacidad de comunicarse entre ellos por medio de un api o un canal de comunicación orientada a mensajes (productores y consumidores), los pasos para empezar a reescribir un monolito a microservicios comienzan partiendo el monolito en tres grandes capas: La capa de presentación, la capa de negocio y la capa de datos (Richardson,2016).

Para comenzar se deben de definir tres granos gruesos representantes de las capas anteriormente mencionadas, y aplicando un enfoque de diseño guiado por dominio, se definirán dentro de capa los *bounded context* que se desean convertir a microservicios. Cada vez que se extrae un módulo y se convierte en un servicio, el monolito se encoje. Una vez que se hayan convertido suficientes módulos el monolito tenderá a desaparecer.



**Ilustración 13 Refactorización de un monolito a microservicios. (Richardson,2016)**

## **Planeamiento y factibilidad**

La arquitectura microservicios permiten romper aplicaciones monolíticas en piezas más manejables de código, lo que permite hacer despliegues más rápido, mitigar los efectos de las fallas del sistema, separar el ciclo de vida de los artefactos los cuales por medio de *devops*<sup>7</sup> pueden ser actualizados, escalados y administrados interdependientemente (Loukides,2012). A continuación se listan algunas recomendaciones que deberían ser evaluadas antes de iniciar un proceso de transformación a microservicios para lograr una exitosa implementación.

### **Determinar el nivel de madurez**

Daniel Bryant (Bryant,2015), elaboró una clasificación taxonómica de las plataformas de acuerdo a su nivel de maduración, a continuación se explican los tipos de plataforma que son candidatas a ser transformados a microservicios. Y se clasifican por edad y KLCF/PA<sup>8</sup>.

**Plataforma megalítica (+10 años, 1000 KLCF/PA).** Aplicación enorme con una sola base de código, es casi imposible entenderla y mantenerla, sus artefactos tiene muy baja cohesión y un altísimo acoplamiento, baja o nula modularidad.

**Plataforma Megalítica (+5 años, 100 KLCF/PA).** Aplicación grande con una sola base de código resultando en un sola aplicación, su mantenimiento se hace difícil , los artefactos tiene mucho acoplamiento y su cohesión va de bajo a intermedio. Escrita en un solo lenguaje y distribuida en algunos componentes

**Plataforma Macro SOA (+5 años, 10 KLCF/PA).** Aplicación SOA clásica, se compone de varios sistemas grandes (Candidatos a monolito) levemente acoplados. Su acoplamiento va de alto a medio y su cohesión de media a alta. Escrita en varios lenguajes y con muchos componentes a nivel de la plataforma

<sup>7</sup> Prácticas automatizan el proceso de entrega del software y los cambios de infraestructura.

<sup>8</sup> KLCF/PA: Kilo Líneas de Código Fuente por Artefacto

**Plataforma Meso aplicación (2-5 años, 10-1 KLCF/PA).** Servicios de tamaño medio interconectados para formar una sola aplicación. Es en esencia un híbrido entre monolitos y microservicios. Su acoplamiento puede ir de alto a bajo y su cohesión de media a alta. Escrita en varios lenguajes y con múltiples componentes

Aplicaciones web fuera de estas clasificaciones dificultan la implementación de los microservicios, de acuerdo con Martin Fowler aplicaciones web nuevas debería seguir un enfoque de primero monolito debido a que su funcionalidad debe ser validada con los usuarios y la delimitación entre los servicios debe estar bien definida.

### **Propiedad intelectual sobre el código fuente**

Es deseable que la organización cuente con el acceso para modificar su código fuente y de esta manera se puedan hacer los ajustes necesarios para migrar la aplicación monolítica a microservicios, en caso de no tener el acceso deberá solicitar mejoras para que la aplicación exponga servicios web que puedan ser accedidos por otros microservicios a futuro.

### **Equipo de desarrollo calificado**

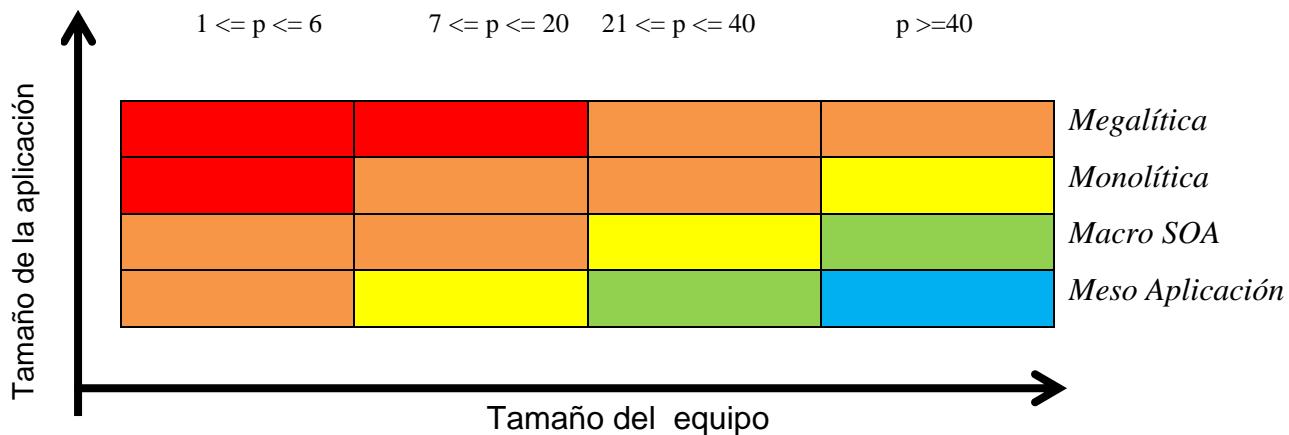
Es necesario preparar a los desarrolladores para un cambio en el paradigma de desarrollo, los microservicios manejan comunicación asíncrona lo cual eleva la complejidad de la codificación, depuración y ejecución de pruebas. Se debe escribir código extra para manejo de errores, manejo de transacciones distribuidas. También la realización de despliegues se hacen más difíciles debido a que se requiere creación de script para realizar integración continua y necesita coordinación entre todos los servicios para un despliegue completo. (Celesti & Leitner, 2015)

### **Comenzar con artefactos de baja criticidad**

Siempre es mejor adoptar nuevas tecnologías sobre artefactos que tenga un rol menos significativo dentro de la aplicación, de esta manera si se presentan defectos o problemas en

el despliegue las consecuencias no serán tan negativas como si se hubieran presentado en un artefacto crítico.

A continuación se representa una matriz basada en la escala de clasificación de proyectos, en la matriz se representa el esfuerzo que debería hacer un equipo para transformar una aplicación centralizada a una aplicación basada en microservicios, el esfuerzo se representa por un código de colores (azul, verde, amarillo, naranja y rojo) El color azul representa un menor esfuerzo, y el color rojo representa el mayor esfuerzo. Originalmente la matriz de Cockburn representa la formalidad de un proyecto, como la formalidad en un proyecto de software y el esfuerzo son proporcionales se construyó una matriz a semejanza usando la misma tendencia lineal.



**Ilustración 14 Adaptación de la escala de Cockburn para determinar el esfuerzo de la transformación de un monolito a microservicios (Larman 2004, Figura 7.2)**

Para determinar si una aplicación web es elegible o no de ser transformada a microservicios basta con hacer una inspección de los enunciados anteriormente mencionados para continuar con la metodología

## **Educción y análisis de requisitos**

### **Definición de metodología ágil y técnicas de educación**

Para iniciar el proceso se requiere escoger una metodología ágil que permita desarrollar los microservicios, para esta metodología se recomienda *scrum* por ser de las más populares entre las metodologías ágiles en la industria (Silverthorne,2015) y de la técnicas de educación se recomiendan las entrevistas, grupos de trabajo y análisis del dominio (revisión de documentación existente) ya que son técnicas que permiten definir rápidamente los requisitos, en vez de largas especificaciones escritas, y revisar la documentación existente para precisar cualquier regla o información generada de las conversaciones.

### **Lista de requisitos**

Luego de aplicar las técnicas de educación se identificarán las funciones que va incluir el sistema, se pueden listar frases generales que diga lo que el sistema debe hacer y posteriormente se pueda priorizar

## **Especificación de requisitos**

### **Construcción de historias de usuario, clasificación y priorización**

Las historias de usuario son las piezas en un sistema de software, ellas deberán describir la funcionalidad del sistema (Requisito funcional), o la propiedad del sistema (Requisito no funcional). Cada historia definirá detalladamente las condiciones iniciales y los criterios de aceptación, una buena manera de hacerlo es usando los siguientes conectores de texto “Como un/a...cuando ... quiero ... porque ...“ . El “como un/a” se refiere a quien solicita la historia, el “cuando” se refiere a las condiciones que deben cumplirse para llevar acabo la acción, el “quiero” se refiere a la funcionalidad deseada y el “porque” a la justificación de la solicitud de esa funcionalidad (Fowler,2013). También en algunos casos serán necesario definir gráficos que permitan entender la arquitectura o un diseño existente para ello se pueden hacer uso de los diagramas UML. Como labor adicional se crearán vínculos entre las historias de usuario y los requisitos que dieron origen a estas de tal manera que se genere una trazabilidad que permita monitorear el ciclo de vida del proyecto.

### **Construcción de prototipos**

En algunas historias de usuario funcionales la construcción bosquejos de la interfaz gráfica sirve poder definir un referente visual que le permita a los programadores tener una base de como el cliente desea las interfaces gráficas, permitiéndole identificar colores, estilos, tipos de fuentes y la división por componentes que requieren las interfaces gráficas, (Cohn, 2004)

## **Validación de requisitos**

### **Validación de historias de usuario**

Cada historia de usuario necesita ser validada antes de ser presentada y estimada al resto del equipo, (Wake, 2003) definió una ayuda nemotécnica que describe como deben ser las buenas historias:

***Independientes:*** Las historias se pueden entregar en cualquier orden

***Negociables:*** Los detalles adicionales a la historias son co-elaborados por los desarrolladores y el cliente durante el tiempo de desarrollo.

***Valioso:*** La funcionalidad debe ser vista como valiosa por el cliente o los usuarios del sistema

***Estimable:*** Los programadores pueden hacer una estimación razonable para construir la historia.

***Pequeña:*** Las historias deberían hacerse en un pequeño periodo de tiempo, usualmente es materia de días. Se debería poder hacer varias historias en una sola iteración

***Comprobable:*** La historia debería poderse probar para verificar si cumple los criterios de aceptación.

### **Priorización y estimación**

Basta definir una escala sencilla como bajo, medio alto donde el cliente pueda definir en que orden se deben construir las historias de usuario. Pueden ser diferentes los criterios para definir este valor como costo, complejidad, riesgo o incertidumbre. Para la estimación se puede utilizar la técnica del *poker* (Massacci & Redwine & Zannone 2009 ) donde en una mesa redonda se expliquen las historias y se voten entre el cliente y los miembros del equipo; los valores se promediarán arrojando un valor final de estimado por historia de usuario. Las reuniones de mesa redonda son valiosas también para aclarar dudas y hacer precisiones de conceptos.

## Diseño y definición de microservicios

### Modelado de servicios

El enfoque de *Domain Driven Development* es el punto de partida para la definición de los microservicios (Stenberg,2016), la definición de los *bounded context* y es el puente que une a los expertos del dominio con los programadores. Para el modelado de servicios se deberán realizar las siguientes actividades:

**Analizar diagramas UML.** Los diagramas UML permitirán poder analizar los objetos y flujo del sistema, por ejemplo los diagramas de actividad son de gran ayuda ya que por medio del nombre de las actividades o de las particiones se pueden inferir generalidades y los nombres de los posibles contextos que tiene la aplicación. También se puede analizar manuales existentes o el código fuente para la extracción de palabras claves que sirvan para la definición de los contextos.

**Definición de los contextos.** A partir del análisis de la documentación de proyecto fuente se procederá a definir los nombres, modelos y límites de los contextos que existen en la aplicación al igual que las relaciones internas entre los modelos. Los contextos deben ser lo suficientemente granulares de tal manera que cumplan un propósito específico.



Ilustración 15 Caso de ejemplo de extracción de contextos

**Resolución de ambigüedades:** Es posible que un modelo sea compartido por dos contextos, y se puede presentar el caso de que el modelo sea idéntico para ambos contextos o bien que tengan el mismo nombre pero sean diferentes en su contenido, por esta razón es preferible duplicar un modelo que adaptarlo para que sea compartido entre dos contextos.



Ilustración 16 Resolución de ambigüedad de un modelo entre varios contextos (Lerman,2014)

## **Fragmentación del empaquetamiento del monolito**

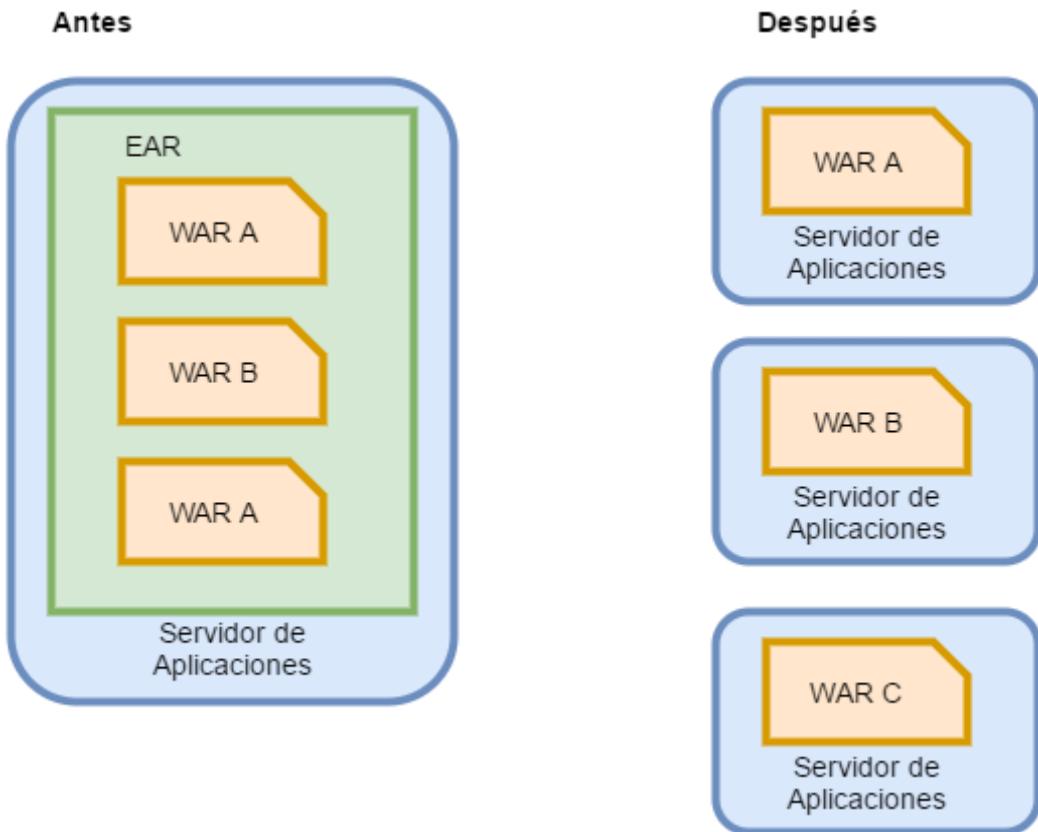
Luego de haber definido los posibles contextos en los que puede ser dividido el monolito se procederá a fragmentarlo a semejanza de la separación planteada por contextos siguiendo los siguientes pasos:

***Refactorizar el empaquetamiento del monolito.*** El código fuente debe de ser empaquetado de acuerdo a la organización de los contextos, se deberán mover los archivos fuentes a nuevos proyectos que sean representativos de cada contexto.

***Divida su archivo distribuible.*** No se debe de generar un solo archivo distribuible para toda la aplicación (Ejm: Archivo ear en java) , más bien los archivos binarios se deben de dividir en varios archivos distribuibles por paquete. Este cambio no representa mayores modificaciones en el código fuente, tan solo ajustes en las rutas de los contextos.

***Contenedor por servicio.*** A continuación aplique el patrón de “Contenedor por servicio” y despliegue cada archivo distribuible en su propio contenedor (Haciendo uso de *Docker* o *Bluemix*). Los contenedores podrán escalarse independientemente.

***Construir, desplegar y administrar independientemente.*** Una vez estén divididos los servicios por contenedor, se puede manejar los archivos distribuibles independientemente a través de trabajos de *DevOps* automatizados. Este es un paso camino hacia el proceso de integración continua.



**Ilustración 17 Particionamiento del empaquetamiento de la aplicación monolítica (Brown 2016)**

### Preparación de la capa de datos de los microservicios

La base de datos también está sujeta a cambios necesarios de tal manera que disminuya el nivel de acoplamiento entre los servicios, los siguientes casos deberán ser evaluados para proceder a una adecuada fragmentación del código fuente. (Belcham,2016).:

***Islas solitarias de datos.*** Es importante mirar las tablas de la base de datos que el servicio está usando. Si las tablas usadas son independiente entre ellas o forman pocas relaciones, entonces solo será necesario dividir estas del resto del modelo de datos. Una vez haya hecho esto, se puede definir cuales tablas se incluirán en cada microservicio

***Desnormalización de tablas.*** La normalización se ha utilizado tradicionalmente para mitigar la duplicidad de la información, sin embargo esta genera acoplamiento en el monolito entre la capa de datos y la lógica del negocio. Mientras no existan restricciones en el

rendimiento para la escritura y acceso de la base de datos relacional se procederá a definir un nuevo esquema da datos distribuido entre las bases de datos de los microservicios implicados, las llaves foráneas se pueden reemplazar por medio de servicios *restful* que ayuden a combinar los registros y a garantizar la integridad referencial. La inserción de datos distribuida se puede soportar haciendo uso de un *middleware* que ofrezca comunicación por mensajería y que permita la inserción de datos de forma asíncrona entre los microservicios.

**Creación de servicio para exponer datos.** En algunos casos es necesario crear un servicio para exponer una base de datos relacional existente o bien para compartir el acceso entre varios microservicios. Para este caso debe ser creado un servicio tipo CRUD que convierta los objetos provenientes de la base de datos a objetos *json* y los exponga en una interfaz tipo *Restful*.

**Utilización de base de datos NoSQL.** Las bases de datos *NoSQL* ofrecen mayor rendimiento que las bases de datos relacionales y su escalabilidad se maneja sobre demanda puesto que su almacenamiento es en memoria *RAM*. Parte de la base de datos relacional podría ser migrada a una base de datos *NoSQL* a favor de algún requisito de rendimiento de la aplicación o para tomar ventaja de su flexibilidad para la definición de datos.

## Refactorizando el código del monolito

Luego de haber modificado la manera como se generan los archivos que distribuyen la aplicación y como se van a acceder a las tablas se procederá a refactorizar el código fuente de tal manera que se pueda establecer niveles más granulares. Existen tres casos en donde se pueda refactorizar código existente para que puedan interactuar con otros microservicios.

**Servicios REST.** Este el caso de refactorización más sencillo, para comenzar se necesita desacoplar el servicio *restful* del código fuente del monolito creando un nuevo proyecto web en el código fuente y generando un nuevo archivo distribuible. El aislamiento del servicio debe ser maximizado por lo tanto la duplicación de código fuente es permitida para que el servicio sea auto contenido y no tenga ninguna dependencia.

**Servicios SOAP y/o protocolos de comunicación binarios.** Si el monolito tiene servicios SOAP o de comunicación binaria probablemente fueron diseñados bajo un enfoque funcional por medio de algún tipo de fachada. Para ofrecer el servicio como microservicio es necesario crear una nueva interfaz tipo *restful* y mapearla con interfaz del servicio de igual manera es necesario convertir la representación de los objetos de memoria a JSON. En caso que el servicio en cuestión tengan funcionalidad compleja se puede implementar el patrón comando.

**Servlets simples / Interfaces JSP.** Muchas aplicaciones web se componen de páginas web que se conectan a tablas en la base de datos. En este caso se debe crear una capa de servicio *restful*, posterior a esto se podrá hacer refactorización de la aplicación existente o bien se puede construir una nueva interfaz gráfica o una aplicación nativa móvil.

## Desarrollo y pruebas de unidad

### Registro de microservicios

Un servidor de registro es una especie de guía telefónica para los microservicios. Cada microservicio se registra así mismo en el servidor de registro y le notifica información de identificación dentro de la red como su ip, puerto, nombre del host entre otros posibles datos. El servidor de registro es una aplicación que permite la redirección de los llamados entre los microservicios (Long, 2015). Los microservicios necesitaran de un cliente que se comunique al servidor de registro, el cliente se identificará con el registro al momento de su arranque y de ahí en adelante enviará llamados tipo de renovación o *heartbeat*, periódicamente, donde notificara que se encuentra disponible en la red. A continuación se muestra una imagen que muestra la manera como se comunican los microservicios con un servidor de registro y la forma como acceden al descubrimiento de los servicios registrados.

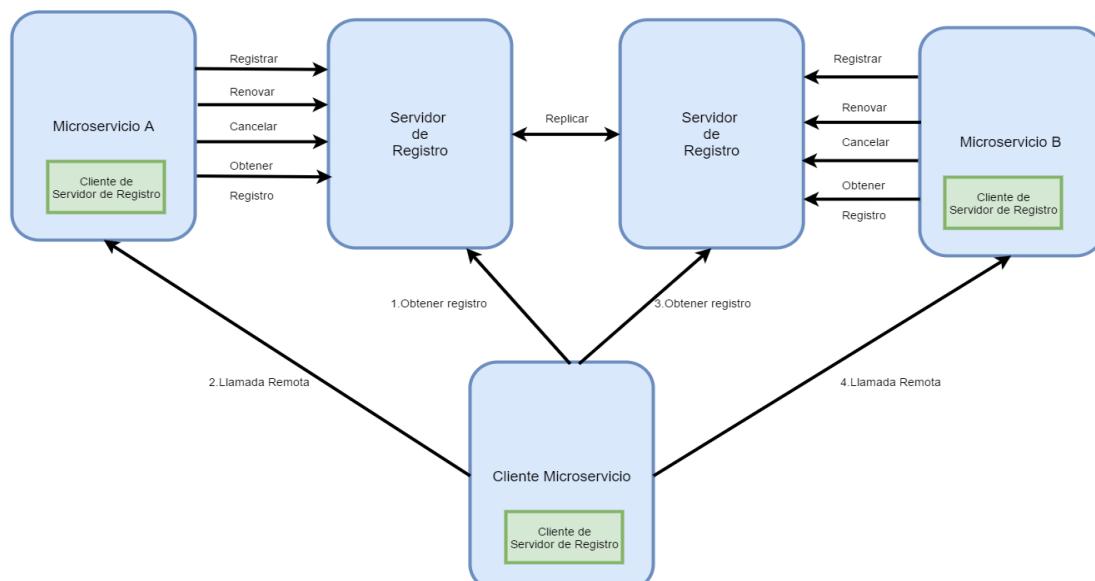


Ilustración 18 Arquitectura de un servidor de registro de microservicios (Haddad,2012)

**Definir servicio de registro.** Es necesario definir un proyecto de código fuente exclusivamente para el ofrecimiento del registro y descubrimiento de microservicios, existen

varias alternativas que pueden ser usadas para este propósito, entre las más populares se encuentran:

- *Netflix eureka*: Provee un API tipo REST para registrar y consultar instancia de servicio. Eureka tiene una alta disponibilidad en servidores de amazon que permite formar clusters de servidores y contienen un balanceador de carga interno.
- *Consul*: Es una herramienta para el descubrimiento y configuración de servicios. Provee un API interna que permite a los clientes poder registrar y descubrir servicios al igual provee una interfaz DNS interna para hacer consultas por medio del protocolo DNS. Consul permite diagnosticar estados de salud para determinar la disponibilidad de los servicios.
- Apache Zookeper: Ampliamente utilizado, facilita la coordinación de servicios de alto desempeño en aplicaciones distribuidas.

***Configurar clientes en los servicios y en el monolito.*** Luego de haber definido un proyecto de código fuente para el servicio de registro, deberán de incluirse aplicaciones cliente por cada microservicio, los clientes deberán de tener en su configuración necesaria para hacer uso del servicio de registro (Host,Puerto, Endpoint, etc..), cada cliente se registra y podrán realizar el descubrimiento de los demás microservicios.

Para las aplicaciones monolíticas el procedimiento será igual, el cliente definido deberá servir de interfaz entre el monolito y el protocolo usado por el servicio de registro.

***Proxy para ofrecimiento de microservicios.*** Cuando sea necesario ofrecer microservicios por medio de un API o bien se disponga de una aplicación web que utilice intensamente llamadas Ajax es conveniente por seguridad y limpieza del código aplicar un patrón proxy que permita centralizar las llamadas de los microservicios hacia un host que maneje unificadamente las rutas web.

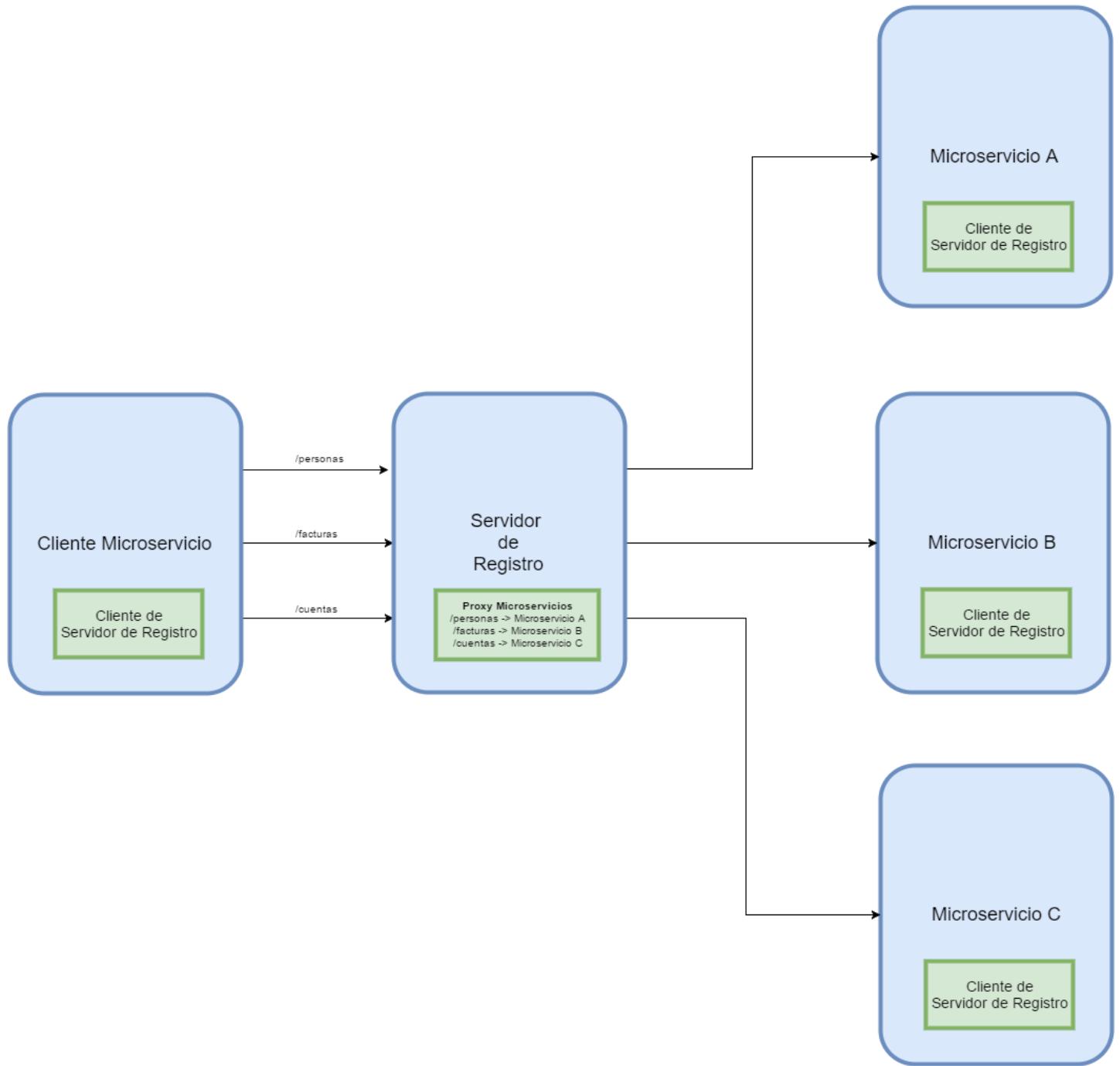


Ilustración 19 Enrutamiento de microservicios por medio de filtro a través de un proxy (Sastrasinh,2016)

## Manejo De Tolerancia A Fallos

Uno de los mayores problemas en las aplicaciones distribuidas es la propagación en cascada de fallos entre los servicios, es por ello que se hace necesario el aislamiento de servicios para evitar un fallo general en el sistema.

**Uso del patrón disyuntor.** El patrón disyuntor es la solución más popular para proteger las llamadas de servicios remotos, mientras no se presente ningún tipo de error el circuito se mantiene cerrado y las llamadas remotas se realizan de manera normal (Fowler,2014). En caso de presentarse algún tipo de error el circuito se abre evitando que la llamada remota sea realizada, una vez el circuito detecta que la llamada remota puede realizarse exitosamente el circuito se vuelve a cerrar y permita que el llamado remoto sea realizado.

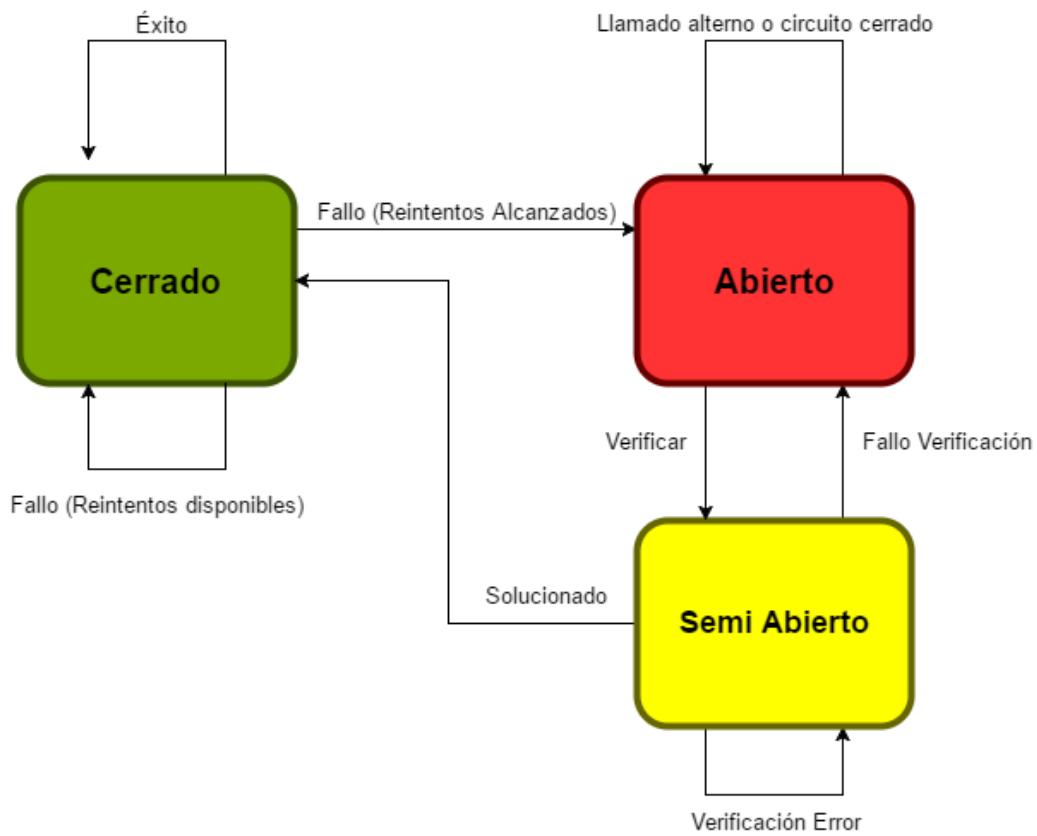
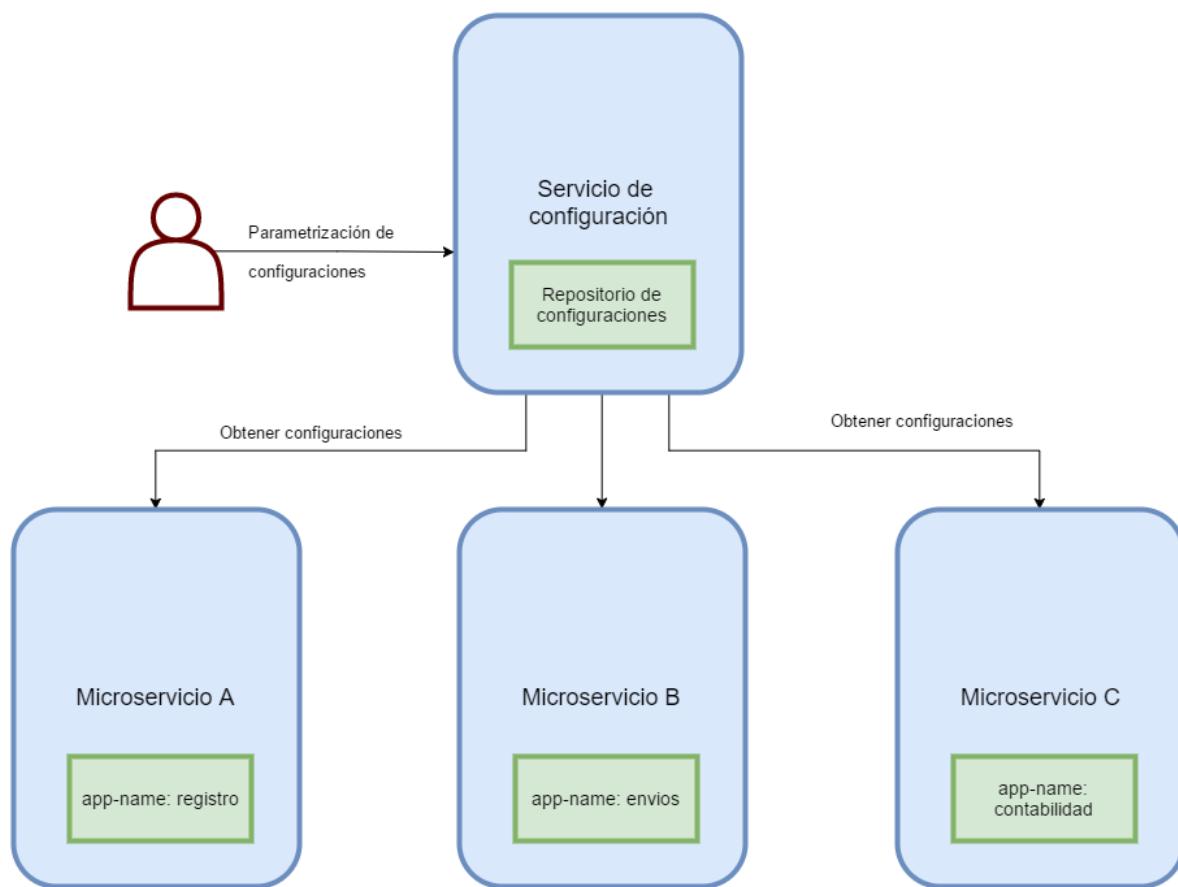


Ilustración 20 Representación del patrón disyuntor

*Adición de métodos manejadores de errores para construir el disyuntor.* Para la implementación del patrón disyuntor en los servicios se deberán definir métodos alternativos que puedan ser ejecutados cuando el circuito se abra, los métodos que se deberán intervenir serán aquellos que encargados de realizar llamadas remotas y de esta manera controlar la mayor parte de errores (Gajda,2016) .

## Configuración centralizada

La entrega continua requiere que la aplicación esté en capacidad de funcionar en diferentes tipos de entorno de despliegue como por ejemplo: Local, desarrollo, pruebas, *QA* y producción. Para ello es posible disponer de un servicio que permita compartir la configuración entre los microservicios de acuerdo al tipo de entorno que se deseé, entre los valores de configuración más comunes de configurar son las dirección ip, puertos, nombres de bases de datos, etc. (Stine,2015) En la siguiente figura se muestra como un servidor de configuración interactúa entre los microservicios.



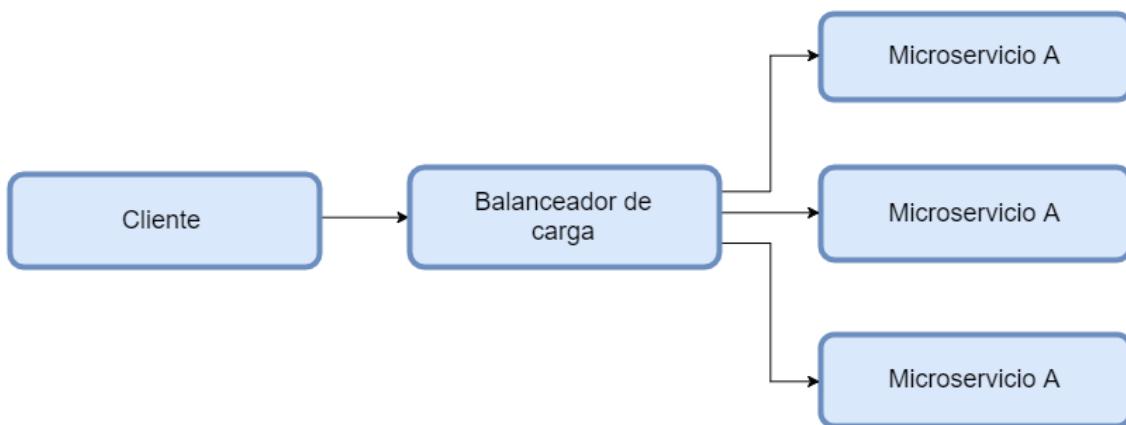
**Ilustración 21 Arquitectura del servidor de configuración**

***Creación de microservicios y repositorio de configuración.*** Es necesario definir un proyecto de código fuente adicional encargado de publicar los valores de configuración por medio de una mapa de tipo llave, valor. El servicio de configuración deberá de tener acceso a un repositorio de versionamiento o archivo que permita a los programadores adicionar o modificar los valores de las configuraciones. La llaves de las configuraciones deberá estar conformadas por el nombre de la aplicación, el perfil del entorno (desarrollo, pruebas, aseguramiento de la calidad y Producción) y por el nombre de la propiedad.

***Uso de marcadores en los microservicios.*** Luego de haber creado el servicio de configuración y de inicializar los valores de las configuraciones iniciales se procederá a reemplazar en el código fuente los valores “quemados” por las llaves de las propiedades que se necesitan parametrizar desde el servicio de configuración.

## Balanceo de carga y monitoreo de llamadas

Para permitir que un servicio sea recuperable de fallos es necesario evitar la singularidad en los puntos de fallos, la manera más sencilla de solucionar este inconveniente consiste en tener múltiples instancias de un microservicio coordinadas por medio de un balanceador de carga centralizado encargado de distribuir las peticiones como se aprecia en la siguiente figura.



**Ilustración 22 Balanceador de carga centralizado con microservicios (Newman 2015, Figura 11-4)**

Este diseño es recomendable para sitios web que son fáciles de implementar, sin embargo este enfoque presenta el inconveniente que puede convertirse en un cuello de botella o bien caso de fallo del balanceador de carga el acceso al microservicios se denegaría para todas sus instancias.

**Soporte de balanceo de carga por cliente.** El balanceador de carga por cliente es una alternativa al centralizado porque permite que cada cliente pueda manejar sus propias decisiones a partir de estadísticas de la red y reglas internas para escoger cuál instancia de microservicios es la más adecuada a utilizar (Tonse, 2015). Este enfoque simplifica la administración del servicio, se auto escala de acuerdo a la cantidad de instancias disponibles y evita el embotellamiento a través de un punto de fallo.(Salerno,2016)

**Monitoreo de métricas microservicio.** El monitoreo de microservicios es una importante actividad que permitirá saber el estado de salud de sus instancias y conocer los el uso de los recursos de hardware al igual que información estadística de las peticiones de red de la lógica del negocio.

Cada instancia del microservicio deberá estar corriendo en un host por separado, se debe de definir el estado de salud de un servicio dependiendo el nivel del uso de la CPU, el tiempo de respuesta y tasas de error. Si el sistema de monitoreo detecta que las estadísticas se encuentra por fuera de los umbrales deberá de lanzar alertas al administrador (Bryant,2015).

En las colas de trabajo de monitoreo se puede crear eventos ficticios que verifiquen si el monitoreo está funcionando correctamente, esta técnica se denomina monitoreo semántico. El monitoreo semántico permite detectar errores en los puntos de integración entre servicios, los eventos ficticios ayudan a verificar los tiempos de respuesta entre el servicio y sus dependencias (Falé, Gebhardt,2017). Existen aplicaciones como Hystrix Dashboard de Netflix que permiten visualizar las métricas por instancia o cluster de instancias de un servicio para analizar transacciones exitosas, errores, fallos manejados, tiempos de respuestas entre otras.

## **Desacoplamiento de la comunicación entre los microservicios**

Normalmente los microservicios están orientadas bajo una comunicación sincronizada entre ellos, sin embargo la sincronización genera acoplamiento lo cual puede ser un impedimento a la hora de escalar algún servicio. Certo tipo de llamados que no necesitan respuesta inmediata como la creación o actualización o borrado de una relación foránea o por ejemplo la escritura de log son casos a elegibles para ser transformados a comunicaciones asincrónicas (Wolff,2010).

Se puede implementar una comunicación de sincronizada entre dos microservicios por medio de mensajes y haciendo uso de un *broker* o intermediario que prepare colas donde un productor pueda hacer llegar mensajes y los consumidores se puedan suscribir para recibir los mensajes. Cada mensaje está formado de una cadena de texto simple, sin embargo se puede hacer uso de procesos de serialización o deserialización de objetos en formato JSON para la transmisión de objetos java.

## **Manejo de bitácora (Log)**

La administración de bitácoras o logs de un microservicio es una actividad que manualmente puede ser muy tediosa puesto que los archivos se encuentran distribuidos entre varios hosts. El análisis de bitácoras, la estandarización y trazabilidad necesitan ser apoyadas por aplicaciones externas que faciliten una adecuada lectura.

***Seguimiento de la bitácora entre microservicios.*** Llevar a cabo un seguimiento a la traza de una bitácora entre varios microservicios requiere de un almacenamiento centralizado en una base de datos, pero a la vez desacoplado del resto del sistema por medio de mensajería facilita la lectura de los errores. Es recomendable que la sintaxis de las trazas de las bitácoras entre todos los microservicios sea homogénea para hacer más fácil la lectura de los mensajes.

***Interfaz gráfica para administrar bitácoras.*** Existen herramientas como *Kibana* que apoyan estas labores y permiten realizar búsquedas de texto, rango de fechas y expresiones regulares. Inclusive puede generar gráficos desde las bitácoras permitiendo cuantos errores se han generado a lo largo del tiempo.

## Pruebas de unidad

Las pruebas de unidad son pequeños fragmentos de código que sirven para verificar que la lógica de negocio del servicio no tiene errores y funciona correctamente, por lo general una prueba de unidad debe de probar una sola función o método siendo el nivel de granularidad más fino. Además, una prueba de unidad ayuda a verificar la cobertura del código, por lo tanto debe ser escrita lo más sencilla e independiente posible, se deben de ejecutar en solitario y asiladas del resto de componentes sin utilizar ningún punto de integración.

Las pruebas de unidad son una técnica casi obligatoria en la construcción o transformación de los servicios ya que permite detectar bugs que hayan sido introducidos durante la etapa actualización del código fuente. De igual manera las pruebas de unidad pueden ser un soporte para el desarrollo si deciden seguirse un enfoque de desarrollo guiado por pruebas o *TDD*<sup>9</sup>.

<sup>9</sup> TDD: Test driven development

## Pruebas de integración, contratos y punto a punto

La arquitectura de microservicios facilita refinamiento de los límites de la lógica de negocio lo cual permite aislar las pruebas de unidad haciéndolas más simples y fáciles de entender y mantener. La pirámide de Mike Cohn (Newman,2015) plantea la separación de pruebas por nivel de granularidad la cual es ideal para ser aplicada en un microservicio por sus naturaleza aislada y fragmentada.

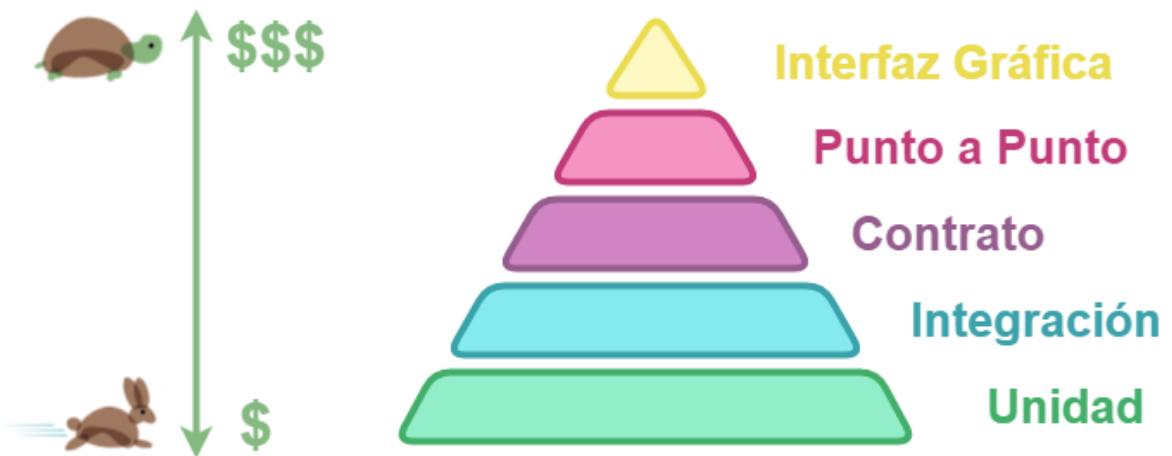


Ilustración 23 Fragmentación de la pirámide de Cok de acuerdo a los niveles de granularidad de las pruebas (Clemson 2014)

### Pruebas de integración

A continuación se enumeran algunos tipos de pruebas que tienen como propósito validar el comportamiento del servicio externos a este, con un nivel de granularidad intermedio.

**Pruebas protocolo.** Este tipo de pruebas deben de verificar puertos de escucha, encabezados HTTP, incorrecto manejo de SSL, análisis de los cuerpos de mensaje de las peticiones y las respuestas, cada prueba debe de poder responder en el protocolo utilizado por el cliente del servicio.

**Pruebas de servicio.** Por medio de aplicaciones que simulan clientes HTTP es posible realizar llamados de prueba al servicio para obtener respuestas y compararlas con las esperadas en el cliente HTTP. De esta manera se garantiza el correcto funcionamiento de los métodos públicos del servicio.

**Pruebas de persistencia.** Se debe de asegurar que el esquema de base de datos asumido por el código es el correcto, la ejecución de comandos y el soporte y el correcto funcionamiento de las transacciones también pueden ser probados aquí.

## **Pruebas de contratos**

Las pruebas por contrato permiten garantizar que las interfaces entre los servicios clientes y los servicios servidores se hallan mapeado correctamente o bien sirven para validar si un cambio futuro no corrompe la comunicación. Existen aplicaciones como *Pact* que permiten realizar este tipo de pruebas de manera automatizada.

## **Pruebas punto a punto**

Son las pruebas de mayor granularidad, permiten hacer pruebas reales y completas del sistema, en caso de hacer pruebas desde la interfaz gráfica se la prueba se puede apoyar en librería existentes que ayudan a automatizar pruebas de este tipo, o en caso de no necesitar una prueba desde la interfaz gráfica se puede usar un cliente web. Las pruebas punto a punto son difíciles de mantener a lo largo del tiempo por ello no recomienda escribir demasiadas pruebas de este tipo, prefiriendo realizar más pruebas de grano más fino.

## **Pruebas de producción**

La mayoría de pruebas se realizan antes de producción, sin embargo existen técnicas como *Canary Releasing* (Sato,2014), desarrollada por *Netflix*, la cual tiene como objetivo verificar una nueva versión de código de un servicio antes de redirigir el tráfico web de producción. La idea es verificar el desempeño de la nueva versión, el tiempo de latencia, capacidad de usuario, entre otros antes de hacer una migración. Se puede utilizar herramientas de monitoreo como *Hystrix Dashboard* para hacer el análisis de la nueva versión del servicio.

## Despliegue e integración continua

### Preparación entorno virtualización

La virtualización es la técnica que permite manejar divisores de máquinas virtuales dentro de un servidor, cada máquina virtual estará en condiciones de ejecutar diferentes tipos de programas. Para el despliegue de microservicios la técnica más utilizada es el uso de contenedores los cuales carecen de hipervisor, el componente que permite a un servidor tener varios sistemas operativos. Un contenedor funciona como un proceso separado con espacio en memoria y que convive con otros contenedores.

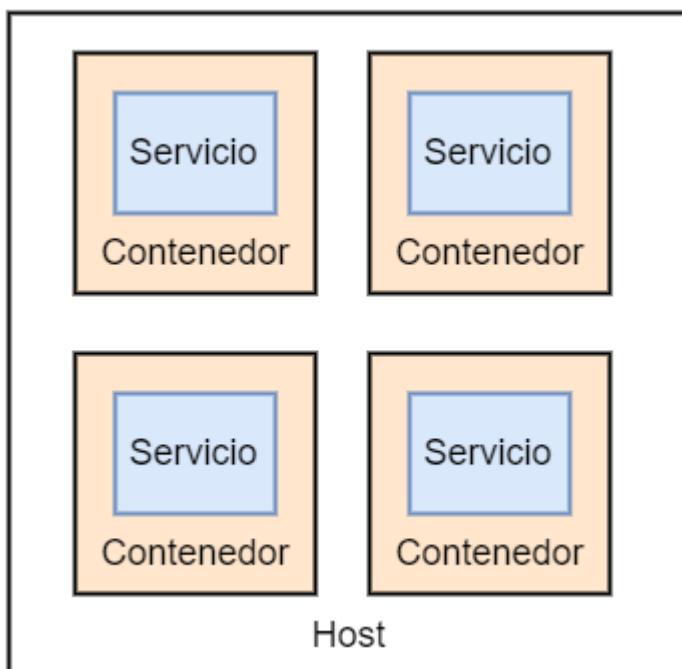


Ilustración 24 Múltiples servicios ejecutándose en un mismo host (Newman 2015, Figura 6-10)

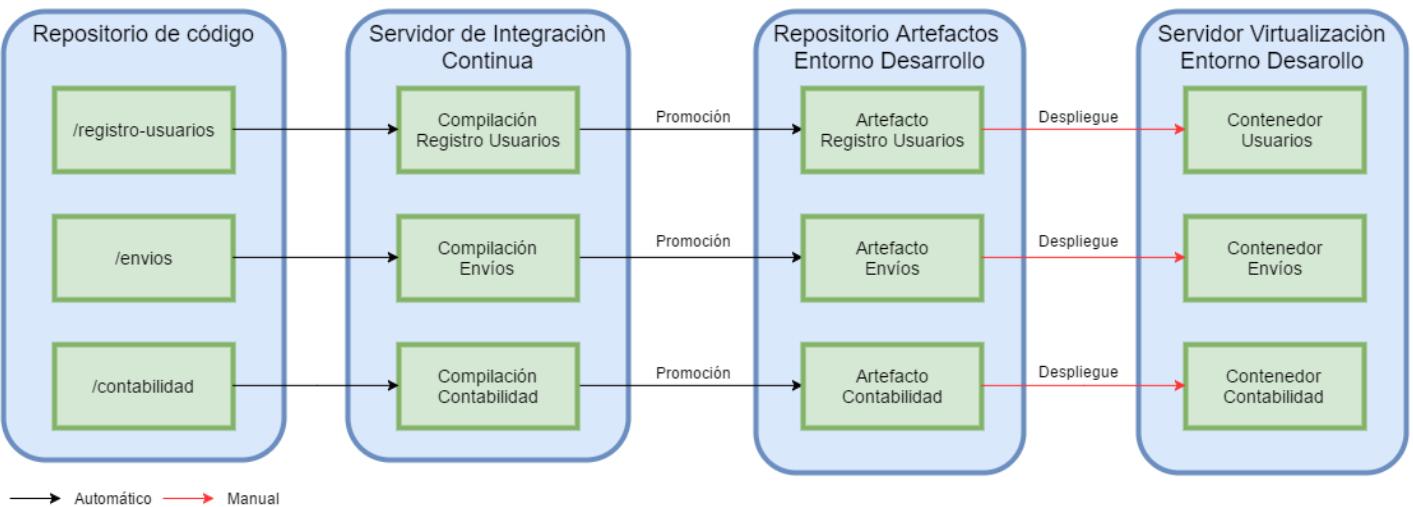
GNU/Linux permite construir contenedores, sin embargo la configuración y la administración de cada contenedor puede ser una labor muy dispendiosa ya que incluye tareas como la instalación o actualización de dependencias, configuraciones del sistema mapeo de puertos, etc.. al igual que existe la propensión a fallos si no se tiene la suficiente experticia sobre administración de sistemas operativos.

Existen aplicaciones como Docker, Kubernetes o Deis que funcionan como plataformas para PaaS (Platform as Service) que facilita la creación, aislamiento y administración de contenedores sin que los programadores tengan que preocuparse por asuntos de hardware , ayudan a la pre-configuración de las interfaces de red, y facilitan el escalamiento de microservicios en la nube por medio de imágenes redistribuibles por medio de servidores docker.

En caso que se necesite un mayor control sobre la configuración de la infraestructura se pueden utilizar plataformas para IaaS (Infrastructure as a Service) como Chef o Puppet las cuales permiten describir infraestructura in un lenguaje 4GL (Cuarto generación), estos archivos pueden ser replicables en la nube entre varios servidores.

### **Integración continua y entrega continua**

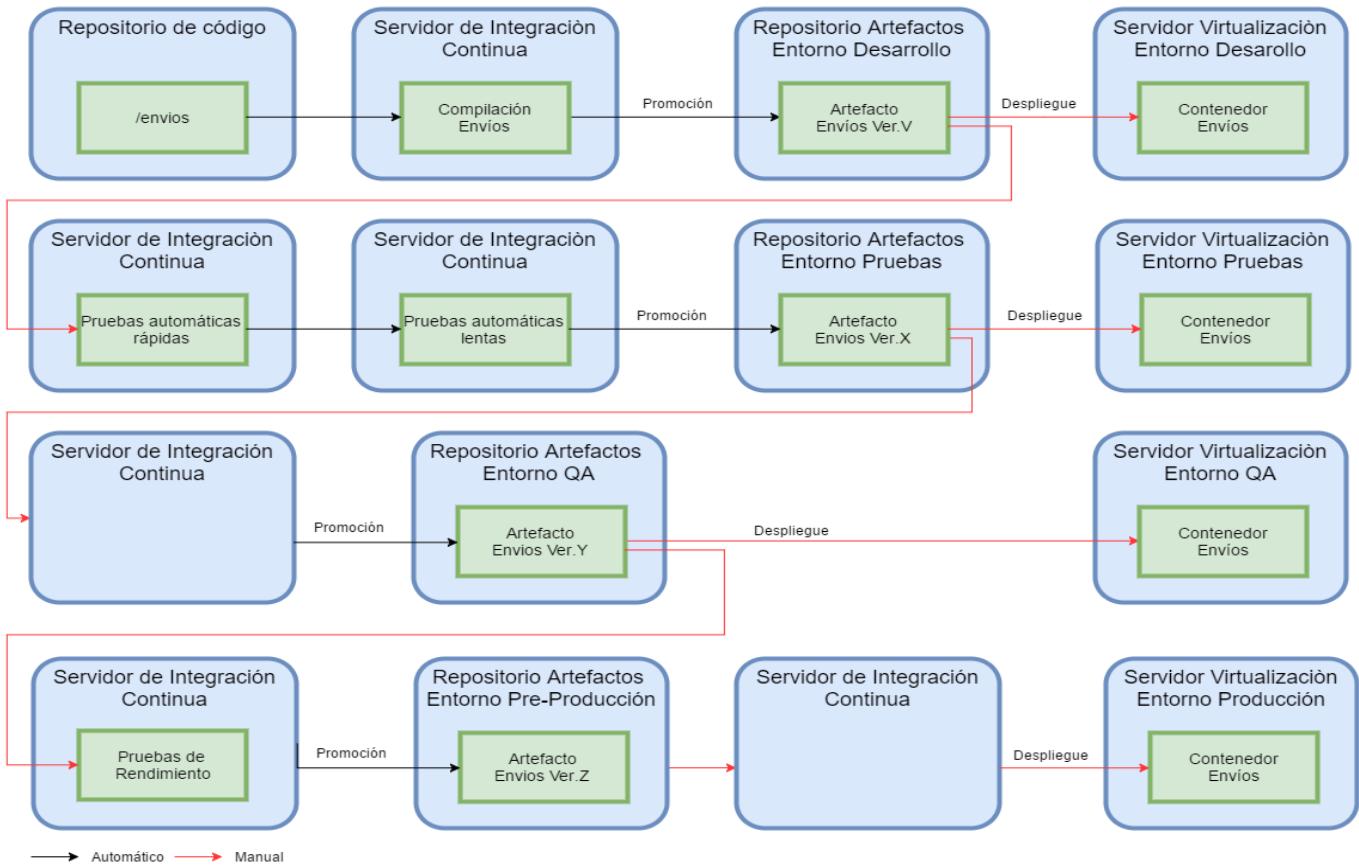
De acuerdo con (Fowler, 2006) El proceso de integración continua es la práctica que permite mantener sincronizados los artefactos con los últimos cambios hechos en el código fuente. Cualquier modificación realizada en el repositorio de código fuente será detectada por el servidor encargado de la integración continua el cual compilará de nuevo el artefacto y realizará las pruebas para verificar que todas fueron ejecutadas exitosamente; posteriormente el artefacto estará disponible para ser desplegado en un ambiente nuevo. El código fuente usado en el proceso de integración continua debe de estar versionado de tal manera que exista trazabilidad entre el código fuente y los artefactos compilados y así poder ser recreados en cualquier momento. En la siguiente gráfico se puede ver de manera esencial cual sería procedimiento de integración continua para una aplicación hecha en microservicios.



**Ilustración 25 Proceso de integración continua para varios microservicios**

Algunos pasos del flujo son automáticos y otros son manuales, inicialmente el servidor de integración continua estará detectando cualquier cambio que en el repositorio del código fuente para ejecutar una nueva compilación. De ser exitosa la compilación promocionará el artefacto al repositorio de artefactos para el entorno de desarrollo que quedará listo para ser desplegado.

Sin embargo, la integración continua por sí sola no ofrece la manera de evaluar la calidad de los servicios y es por ello que debe ser incluida dentro un flujo que permita evaluar calidad de los artefactos antes de su salida a producción. Humble (2011) define la entrega continua como el procedimiento por el cual se prueba y se prepara un artefacto antes de salir a producción, durante cada etapa del procedimiento se construye una versión del artefacto y se obtiene una retroalimentación para el perfeccionamiento del mismo. Para esta metodología se ha definido el siguiente flujo el cual deberá de seguir un servicio desde su compilación hasta ser liberado a producción.



**Ilustración 26 Proceso de Integración continua y entrega continua de un solo microservicio**

En el anterior gráfico se puede observar que el artefacto es promocionado en diferentes tipos de entornos, inicialmente el entorno de desarrollo será usado para que los programadores puedan realizar pruebas de desarrollo de los servicios, el entorno de pruebas servirá para ejecutar pruebas automáticas al igual que pruebas manuales hechas por los analistas de pruebas, el ambiente de QA será utilizado por el cliente para verificar las validaciones del aplicativo, el ambiente de pre-producción permitirá realizar pruebas de rendimiento antes de su liberación y si pasan exitosamente, el artefacto será desplegado en el ambiente de producción como punto final.

Cabe notar que las pruebas se dividieron en pruebas rápidas (unidad, contratos) y lentas (Integración y Punto a Punto), esta división permite optimizar los tiempos de corrección de las pruebas rápidas en caso de algún fallo de las mismas.

## **Ventajas y desventajas de la metodología**

La metodología ofrece un marco de trabajo basado en una filosofía de construcción software ágil, desde la etapa de prefactibilidad, el análisis y diseño, el desarrollo y el despliegue y soporte de la aplicación. El uso de la misma puede ser de gran utilidad para que las oficinas de tecnología puedan proponer un nuevo modelo de trabajo a sus empleados o a sus contratistas para que el desarrollo se enmarque en la construcción componentes de software distribuidos y autónomos que harán parte de un sistema de información legado. La construcción de nuevos componentes desacoplados facilitará su monitoreo en tiempo real y su escalabilidad en la nube donde continuarán interactuando con los sistemas monolíticos por medio de llamados servicios web,

La metodología también plantea el uso de integración y entrega continua con el objetivo de automatizar las tareas de compilación, pruebas y despliegues en diferentes ambientes de despliegue que permitirán detectar y/o prevenir defectos introducidos en el sistema antes de su salida a producción.

Como desventaja, la implementación de metodología requiere de entrenamiento del personal que estará a cargo de implementarla ya que su uso incrementa la complejidad y el esfuerzo necesario en la construcción y pruebas del software ya que está basada en una arquitectura distribuida y asíncrona.

## **VALIDACIÓN DE LA METODOLOGIA**

Para el ejemplo de la validación de la metodología se tomó un proyecto de software pequeño, y de código abierto, con el propósito de verificar los pasos expuestos en la anterior metodología, no se validará la etapa de factibilidad ya que su bajo nivel de madurez es ideal para ilustrar la metodología.

### **Preparación del ambiente de desarrollo**

El aplicativo monolito que se va a convertir a microservicios es un conjunto de servicios rest que funcionan a modo de un DAO los cuales realizan operaciones de lectura, creación, actualización y borrado de las entidades que componen un blog. Se necesitan conocimiento básicos en lenguaje de programación java y deseable pero no imprescindible conocimientos con las librerías apache-maven, spring framework, git, jpa, apache-tomcat y mysql .

#### **Aplicaciones y herramientas de desarrollo necesarias:**

Antes de comenzar, es necesario descargar las siguientes herramientas y proceder a instalarlas en la máquina. Para esta guía el paso a paso será desarrollada en un máquina con sistema operativo Windows 10.

Cliente git

<https://git-scm.com/download/win>

Java Development Kit (1.6 o superior)

<http://www.oracle.com/technetwork/es/java/javase/downloads/index.html>

IntelliJ Community Edition

<https://www.jetbrains.com/idea/download/>

Mysql (Seleccionar Servidor, Cliente y Workbench)

<https://dev.mysql.com/downloads/installer/>

Apache Tomcat (7.0.x o superior )

<http://tomcat.apache.org/>

Apache Maven (Versión 3.x.x o superior)

<https://maven.apache.org/download.cgi>

### **Configuración y consideraciones de las herramientas a instalar.**

Para la instalación de las herramientas bastará utilizar la configuración por defecto con las que viene el instalador, sin embargo hay algunas consideraciones que se tuvieron en cuenta para una correcta instalación del entorno:

- Al instalar el JDK asegúrese de crear la variable de entorno JAVA\_HOME en el sistema, esta variable debe de tener la ruta de donde quedó instalado el JDK de java, por ejemplo : C:\Program Files\Java\jdk1.8.0\_101
- Apache tomcat se distribuye como un zip y solo necesita ser descomprimido en una ruta en particular, es muy importante que se crea una nueva variable de sistema denominada CATALINA\_HOME la cual contiene la ruta donde tomcat fue descomprimido, en este caso se utilizará la ruta *D:\microservicios\apache-tomcat-7.0.73* (El número varia de acuerdo a la versión descargada). Para la correcta configuración del servidor tomcat y el manejo de la consola administrativa remítase al anexo de instalación y configuración de apache tomcat.
- Oracle provee un instalador de Mysql para Windows, este instalador trae muchos componentes pero la instalación puede ser personalizada para instalar los

componentes necesarios: Servidor, cliente y workbench. La instalación obliga a definirle una contraseña al usuario root de manera obligatoria, para ello se definirá la palabra root como contraseña.

*Ver anexo 3.*

- Apache maven, se distribuye en un zip y necesita ser descomprimido para uso, se utilizará la ruta *D:\microservicios\apache-maven-3.2.1* y se incluirá dentro de la variable PATH la ubicación de sus archivos binarios *C:\Program Files\maven\apache-maven-3.2.1\bin*, no se necesita realizar una configuración adicional para utilizarlo normalmente.

### **Descarga del código fuente, preparación de la base de datos y despliegue de la aplicación**

Como paso inicial se comenzará a descargar el código fuente del proyecto, se creará un folder llamado *D:\microservicios\ejemplo-metodologia-microservicios* en el computador desde donde se abrirá consola de git y se ejecutará el siguiente comando:

```
git clone https://github.com/benjsicam/restful-blog.git
```

Se procederá a actualizar el archivo propiedades dentro del proyecto *restful-blog\src\main\resources\META-INF\application.properties* modificando los siguientes valores:

```
database.host=jdbc:mysql://localhost:3306/restful_blog  
database.password=root
```

El primero corresponde a un cambio en el nombre de la base de datos y el segundo a la contraseña del usuario root. Se abrirá mysql-workbench y desde la consola sql se creará una nueva base de datos ejecutando el siguiente comando

```
CREATE DATABASE restful_blog;
```

Se seleccionará la base de datos

*USE restful\_blog;*

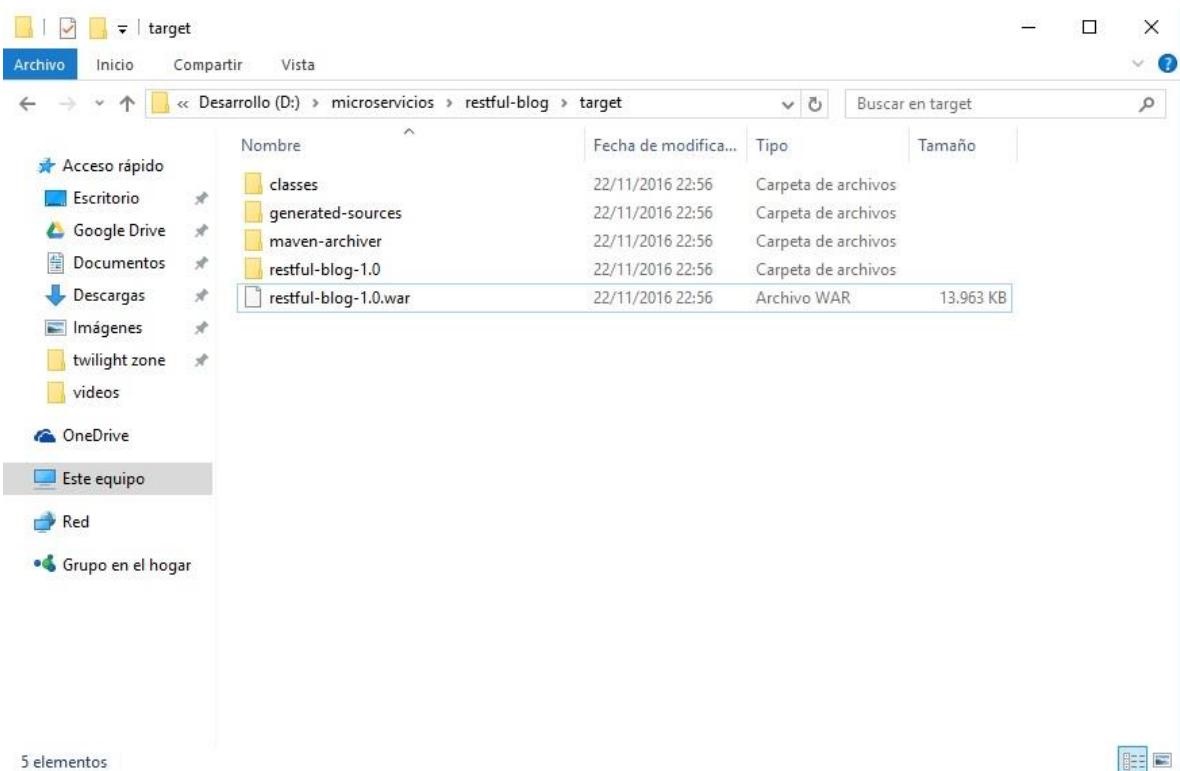
Se ejecutará el script sql contenido en el archivo src/main/resources/META-INF/init.sql

Este script creará las tablas y cargará los datos en ellas.

Posteriormente se regresará a la consola git y se realizará la compilación del proyecto maven usando el siguiente comando:

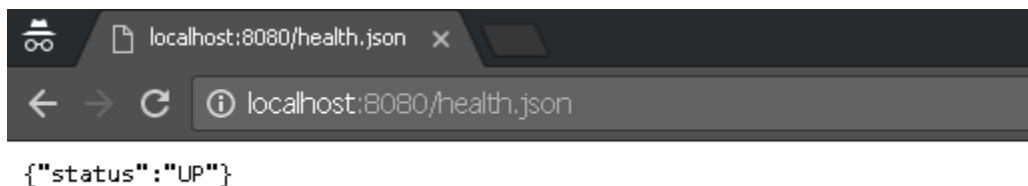
\$ mvn install package

Se iniciará el servidor apache-tomcat y desde el navegador se ingresará a la consola administrativa (Ver paso a paso en el anexo de tomcat) y se realizará el despliegue de la aplicación, para ello se cargará el archivo restful-blog-1.0.war existente en el directorio target del repositorio y se arrancará la aplicación. La prueba de la aplicación se podrá realizar por medio de los endpoints que vienen descritos en la referencia escrita en el proyecto github <https://github.com/benjsicam/restful-blog>.



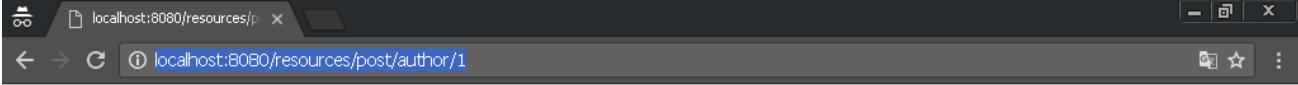
**Ilustración 27 Ubicación del archivo .war desde el explorador de Ms Windows**

Hasta este punto se tendrá ejecutando y arriba el monolito que se convertirá a microservicios. Se podrá verificar la url <http://localhost:8080/health.json> que el monolito se encuentra corriendo en nuestra máquina



**Ilustración 28 Endpoint del microservicio que permite saber el estado de salud del monolito**

Y verificar que la base de datos esté funcionando correctamente accediendo a un servicio como por ejemplo <http://localhost:8080/resources/post/author/1>



```
[{"id":1,"date":"2013-05-07","content":"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus bibendum, lorem sit amet hendrerit consequat, lorem massa pellentesque odio, et dapibus turpis sem ut nibh. Maecenas luctus eros magna. In hac habitasse platea dictumst. Proin nisi nunc, consectetur ac porttitor a, interdum sit amet massa. Phasellus vulputate imperdiet mauris, eu aliquam odio feugiat vel. Duis quis malesuada velit. Ut facilisis sem non ante dapibus tempor. Nulla arcu metus, varius sed laoreet vitae, tempor vel urna. Vestibulum et interdum elit."}, {"id":2,"date":"2013-05-08","content":"Sed id nunc in nisi pellentesque commodo. Donec malesuada, purus volutpat auctor accumsan, ante arcu scelerisque tellus, eget vulputate eros nisl at dolor. Etiam sit amet risus eget risus sodales tincidunt in ac orci. Curabitur suscipit rhoncus urna. Vestibulum ligula neque, vulputate vel vulputate non, luctus nec massa. Mauris ligula dui, rutrum sed condimentum non, dictum vitae risus. Sed eros ligula, auctor vitae porttitor elementum, iaculis non diam. Nullam euismod fermentum mi at pretium. Sed cursus, elit quis pharetra imperdiet, dolor nisi rhoncus neque, ac laoreet ipsum ligula ut elit. Donec sit amet velit mauris, sed dictum lorem. Vestibulum dignissim convallis mattis. Integer libero erat, rhoncus id interdum ac, vulputate vel justo. Aliquam ac est eu orci tincidunt ullamcorper sed sed sapien."}, {"id":3,"date":"2013-05-09","content":"Ut neque purus, convallis eget dictum ut, convallis ut nisi. Aenean auctor laoreet lacus, molestie placerat elit tincidunt sed. Pellentesque ullamcorper, mauris vel lacinia elementum, purus lectus porttitor risus, ac pellentesque felis nisi ut diam. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam rutrum metus sit amet ligula sollicitudin tincidunt. Integer rutrum neque sed odio varius ac luctus turpis varius. Aliquam interdum molestie orci, a vulputate mauris viverra et. Curabitur ipsum mi, facilisis vel venenatis et, scelerisque eu augue. Nulla facilisi."}]
```

**Ilustración 29 Consulta de servicio autores en el monolito**

## Diseño Y Definición de Microservicios

### Modelado de servicios

El proyecto no posee ningún tipo de diagrama pero si una pequeña referencia escrita en la página web de github, de allí se puede inferir que existen 3 tipos de contexto: autores, categorías y publicaciones, de igual manera el blog maneja un sistema de autenticación para el acceso de los recursos lo que obliga a definir un cuarto contexto encargado de esta funcionalidad.

### Blog RESTful Web Service Sample

#### Introduction

This is a very basic sample application written in Java and Spring Framework which demonstrates how to create a RESTful Web Service.

This sample RESTful web service contains simple Blog resources like Author, Post, and Categories. I will be posting a tutorial on how to create RESTful Web Services on my blog [No-nonsense](#).

#### Deployment

This application is already deployed on Cloud Foundry. You can play with it by going to <https://restful-blog.cloudfoundry.com/>.

#### How to use the deployed application

This sample RESTful web service is secured by Spring Security. If asked for credentials just enter admin for user and admin for password. I will also post a tutorial on my blog on securing RESTful web services.

**Ilustración 30 Descripción general del proyecto restfukl-blog obtenida en  
<https://github.com/benjsicam/restful-blog>**

En el siguiente diagrama se representan los contextos anteriormente mencionados donde se puede apreciar una ambigüedad entre los contextos autenticación y propiedad intelectual puesto que los autores y los usuarios de autenticación tienen propiedades en común de acceso para utilizar la aplicación y tiene información personal asociada a la propiedad intelectual, por lo tanto es necesario resolver la ambigüedad entre ambos contextos.

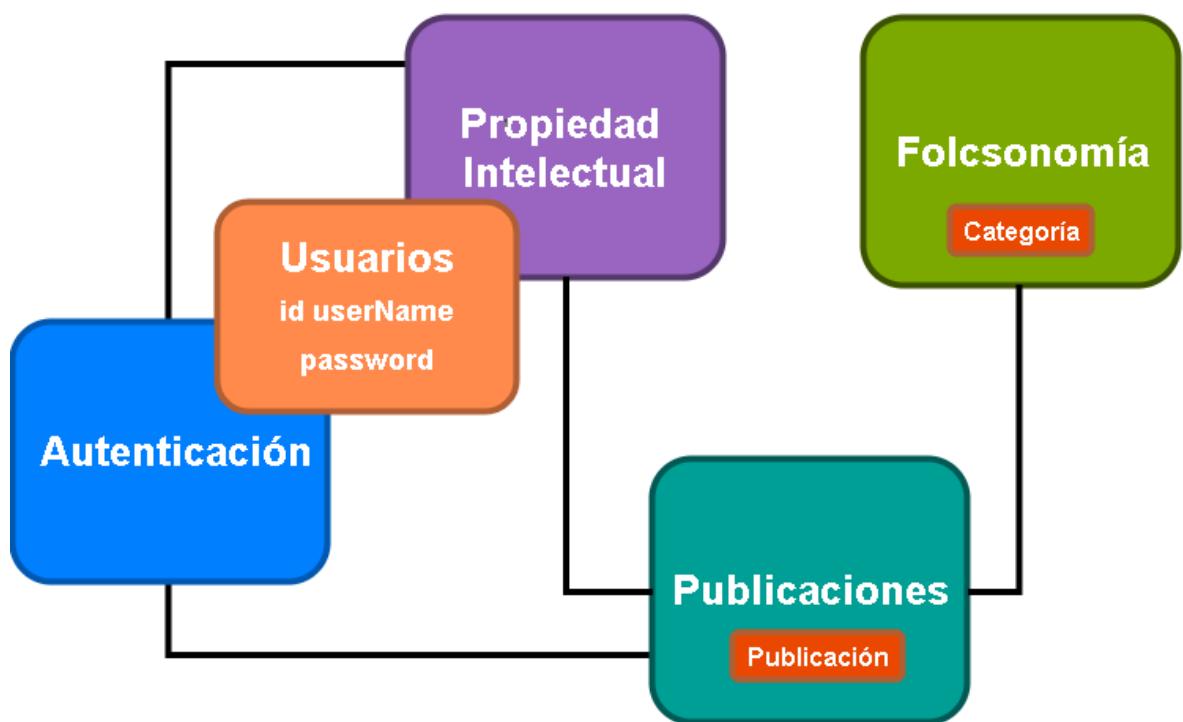


Ilustración 31 Contexto existentes en el monolito

En el siguiente diagrama se resuelve la ambigüedad creando dos nuevos modelos que comparten información asociada a los usuario, desacoplando de esta manera los contextos quedan desacoplados.



Ilustración 32 Desambigüación de los contexto del monolito

## **Fragmentación del monolito**

Todo el código fuente del aplicativo se encuentra concentrado en un solo proyecto, por lo que será necesario dividirlo en cuatro proyectos que proyecten la división de los contextos, para la fragmentación se definirán 5 fases de tal manera que permite hacer una transformación evolutiva, se dividirá de la siguiente manera:

**Fase I Extracción de microservicios de autenticación y propiedad intelectual.** En esta fase se desacoplarán los usuarios y los autores, ambos modelos compartirán propiedades sin embargo cada uno residirá en diferentes códigos base. Ambos microservicios serán extraídos sin embargo seguirán interactuando con el resto del monolito para el resto de funcionalidades del sistema. También se crearán otros servicios que hacen parte de la infraestructura del ecosistema como servicio de descubrimiento, servicio de configuración, servicio de monitoreo y servicio de bitácora.

**Fase II Extracción de microservicios de publicaciones.** En este tiempo intermedio se partirá por segunda vez el monolito para extraer el servicio de publicaciones el cuál se incorporará a la colección de microservicios junto a los dos anteriores

**Fase III Extracción de microservicio de folcosomía y eliminación de monolito.** Finalmente el monolito incorporará su lógica restante a un cuarto microservicio desapareciendo en su totalidad

**Fase IV Despliegue de microservicios en docker.** Se automatizará el despliegue de los microservicios en contenedores haciendo más sencilla su instalación, también se simulará un ambiente similar al producción, puesto que cada microservicio será instalado en una máquina virtual diferente lo que servirá para validar el aislamiento y la comunicación asíncrona de la aplicación.

***Fase V Entrega continua y despliegue de los artefactos de software hasta su salida a producción.*** En esta fase se automatizará la promoción y la publicación de los artefactos de software para en los diferentes ambientes de ejecución y de esta manera preparar diferentes tipos de ambientes como por ejemplo pruebas de errores, validación de requerimientos y producción.

## **Implementación de la fase I**

En la primera fase se realizará la construcción de los dos primeros microservicios : autenticación y autores, también se realizará la preparación del ecosistema de microservicios que permitirá fragmentar el monolito completamente, a continuación se encontrará el paso a paso de como llevarlo acabo, también se podrá encontrar el repositorio git con el resultado de la primera fase que será de utilidad si se desea verlo en cualquier momento <https://github.com/juanwalker/restful-blog-microservices-phase1>

### **Inspección del código fuente de autenticación en el monolito**

Para comenzar se debe de delimitar el contexto de autenticación de tal manera que se permita saber que archivos fuente necesitan ser extraídos para ser llevados al nuevo proyecto de código. Por lo tanto es necesaria una inspección de todo el código fuente del monolito de tal manera que el programador pueda detectar cuales archivos están implicados en la autenticación de los usuarios. A continuación se listan los archivos implicados junto a unas notas técnicas sobre los mismos

Paquete	Clase	Notas
com.benjsicam.restfulblog.service	UserDetailsServiceImpl	Esta clase se encarga de validar las credenciales(userName y password) de las llamadas http, realiza un llamado al método findByUserName en la clase AuthorService.
	AuthorService	Contiene los métodos de consulta de los autores , será necesario fragmentar la base de datos para la extracción de los

		métodos <code>findByName</code> y <code>updatePassword</code> .
com.benjsicam.restfulblog.controller	AuthorController	Contiene todos los endpoint de los servicios rest de los autores, incluyendo el método <code>updatePassword</code> el cual hace parte del contexto de autenticación.

**Tabla 1 Resultados de la inspección del código fuente del contexto credenciales en el monolito**

## Desprendimiento del código fuente del contexto de autenticación en el monolito

Luego de haber detectado los punto de acoplamiento entre el monolito y el contexto de autenticación se procederá a remover del código necesario para desacoplar el nuevo contexto , para ello se removerán los métodos que sirven para consultar un usuario y actualizar su password, a continuación se verán los fragmentos de código en las clases AuthorController y AuthorService que necesitan ser removidos:

```
@Controller
@RequestMapping("/resources/author")
public class AuthorController {

    @Autowired
    private AuthorService authorService;
    ...

    @RequestMapping(value = "/password", method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.OK)
    public void updatePassword(@RequestBody PasswordChangeDTO
                                passwordChangeDTO) {

        String userName =
SecurityContextHolder.getContext().getAuthentication().getName();
        Author author = authorService.findByUsername(userName);
        author.setPassword(passwordChangeDTO.getNewPassword());
        authorService.updatePassword(author);
    }
}
```

Ilustración 34 Fragmento clase AuthorController en restful-blog antes de refactorización

```
@Service
public class AuthorService {

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    private AuthorRepository authorRepository;
    ...
    public Author findByUsername(String username) {
        return authorRepository.findByUsername(username);
    }
    @Transactional
    public void updatePassword(Author author) {
        author.setPassword(passwordEncoder.encode(author.getPassword()));
        authorRepository.saveAndFlush(author);
    }
}
```

Ilustración 33 Fragmento clase AuthorService en restful-blog antes de refactorización

De igual manera se encontró en la clase UserDetailsServiceImpl el llamado que realiza a la base de datos por medio de la clase AuthorService, este llamado tendrá que ser reemplazado por otro que llame al nuevo microservicio.

La tabla Autor contiene información de autenticación que necesita ser manejada en el nuevo microservicio, los campos id y userName necesitan ser replicados, mientras que el password será removido a una nueva tabla de la base de datos. En la imagen de abajo se puede evidenciar la manera como los esquemas antes y después de la transformación.

```
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private AuthorService authorService;

    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Author author = authorService.findByUsername(username);
    }
}
```

Ilustración 35 Fragmento UserDetailsServiceImpl en proyecto restful-blog antes de refactorizar

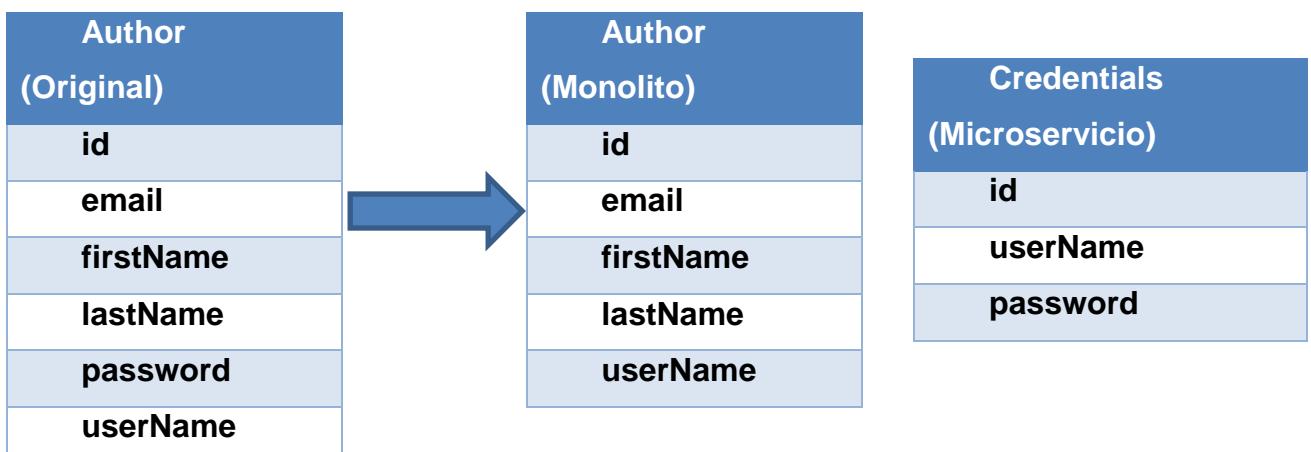


Ilustración 36 Refactorización de tablas para contexto de autenticación

## Creación el servicio de Autenticación

Luego de haber develado los cambios a realizar el monolito, se procederá a crear el nuevo microservicio el cual se llamará proyecto restful-blog-credentials.

**Project Metadata**

Artifact coordinates	Web
Group	<input checked="" type="checkbox"/> <b>Web</b> Full-stack web development with Tomcat and Spring MVC
com.benjsicam.restfulblog	<input type="checkbox"/> <b>Websocket</b> WebSocket development with SockJS and STOMP
Artifact	<input type="checkbox"/> <b>Web Services</b> Contract-first SOAP service development with Spring Web Services
restful-blog-credentials	<input type="checkbox"/> <b>Jersey (JAX-RS)</b> RESTful Web Services framework
Name	<input type="checkbox"/> <b>Ratpack</b> Spring Boot integration for the Ratpack framework
restful-blog-credentials	<input type="checkbox"/> <b>Vaadin</b> Vaadin java web application framework
Description	<input type="checkbox"/> <b>Rest Repositories</b> Exposing Spring Data repositories over REST via spring-data-rest-webmvc
Demo project for Spring Boot	<input type="checkbox"/> <b>HATEOAS</b> HATEOAS-based RESTful services
Package Name	<input type="checkbox"/> <b>Rest Repositories HAL Browser</b> Browsing Spring Data REST repositories in your browser
com.benjsicam.restfulblog	<input type="checkbox"/> <b>Mobile</b> Simplify the development of mobile web applications with spring-mobile
Packaging	<input type="checkbox"/> <b>REST Docs</b> Document RESTful services by combining hand-written and auto-generated documentation
Jar	<input type="checkbox"/> <b>Stormpath</b> Stormpath default starter including Spring MVC, Thymeleaf and Spring Security
Java Version	
1.8	
Language	
Java	

**Ilustración 37 Formulario de start.spring.io para la creación del microservicio restful-blog-credentials**

Es necesario hacer click en el botón Generate Project y descargar el archivo .zip generado en el cual es necesario descomprimir en el directorio d:/microservicios/ejemplo-metodologia-microservicios.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.benjsicam.restfulblog</groupId>
  <artifactId>restful-blog-credentials</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>restful-blog-authentication</name>
  <description>Restful blog credentials microservice</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
    <relativePath/>
  </parent>

  <properties>
    <project.build.sourceEncoding>
      UTF-8
    </project.build.sourceEncoding>
    <project.reporting.outputEncoding>
      UTF-8
    </project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <!-- Spring jpa dependency -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.springframework.boot</groupId>
          <artifactId>
            spring-boot-starter-logging
          </artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <!-- Spring boot dependency -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Security dependencies, http-basic -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- Mysql driver -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <scope>runtime</scope>
    </dependency>
  </dependencies>

```

```

</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<!-- Jackson mapper dependency -->
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.12</version>
</dependency>
</dependencies>

<dependencyManagement>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-dependencies</artifactId>
        <version>Camden.RELEASE</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
</dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

Ilustración 38 Contenido del archivo pom.xml del proyecto resftul-blog-credentials

La anterior configuración permitirá utilizar diferentes tipos de recursos que se irán explicando en la medida se vayan nombrando a lo largo de esta fase. Posteriormente se procederá a modificar la clase RestfulBlogCredentialsApplication.java la cual es clase que funciona como punto de partida para la definición y la ejecución del microservicio, a continuación se hace una descripción breve de las etiquetas más básicas necesarias para la configuración del microservicio.

**@SpringBootApplication:** Es una etiqueta proporcionada por la librería spring boot que permite configurar el proyecto como una aplicación web adicionando automáticamente todas las configuraciones de spring y preparaciones de servidores web que era necesaria hacer manualmente.

**@RestController:** Esta etiqueta declarará la clase como una interfaz tipo fachada para exponer métodos cuando el servicio esté en ejecución

**@RequestMapping:** Permite definir una ruta web base que será utilizada para todas las llamadas de los servicios rest a exponer.

**@Autowire:** Esta etiqueta permite inyectar una instancia de un bean definido en la aplicación web. El microservicio tendrá 3 endpoints que permitirán leer, escribir o actualizar y borrar credenciales asociadas a un autor. En CredentialsService contendrá la lógica correspondiente a las llamadas en la base de datos por medio del repositorio, se hace una transcripción de la lógica existente en el monolito en cada uno de sus métodos.

**@Service:** Es una especialización de un componente tipo bean, la clase se incluyen en el contexto de spring para ser utilizada como inyección de dependencia.

```
@SpringBootApplication
@RestController
@RequestMapping("/resources/credentials")
public class RestfulBlogCredentialsApplication {
    private static final Log logger =
        LogFactory.getLog(RestfulBlogCredentialsApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(
            RestfulBlogCredentialsApplication.class, args);
    }

    @Autowired
    private CredentialsService credentialsService;

    @RequestMapping(value = "/{username}", method = RequestMethod.GET)
    public @ResponseBody
    Credentials loadUserByUsername(@PathVariable("username")
                                    String userName) {
        logger.info("you called loadUserByUsername");
        return credentialsService.getCredentialsByUserName(userName);
    }

    @RequestMapping(value = "/", method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.OK)
    Credentials save(@RequestBody Credentials credentials) {
        logger.info("you called save");
        credentialsService.saveAndFlush(credentials);
        return credentials;
    }
}
```

El método getCredentialsByUserName no solo devuelve las credenciales de la base de datos sino que también asigna el tipo de rol el cual será utilizado para efectos de validación en otros microservicios. El repositorio permite manejar funcionalidades tipo Crud (Create, read, update, delete) para la entidad credentials.

```
@Service
public class CredentialsService {
    @Autowired
    CredentialsRepository credentialsRepository;

    public Credentials getCredentialsByUserName(String userName) {
        Credentials credentials =
            this.credentialsRepository.findByUsername(userName);
        credentials.getRoles().add("ROLE_USER");
        return credentials;
    }

    public void saveAndFlush(Credentials credentials) {
        this.credentialsRepository.saveAndFlush(credentials);
    }

    public void delete(Long id) {
        this.credentialsRepository.delete(id);
    }
}
```

**Ilustración 40 Clase CredentialsService del proyecto restful-blog-credentials**

```
public interface CredentialsRepository extends
JpaRepository<Credentials, Long> {
    @Query
    Credentials findByUsername(String username);
```

**Ilustración 41 Clase CredentialsRepository del proyecto restful-blog-credentials**

```

@Entity
@Table(name = "credentials")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Credentials {

    @Id
    @Column(name = "id")
    private Long id;

    @Column(name = "username", length = 50, nullable = false,
             unique = true)
    private String username;

    @Column(name = "password", length = 50, nullable = false,
             unique = true)
    private String password;

    @Transient
    private List<String> roles;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public List<String> getRoles() {
        if (roles == null){
            roles = new ArrayList<String>();
        }
        return roles;
    }

    public void setRoles(List<String> roles) {
        this.roles = roles;
    }
}

```

Ilustración 42 Clase Credentials del proyecto restful-blog-credentials

En el dominio se ha definido la clase Credentials la cual cargará el id, userName y password asociados a un usuario. Esta información será obtenida desde la tabla credentials en la nueva base de datos del microservicio.

La configuración del microservicio se debe de definir en archivos de propiedades los cuales el contexto de spring carga por defecto, en ellos en ella se pueden definir atributos como el nombre, puerto de escucha y configuración de acceso a la base de datos ambos archivos se escriben utilizando el formato YAML y se guardan dentro del paquete resources:

El primer archivo en ser leído por el contexto de spring es bootstrap.yml y en él se almacenará el nombre del microservicio

```
spring:  
  application:  
    name: credentials
```

Ilustración 43 Archivo bootstrap.yml del proyecto restful-blog-credentials

Posteriormente el segundo archivo en ser leído por el contexto de spring es application.yml

```
server:  
  port: 8181  
spring:  
  datasource:  
    url: jdbc:mysql://localhost/restful_blog_auth  
    username: root  
    password: root  
    driverClassName: com.mysql.jdbc.Driver  
jpa:  
  show-sql : true  
  hibernate:  
    ddl-auto: update  
    naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy  
  properties:  
    hibernate:  
      dialect: org.hibernate.dialect.MySQL5Dialect
```

Ilustración 44 Archivo application.yml del proyecto restful-blog-credentials

Este archivo contendrá la configuración del puerto de escucha y la configuración del cliente de la base de datos.

Nótese que se ha definido en la propiedad spring.datasource.url una nueva base de datos la cual será utilizada por el microservicio, se debe por lo tanto que crear una nueva base de datos en el servidor mysql local para ellos se debe de dirigir al cliente de mysql y desde la consola se ejecutará el siguiente comando

```
CREATE DATABASE restful_blog_auth;
```

Se seleccionará la base de datos

```
USE restful_blog_auth;
```

Y posteriormente se ejecutará el siguiente script

```
CREATE TABLE IF NOT EXISTS `credentials` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8;
```

Ilustración 45 Script de la tabla Credentials para la base de datos restful\_blog\_auth

También es necesario ingresar los datos existentes en el monolito pero con la estructura de la nueva tabla, por lo tanto se debe de ejecutar el siguiente script para insertar dos nuevas filas.

```
INSERT INTO `credentials` (`id`, `username`, `password`) VALUES
(1, 'admin',
 '$2a$10$4nz/1AntXH017MFHPX1R5.m3YEAhVICaJtrQUP87ZvGr1dEIKNyPq'),
(2, 'user',
 '$2a$10$NtDZRpRKw190x5QR/oNCMuW5Ff6yx4klaw.Qd5PP2/i8DXjZzqHaG');
```

Ilustración 46 Script de los valores de la tabla Credentials para la base de datos restful\_blog\_auth

En este punto se puede ejecutar el microservicio para verificar que esté funcionando correctamente. Ubicados en el la ruta del proyecto en una línea de comandos se ejecutará la rutina

```
gradlew bootRun
```

Este comando permitirá arrancar el microservicio, luego de haber terminado la inicialización desde un navegador web y se ingresará a la url <http://localhost:8181/resources/credentials/admin>, se ingresarán las credenciales que se han definido para utilizar el api, usuario authUser y contraseña KEQ9bx5p. Debe de aparecer el siguiente resultado:



**Ilustración 47 Obtención de las credenciales del usuario admin desde el microservicio credentials**

Se procederá ahora a configurar una autenticación básica de http, tal como lo hace el monolito, para que solo clientes autenticados puedan hacer llamado a los métodos. La dependencia spring-boot-starter-security trae por defecto la autenticación http básica activada por lo tanto solo es necesario personalizar los métodos de autenticación que el microservicio utilizará para que sean usado como validadores de acceso. La anotaciones **@EnableWebSecurity @Configuration** permitirán definir y utilizar configuración que se definan en la clase SecurityConfiguration. La configuración utiliza el mismo encriptador de passwords utilizado en el monolito, verifica que los accesos solo puedan ser concedido a aquellos clientes con rol ROLE\_SERVICE y el sistema de autenticación se tratará por medio

de la clase UserDetailsServiceImpl, usada de manera similar en el monolito, que se encargará de hacer la validación de las credenciales del cliente (usuario y password) .

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends
WebSecurityConfigurerAdapter {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Autowired
    private UserDetailsService userDetailsService;
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/resources/**")
            .access("hasAnyRole('ROLE_SERVICE')")
            .and().httpBasic().and().csrf().disable();
    }
    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
        throws Exception {
        PasswordEncoder encoder = new BCryptPasswordEncoder();
        auth.userDetailsService(userDetailsService).
            passwordEncoder(encoder);
    }
}
```

Ilustración 48 Clase SecurityConfiguration del proyecto restful-blog-auth

```
@SuppressWarnings("deprecation")
@Component("userDetailsService")
public class UserDetailsServiceImpl implements UserDetailsService {

    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException{
        if(!username.equals("authUser")) {
            throw new UsernameNotFoundException
("Username or password is invalid.");
        } else{
            Collection<GrantedAuthority> authorities =
            new ArrayList<GrantedAuthority>();
            authorities.add(new SimpleGrantedAuthority("ROLE_SERVICE"));
            return new User("authUser",
"$2a$10$kEmC0j7JDihHsG2boXy94.e6m39tGffIz2x/luw9kn4JZVu.BqHxm"
,true, true, true, authorities);
        }
    }
}
```

Ilustración 49 Clase UserDetailsServiceImpl del proyecto restful-blog-auth

Ahora que se ha preparado el nuevo microservicio se debe de configurar el monolito para utilice el nuevo servicio Restful de autenticación para la consulta de la contraseña de un usuario.

```
@Service
public class CredentialClient{

    @Autowired
    private ApplicationContext applicationContext;

    @Autowired
    SpringCloudConfigClient springCloudConfigClient;

    public Credentials findByUsername(String userName) {

        RestTemplate restTemplate = new RestTemplate();
        HttpEntity<String> request = new HttpEntity<String>(buildHeaders());
        return restTemplate
            .exchange("http://localhost:8181/resources/credentials/"
            .concat("/").concat(String.valueOf(userName)),
            HttpMethod.GET, request, Credentials.class).getBody();
    }

    private HttpHeaders buildHeaders(){
        byte[] plainCredsBytes = "authUser:KEQ9bx5p".getBytes();
        byte[] base64CredsBytes = Base64.encodeBase64(plainCredsBytes);
        String base64Creds = new String(base64CredsBytes);
        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", "Basic " + base64Creds);
        return headers;
    }
}
```

Ilustración 50 Clase CredentialsClient del proyecto restful-blog

En la clase UserDetailsServiceImpl del monolito se reemplazará el código existente por el siguiente que permita consultar las credenciales por medio de una llamada rest:

```
@SuppressWarnings("deprecation")
@Component("userDetailsService")
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private CredentialClient credentialService;

    @Autowired SpringCloudConfigClient springCloudConfigClient;

    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {

        if (username.equals(serviceAuthUser.getUser())) {

            Collection<GrantedAuthority> authorities =
                new ArrayList<GrantedAuthority>();
            authorities.add(new SimpleGrantedAuthority("ROLE_SERVICE"));
            return new User("KEQ9bx5p",
                "$2a$10$kEmC0j7JDihHsG2boXy94.e6m39tGffIz2x/luw9kn4JZVu.BqHxm",
                true, true, true, authorities);

        } else{
            Credentials credentials =
                credentialService.findByUsername(username);
            if(credentials == null) {
                throw new UsernameNotFoundException(
                    "Username or password is invalid.");
            }
            Collection<GrantedAuthority> authorities =
                new ArrayList<GrantedAuthority>();
            for(String role: credentials.getRoles()){
                authorities.add(new GrantedAuthorityImpl(role));
            }
            return new User(credentials.getUsername(),
                credentials.getPassword(),
                true, true, true, authorities);
        }
    }
}
```

Ilustración 51 Refactorización de la clase UserDetailsServiceImpl en el proyecto restful-blog

Para verificar el cambio será necesario volver a compilar el proyecto y redesplegar el .war generado en el servido apache-tomcat

## Inspección del código fuente de autores en el monolito

Para comenzar se debe de delimitar el contexto autenticación de tal manera que se permita saber que archivos fuente necesitan ser extraídos para ser llevados al nuevo proyecto de código. Por lo tanto es necesaria una inspección de todo el código fuente del monolito de tal manera que el programador pueda detectar cuales archivos están implicados en la autenticación de los usuarios. A continuación se listan los archivos implicados junto a una notas técnicas sobre los mismos.

Paquete	Clase	Notas
com.benjsicam.restfulblog.service	AuthorService	Contiene los métodos de consulta de la información de los autores, toda la clase deberá de ser trasladada al nuevo microservicio
com.benjsicam.restfulblog.controller	AuthorController	Contiene todos los endpoint de los servicios rest de los autores, la clase también será trasladada en su totalidad al microservicio
com.benjsicam.restfulblog.dao	AuthorRepository	Interfaz utilizada para las llamadas JPA hacia la base de datos. Deberá ser trasladada al nuevo microservicios
com.benjsicam.restfulblog.domain	Author,Post,Category	Clases de dominio, la clase Author se deberá de transcribir con todas las anotaciones JPA en tanto que en las otras será necesario remover las anotaciones JPA

**Tabla 2 Resultados de la inspección del código fuente del contexto author en el monolito**

## Desprendimiento del código fuente del contexto de propiedad intelectual en el monolito

A continuación se listan los fragmentos de código fuente asociado al contexto Authors

```
@Controller
@RequestMapping("/resources/author")
public class AuthorController {
    @Autowired
    private AuthorService authorService;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(value = HttpStatus.CREATED)
    public void create(@RequestBody Author entity) {

        authorService.create(entity);
    }
    @RequestMapping(method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public void update(@RequestBody Author entity) {

        authorService.update(entity);
    }
    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.OK)
    public void delete(@PathVariable("id") Long id) {

        authorService.delete(id);
    }
    @RequestMapping(method = RequestMethod.GET)
    public @ResponseBody List<Author> find() {

        return authorService.find();
    }
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public @ResponseBody Author findAuthor(@PathVariable("id") Long id) {

        return authorService.findById(id);
    }
    @RequestMapping(value = "/{id}/posts", method = RequestMethod.GET)
    public @ResponseBody List<Post> findAuthorPosts(@PathVariable("id")
Long id ) {

        return authorService.findAuthorPosts(id);
    }
}
```

Ilustración 52 Clase AuthorController en el proyecto restful-blog

```

@Service
public class AuthorService {

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    private AuthorRepository authorRepository;

    @Autowired
    private PostRepository postRepository;

    @Transactional
    public void create(Author author) {

        author.setPassword(passwordEncoder.encode(author.getPassword()));
        authorRepository.saveAndFlush(author);
    }

    @Transactional
    public void update(Author author) {
        author = this.findById(author.getId());

        author.setFirstName(author.getFirstName());
        author.setLastName(author.getLastName());

        authorRepository.saveAndFlush(author);
    }

    @Transactional
    public void delete(Long id) {
        authorRepository.delete(id);
    }

    public List<Author> find() {
        return authorRepository.findAll();
    }

    public Author findById(Long id) {
        return authorRepository.findOne(id);
    }

    public List<Post> findAuthorPosts(Long id) {
        return
postRepository.findByAuthor(authorRepository.findOne(id));
    }

    public Author findByUsername(String username) {
        return authorRepository.findByUsername(username);
    }
}

```

Ilustración 53 Clase AuthorService en el proyecto restful-blog

La tabla Autor deberá de ser removida en su totalidad y trasladada a la nuevas base de datos del microservicio, no tendrá ninguna alteración en su estructura

<b>Author</b>
<b>(Monolito)</b>
<b>id</b>
<b>email</b>
<b>firstName</b>
<b>lastName</b>
<b>userName</b>

**Ilustración 54 Tabla Author luego de refactorización del contexto credenciales**

## Creación del servicio de autores

Es necesario ingresar a la siguiente url <http://start.spring.io/> y crear un nuevo proyecto maven con la última versión de spring-boot y dependencias web, que contendrán el código fuente removido del monolito asociado con el manejo de los autores.

Project Metadata		Web
Artifact coordinates		<input checked="" type="checkbox"/> <b>Web</b> Full-stack web development with Tomcat and Spring MVC
Group	com.benjsicam.restfulblog	<input type="checkbox"/> <b>Websocket</b> Websocket development with SockJS and STOMP
Artifact	restful-blog-author	<input type="checkbox"/> <b>Web Services</b> Contract-first SOAP service development with Spring Web Services
Name	restful-blog-author	<input type="checkbox"/> <b>Jersey (JAX-RS)</b> RESTful Web Services framework
Description	Demo project for Spring Boot	<input type="checkbox"/> <b>Ratpack</b> Spring Boot integration for the Ratpack framework
Package Name	com.benjsicam.restfulblog	<input type="checkbox"/> <b>Vaadin</b> Vaadin java web application framework
Packaging	Jar	<input type="checkbox"/> <b>Rest Repositories</b> Exposing Spring Data repositories over REST via spring-data-rest-webmvc
Java Versión	1.8	<input type="checkbox"/> <b>HATEOAS</b> HATEOAS-based RESTful services
Language	Java	<input type="checkbox"/> <b>Rest Repositories HAL Browser</b> Browsing Spring Data REST repositories in your browser
		<input type="checkbox"/> <b>Mobile</b> Simplify the development of mobile web applications with spring-mobile
		<input type="checkbox"/> <b>REST Docs</b> Document RESTful services by combining hand-written and auto-generated documentation

Ilustración 55 Creación de microservicio restful-blog-author

Es necesario hacer click en el botón Generate Project y descargar el archivo .zip generado en el cual es necesario descomprimir en el directorio d:/microservicios/*ejemplo-metodologia-microservicios*. Los proyectos generados vienen por defecto empaquetados en apache maven por lo que se debe de abrir desde IntelliJ el archivo pom.xml el cual se debe de modificar con las siguientes dependencias:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.benjsicam.restfulblog</groupId>
  <artifactId>restful-blog-author</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>restful-blog-author</name>
  <description>Restful blog author microservice</description>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>
  <!-- Spring boot dependency -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Security dependencies, http-basic -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- Mysql driver -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  
```

```

</dependency>
<!-- Jackson mapper dependency -->
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.12</version>
</dependency>
<!-- Feing client-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Brixton.SR6</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

**Ilustración 56 Archivo pom.xml del proyecto restful-blog-author**

La clase RestfulBlogAuthorApplication es la clase principal del microservicio y contendrá los servicios web REST de los autores, estos servicios serán los mismos que se encontraban en la clase AuthorController del monolito.

```

@SpringBootApplication
@RestController
@ComponentScan
@RequestMapping("/resources/author")
public class RestfulBlogAuthorApplication {

    private static final Log logger =
LogFactory.getLog(RestfulBlogAuthorApplication.class);
    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogAuthorApplication.class, args);
    }
    @Autowired
    private AuthorService authorService;
    @Autowired
    private PasswordEncoder passwordEncoder;

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(value = HttpStatus.CREATED)
    public void create(@RequestBody Author entity) {
        authorService.create(entity);
        logger.info("you called create");
    }

    @RequestMapping(method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public void update(@RequestBody Author entity) {
        authorService.update(entity);
        logger.info("you called update");
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.OK)
    public void delete(@PathVariable("id") Long id) {
        authorService.delete(id);
        logger.info("you called delete");
    }

    @RequestMapping(method = RequestMethod.GET)
    public @ResponseBody
    List<Author> find() {
        logger.info("you called find");
        return authorService.find();
    }
}

```

```

@RequestMapping(value = "/{id}", method = RequestMethod.GET)
public @ResponseBody Author findAuthor(@PathVariable("id") Long id)
{
    logger.info( "you called findAuthor");
    return authorService.findById(id);
}

@RequestMapping(value = "{authorId}/posts", method =
RequestMethod.GET)
public @ResponseBody List<Post>
findAuthorPosts(@PathVariable("authorId") Long authorId ) {
    logger.info("you called findAuthorPosts");
    return authorService.findAuthorPosts(authorId);
}

@RequestMapping(value = "/password", method = RequestMethod.POST)
@ResponseStatus(HttpStatus.OK)
public void updatePassword(@RequestBody PasswordChangeDTO
passwordChangeDTO) {
    String userName = SecurityContextHolder.getContext()
                    .getAuthentication().getName();
    Author author = authorService.findByUsername(userName);
    author.setPassword(passwordChangeDTO.getNewPassword());
    authorService.updatePassword(author);
    logger.info( "you called updatePassword");
}
}

```

**Ilustración 57 Clase RestfulBlogAuthorApplication del proyecto restful-blog-authors**

La clase AuthorService se encargará de incluir los métodos existentes de la misma clase par existente en el monolito, tendrá acceso directo con la base de datos del microservicio. Los cambios significativos se dio los métodos updatePassword y delete los cual realiza un llamados a los servicios rest *save* y *delete* del microservicio *credentials* y , también se modificó el método *findAuthorPosts* el cual consulta el método *findById* en el microservicio *authors*, ambas llamadas se hacen por medio de clientes web que hacen uso de la librería de feign el cual se describirá más abajo.

```
@Service
public class AuthorService {

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    PostsClientService postsClientService;

    @Autowired
    private AuthorRepository authorRepository;

    @Autowired
    private CredentialsClientService credentialsClientService;

    @Transactional
    public void create(Author author) {
        author.setPassword(passwordEncoder.encode(author.getPassword()));
        authorRepository.saveAndFlush(author);
        this.saveAuthorCredentails(author);
    }

    @Transactional
    public void update(Author author) {
        if (this.findById(author.getId()) != null) {

author.setPassword(passwordEncoder.encode(author.getPassword()));
            authorRepository.saveAndFlush(author);
            this.saveAuthorCredentails(author);
        }
    }

    @Transactional
    public void delete(Long id) {
        authorRepository.delete(id);
        credentialsClientService.deleteCredentials(id);
    }

    public Author findById(Long id) {
```

```

        return authorRepository.findOne(id);
    }

    public List<Post> findAuthorPosts(Long authorId ) {
        return postsClientService.findByAuthor(authorId);
    }
    public Author findByUsername(String userName) {
        return authorRepository.findByUsername(userName);
    }

    public Author getDefaultAuthorUsername(String userName) {
        return new Author();
    }
    @Transactional
    public void updatePassword(Author author) {

author.setPassword(passwordEncoder.encode(author.getPassword()));
        this.saveAuthorCredentails(author);
    }

    private void saveAuthorCredentails(Author author){
        Credentials credentials = new Credentials();
        credentials.setId(author.getId());
        credentials.setUsername(author.getUsername());
        credentials.setPassword(author.getPassword());
        credentialsClientService.saveCredentials(credentials);
    }

}

```

**Ilustración 58 Clase AuthorService del proyecto restful-blog-author**

La clase Author se transcribe de la misma manera como se encuentra en el monolito, con la excepción de la propiedad password que ya no tendrá la etiqueta **@column** sino que será una propiedad volátil, osea no persistible en base de datos sino únicamente cargada en memoria, por lo que tendrá asociada la etiqueta **@transient**.

Las demás clases que hacen parte del dominio como Author,Category,Credentials,PasswordChangeDTO y Post se copia igual que las originales y posteriormente suprimiendo todas las etiquetas asociadas a la persistencia de la base de datos y pertenecientes al paquete de java javax.persistence.\* como **@column**, **@id**, **@GeneratedValue**, **@Entity** y **@Table**, **@OneToMany** **@ManyToMany** etc...

```

...
public class Author {

    @Transient
    private String password;
...

```

**Ilustración 59 Fragmento de la clase Author del proyecto restful-blog-author**

La clase AuthorRepository permitirá almacenar la información en la tabla Author en la base de datos local del microservicio.

```

public interface AuthorRepository extends JpaRepository<Author,
Long>{
    Author findByUsername(String username);
}

```

**Ilustración 60 Clase AuthorRepository del proyecto restful-blog-author**

Para la comunicación entre microservicios se han creado los clientes de servicio REST, que son utilizados por la clase AuthorService , haciendo uso de la librería de Feing el cual funciona como un frontend para un cliente web, en este caso se utiliza ribbon el cual viene por defecto en spring boot.

```

@FeignClient(name="http://localhost:8181", configuration =
FeignClientConfiguration.class)
public interface CredentialsClientService {
    @RequestMapping(method = RequestMethod.POST,
                    value="/resources/credentials/",
                    produces = MediaType.APPLICATION_JSON_VALUE,
                    consumes = MediaType.APPLICATION_JSON_VALUE)
    void saveCredentials(@RequestBody Credentials credentials);

    @RequestMapping(method = RequestMethod.GET,
                    value="/resources/credentials/{userName}")
    Credentials loadCredentialsByUserName(@PathVariable("userName")
                                            String userName);

    @RequestMapping(method = RequestMethod.DELETE,
                    value="/resources/credentials/{id}")
    void deleteCredentials(@PathVariable("id") Long userId);
}

```

**Ilustración 61 Clase CredentialsClientService del proyecto restful-blog-author**

```

@FeignClient(name="http://localhost:8080/", configuration =
FeingClientConfiguration.class)
public interface PostsClientService {

    @RequestMapping(method = RequestMethod.GET,
                    value="/resources/post/author/{authorId}")
    List<Post> findByAuthor(@PathVariable("authorId") Long authorId);
}

```

**Ilustración 62 Clase PostsClientService del proyecto restful-blog-author**

Debido a que el microservicio *credentials* y el monolito hacen uso de la autenticación http básica para la ejecución de sus llamadas, es necesario definir una configuración general que incluya el manejo de credenciales para las llamadas http.

```

package com.benjsicam.restfulblog.client;

import feign.auth.BasicAuthRequestInterceptor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FeingClientConfiguration {

    @Bean
    public BasicAuthRequestInterceptor basicAuthRequestInterceptor() {
        return new BasicAuthRequestInterceptor("authUser", "KEQ9bx5p");
    }
}

```

**Ilustración 63 Clase FeingClientConfiguration del proyecto restful-blog-author**

Las clases SecurityConfiguration y UserDetailsServiceImpl también se utilizan aquí de la misma manera que las utilizaba el microservicio de credenciales, con la diferencia que la clase UserDetailsServiceImpl ahora utiliza el cliente feing para la consulta de las credenciales al servicio rest loadCredentialsByUserName, quedando de la siguiente manera

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    private static final Logger log =
LoggerFactory.getLogger(UserDetailsServiceImpl.class);

    @Autowired
    private CredentialsClientService credentialsClientService;
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

        if (username.equals(serviceAuthUser)){
            Collection<GrantedAuthority> authorities =
                new ArrayList<GrantedAuthority>();
            authorities.add(new SimpleGrantedAuthority("ROLE_SERVICE"));
            return new User("authUser",
                "$2a$10$kEmC0j7JDihHsG2boXy94.e6m39tGffIz2x/luw9kn4JZVu.BqHxm",
                true, true, true, true, authorities);
        }else{
            Credentials credentials = credentialsClientService
                .loadCredentialsByUserName(username);
            if(credentials == null) {
                throw new UsernameNotFoundException(
                    "Username or password is invalid.");
            }
            return this.createUserDetails(credentials);
        }
    }

    private UserDetails createUserDetails(Credentials credentials){
        Collection<GrantedAuthority> authorities =
            new ArrayList<GrantedAuthority>();

        credentials.getRoles().stream().forEach(role -> authorities
            .add(new SimpleGrantedAuthority(role)));

        return new User(credentials.getUsername(),
            credentials.getPassword(), true, true,
            true, true, authorities);
    }
}
```

Ilustración 64 Clase UserDetailsServiceImpl del proyecto restful-blog-author

En el monolito se modificará también la manera como los datos del autor eran obtenidos cuando se llamaba el método `getPostAuthor`, se creará un cliente más básico que el creado con Feing que permita llamar el microservicio desde el monolito.

```
@Controller
@RequestMapping("/resources/post")
public class PostController {
    @Autowired
    private AuthorClient authorClient;

    ...

    @RequestMapping(value = "/{id}/author", method =
RequestMethod.GET)
    public @ResponseBody
    Author getPostAuthor(@PathVariable("id") Long id) {
        Long postAuthorId = postService.findPostAuthor(id);
        logger.info("findByIdUserId called");
        return this.authorClient.findById(postAuthorId);
    }
}
```

**Ilustración 65 Fragmento de la clase PostController del proyecto restful-blog**

La clase `AuthorClient` es casi exactamente igual a la de `credentialClient` con la diferencia que incluye el método `findByIdUserId` encargado de retornar un autor a partir del `id` que le ingresen.

El siguiente paso es llevar acabo configuración de la aplicación en los archivos `bootstrap.yml` y `application.yml` para la definición de su nombre, puerto y conexión a la base de datos:

```
spring:
  application:
    name: author
```

**Ilustración 66 Archivo bootstrap.yml del proyecto restful-blog-author**

```

server:
  port: 8282

spring:
  datasource:
    url: jdbc:mysql://localhost/restful_blog_author
    username: root
    password: root
    driverClassName: com.mysql.jdbc.Driver
  jpa:
    show-sql : true
    hibernate:
      ddl-auto: update
      naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5Dialect

```

**Ilustración 67 Archivo application.yml del proyecto restful-blog-author**

```

@Service
public class AuthorClient{

  @Autowired
  private ApplicationContext applicationContext;

  public Author findByUserId(Long id) {

    RestTemplate restTemplate = new RestTemplate();
    HttpEntity<String> request = new
    HttpEntity<String>(buildHeaders());
    return
    restTemplate.exchange("http://localhost:8282/resources/author/".
      concat("/").concat(String.valueOf(id)),
      HttpMethod.GET, request,
    Author.class).getBody();
  }
  private HttpHeaders buildHeaders(){
    byte[] plainCredsBytes = "authUser:KEQ9bx5p".getBytes();
    byte[] base64CredsBytes =
    Base64.encodeBase64(plainCredsBytes);
    String base64Creds = new String(base64CredsBytes);
    HttpHeaders headers = new HttpHeaders();
    headers.add("Authorization", "Basic " + base64Creds);
    return headers;
  }
}

```

**Ilustración 68 Clase AuthorClient del proyecto restful-blog**

Finalmente es necesario crear la nueva base de datos para el servicio en el servidor mysql de la máquina local, se ejecutará el siguiente comando

```
CREATE DATABASE restful_blog_author;
```

Se seleccionará la base de datos

```
USE restful_blog_author;
```

Y posteriormente se ejecutará el siguiente script

```
CREATE TABLE IF NOT EXISTS `author` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `username` varchar(45) DEFAULT NULL,
  `firstname` varchar(50) DEFAULT NULL,
  `lastname` varchar(50) DEFAULT NULL,
  `email` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8;
```

**Ilustración 69 Script de la tabla Author para la base de datos restful-blog-author**

Se insertarán los dos registros iniciales en la base de datos:

```
INSERT INTO `author` (`id`, `username`, `firstname`, `lastname`,
`email`) VALUES
(1, 'admin', 'admin', 'admin', 'admin@benjsicam.me'),
(2, 'user', 'Juan', 'dela Cruz', 'info@benjsicam.me');
```

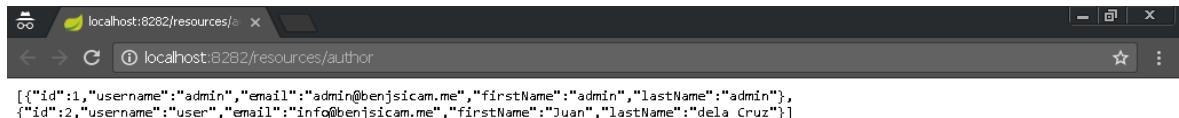
**Ilustración 70 Valores de inserción para la tabla Authors**

Se procederá a verificar el correcto funcionamiento del microservicio, es necesario ubicarse en el la ruta del proyecto desde una línea de comandos y se ejecutará la siguiente rutina

```
gradlew bootRun
```

Este comando permitirá arrancar el microservicio, luego de haber terminado la inicialización se abrirá un navegador web y luego se ingresará a la url

<http://localhost:8282/resources/author> , luego es necesario ingresar las credenciales de admin, usuario admin y contraseña admin. Debe de aparecer el siguiente resultado:



A screenshot of a web browser window titled "localhost:8282/resources/author". The address bar shows the URL. The content area displays a JSON array with two elements:

```
[{"id":1,"username":"admin","email":"admin@benjsicam.me","firstName":"admin","lastName":"admin"}, {"id":2,"username":"user","email":"info@benjsicam.me","firstName":"Juan","lastName":"dela Cruz"}]
```

**Ilustración 71 Consulta de autores al microservicio Authors**

## Creación del servicio y de los clientes del descubrimiento

Hasta el momento se han creado dos nuevos microservicios a partir del monolito, todos preparados para ser ejecutados en la máquina local sin embargo en caso que se desee instalar los aplicativo en otro ambiente como docker, pruebas o producción es necesario de un mecanismo que permita localizar los accesos a los servicios sin conocer la ip. Para dar solución a este inconveniente se creará un servidor Eureka donde tanto los microservicios como el monolito se registrarán por medio de un cliente para habilitar un redireccionamiento dinámico sin utilizar la ip.

Es necesario ingresar a <http://start.spring.io/> y se generará un nuevo proyecto para el servidor Eureka el cual se llamará restful-blog-eureka con el mismo id de grupo que los demás proyectos

## Project Metadata

Artifact coordinates

Group

Artifact

Name

Description

Package Name

Packaging

Java Version

Language

## Cloud Discovery

Eureka Discovery

Service discovery using spring-cloud-netflix and Eureka

Eureka Server

spring-cloud-netflix Eureka Server

Zookeeper Discovery

Service discovery with Zookeeper and spring-cloud-zookeeper-discovery

Cloud Foundry Discovery

Service discovery with Cloud Foundry

Consul Discovery

Service discovery with Hashicorp Consul

Ilustración 72 Formulario de start.spring.io para la creación del proyecto restful-blog-eureka

Se procederá a descargar el archivo y descomprimirlo en la carpeta ejemplo-metodología-microservicios , desde intellj se abrirá el proyecto y en el paquete resources y se creará el archivo application.yml el cual contendrá los siguientes valores:

Se abrirá el archivo RestfulBlogEurekaApplication y se incluirá la anotación **@EnableEurekaServer** de tal manera que habilite el servidor Eureka en el microservicio.

```

@SpringBootApplication
@EnableEurekaServer
public class RestfulBlogEurekaApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogEurekaApplication.class, args);
    }
}

```

**Ilustración 73 Configuración de la clase RestfulBlogEurekaApplication**

El microservicio escuchará por defecto en el puerto 8761 y lo se está configurando de tal manera para que no interactúe ni registrándose el mismo ni que tampoco obtenga los microservicios registrados para su propio funcionamiento

```

server:
  port: 8761
eureka:
  instance:
    hostname: localhost
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone:
        http://${eureka.instance.hostname}:${server.port}/eureka/

```

**Ilustración 74 Modificación del archivo application.yml del proyecto restful-blog-eureka**

Sin embargo en ambos microservicios se debe de configurar el cliente de tal manera que ambos puedan interactuar con eureka. Se abrirán los archivos application.yml resources de ambos proyectos y se incluirá la siguiente propiedad:

```
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8761/eureka/
```

Ilustración 75 Configuración de cliente eureka en los archivos application.yml

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-eureka</artifactId>  
</dependency>
```

Ilustración 76 Dependencia de cliente de eureka necesaria en el pom.xml de cada microservicio

También es necesario como dependencia en los archivos pom.xml de cada proyecto incluir el cliente de Eureka para que lo se pueda utilizar: Finalmente, se puede reemplazar las urls quemadas en los clientes de Feing del microservicio restful-blog-por los nombres de las aplicaciones ya que se hará uso del servidor de eureka para el descubrimiento entre los microservicios

```
...  
  
  @FeignClient(name="credentials", configuration  
  FeingClientConfiguration.class)  
  public interface CredentialsClientService {  
  
  ...
```

Ilustración 77 Reemplazo de urls por alias de eureka en la clase CredentialsClientService

```

    ...
    @FeignClient(name="monolith", configuration =
FeingClientConfiguration.class)
public interface PostsClientService {
    ...

```

En el caso del monolito es diferente el uso de eureka debido a que el microservicio no

**Ilustración 78 Reemplazo de urls por alias de eureka en la clase PostsClientService**

dispone de un cliente como tal que le permita, por lo tanto las url siguen estando declaradas en el código y será necesario el uso de variables de entorno que se definen en cada ambiente. Por ello se procederá a cambiar los clientes de los microservicios del monolito adicionando el soporte de variables de entorno que permitan asignación de url para substituir el descubrimiento de microservicios.

```

@Service
public class AuthorClient{

    ...
    private String getAuthorBase() {
        String fooResourceUrl;
        if (System.getenv("AUTHOR_PORT_8282_TCP_ADDR") != null)
            String host =
        System.getenv("AUTHOR_PORT_8282_TCP_ADDR").concat(":8282");
            fooResourceUrl=
        "http://".concat(host).concat("/resources/author/");
        }else{
            fooResourceUrl=
        "http://localhost:8282/resources/author/";
        }
        return fooResourceUrl;
    }
}

```

**Ilustración 79 Uso de variables de sistema para nombres de microservicios en la clase AuthorClient**

```
@Service
public class CredentialClient{
    private String getCredentialsBase() {
        String fooResourceUrl;
        if (System.getenv("CREDENTIALS_PORT_8181_TCP_ADDR") != null)
            String host = System.getenv("CREDENTIALS_PORT_8181_TCP_ADDR")
                .concat(":8181");
            fooResourceUrl= "http://".concat(host)
                .concat("/resources/credentials/");
        }else{
            fooResourceUrl= http://localhost:8181/resources/credentials/";
        }
        return fooResourceUrl;
    }
}
```

**Ilustración 80 Uso de variables de sistema para nombres de microservicios en la clase CredentialClient**

A pesar que el monolito no está en capacidad de interactuar con el servicio de eureka, se puede de cierta manera levantar un microservicio que funcione como cliente para el descubrimiento del monolito en la red.

Las librerías spring-cloud-sidecar permiten levantar un servicio intermediario que verifica que el monolito esté arriba y a su vez se encarga de notificárselo a eureka. Para comenzar se bajará un nuevo proyecto desde <http://start.spring.io/> desde donde se seleccionarán algunas dependencias que necesitan de sidecar para funcionar, se creará un nuevo proyecto llamado *restful-blog-sidecar* con grupo *com.benjsicam.restfulblog* sin ninguna dependencia en particular.

## Project Metadata

Artifact coordinates

Group

com.benjsicam.restfulblog

Artifact

restful-blog-sidecar

Name

restful-blog-sidecar

Description

Demo project for Spring Boot

Package Name

com.benjsicam.restfulblog

Packaging

Jar

Java Version

1.8

Language

Java

**Ilustración 81 Formulario de start.spring.io para la creación del proyecto  
restful-blog-sidecar**

Se descargará el archivo y se descomprimirá en el workspace de microservicios, posteriormente se abrirá desde intelliJ y se modificará el pom de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.benjsicam.restfulblog</groupId>
  <artifactId>restful-blog-sidecar</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>restful-blog-sidecar</name>
  <description>Sidecar of restful blog</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
    <relativePath/> 
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-zuul</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>
```

```

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-netflix-sidecar</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Brixton.SR6</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

**Ilustración 82 Archivo pom.xml del proyecto restful-blog-sidecar**

Se tiene que habilitar la etiqueta **@EnableSidecar** en la clase principal del proyecto de tal manera que se vea así:

```

@SpringBootApplication
@EnableSidecar
public class RestfulBlogSidecarApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogSidecarApplication.class, args);
    }
}

```

**Ilustración 83 Clase RestfulBlogSidecarApplication del proyecto restful-blog-sidecar**

Luego se debe de modificar los archivos de configuración del microservicio, el archivo bootstrap.yml tendrá la ruta al servidor eureka

```
spring:
  application:
    name: monolith

  eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

Ilustración 84 Archivo bootstrap.yml del proyecto restful-blog-sidecar

En tanto que el archivo application.yml tendrá la configuración del puerto, y la ruta del servicio web encargado de verificar el estado de salud del monolito.

```
server:
  port: 9090

sidecar:
  port: 8080
  health-uri: http://localhost:${sidecar.port}/health.json
```

Ilustración 85 Archivo application.yml del proyecto restful-blog-sidecar

Para probar el funcionamiento de Eureka y de SideCar es necesario detener todos los microservicios y el monolito de tal manera que el primer microservicio en subir sea Eureka. Luego de haber detenido todos los microservicios desde un línea de comandos se ingresará a la ruta del proyecto de eureka y lo se iniciará por medio del comando `./gradlew bootRun`, luego de terminar la ejecución del microservicio se ejecutarán los demás microservicios con el comando `bootrun` y desde donde es necesario redesplegar nuevamente el archivo .war del monolito en tomcat e iniciar nuevamente los microservicios.

Para verificar la ejecución de eureka se puede ingresar desde el navegador la url <http://localhost:8761/> y allí se podrá evidenciar que los demás microservicios se hayan registrado

The screenshot shows a browser window titled "Eureka" with the URL "localhost:8761". The page header includes the Spring logo and the word "Eureka". On the right, there are links for "HOME" and "LAST 1000 SINCE STARTUP".

**System Status**

Environment	test
Data center	default

Current time	2017-02-10T14:08:39 -0500
Uptime	04:04
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	6

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**

**DS Replicas**

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones	Status
AUTHOR	n/a (1)	(1)	UP (1) - The-Colombia-PC:author:8282
CREDENTIALS	n/a (1)	(1)	UP (1) - The-Colombia-PC:credentials:8181
MONOLITH	n/a (1)	(1)	UP (1) - The-Colombia-PC:monolith:9090

Ilustración 86 Interfáz gráfica del servicio eureka

## Creación del servicio y de los clientes de la configuración

Entre las librerías de spring cloud disponibles existe una que permite poner archivos de propiedades remotos por microservicio, spring cloud config se levanta como un servicio el cual se conecta a un repositorio desde donde se leen los archivos de propiedades y los expone como servicios rest.

Se ingresará a <http://start.spring.io/> y se creará un nuevo proyecto llamado restful-blog-configuration con grupo com.benjsicam.restfulblog y del cual se incluirá la opción de config-server.

Project Metadata		Cloud Config
Artifact coordinates		
Group	com.benjsicam.restfulblog	<input type="checkbox"/> Config Client spring-cloud-config Client
Artifact	restful-blog-configuration	<input checked="" type="checkbox"/> Config Server Central management for configuration via a git or svn backend
Name	restful-blog-configuration	<input type="checkbox"/> Zookeeper Configuration Configuration management with Zookeeper and spring-cloud-zookeeper-config
Description	Demo project for Spring Boot	<input type="checkbox"/> Consul Configuration Configuration management with Hashicorp Consul
Package Name	com.benjsicam.restfulblog	
Packaging	Jar	
Java Version	1.8	
Language	Java	

**Ilustración 87 Formulario de start.spring.io para la creación del proyecto restful-blog-configuration**

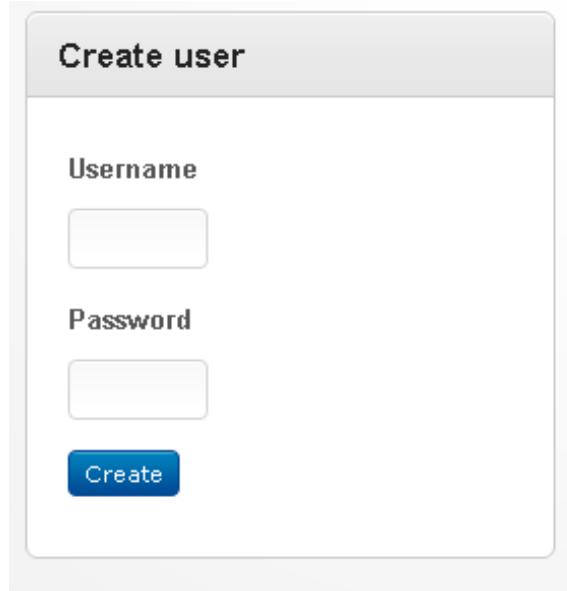
Se abrirá la clase RestfulBlogConfigurationApplication y se incluirá la etiqueta @EnableConfigServer la cual permitirá iniciar el servidor de configuración cuando se arranque el microservicio.

```
@EnableConfigServer
@SpringBootApplication
public class RestfulBlogConfigurationApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogConfigurationApplication.class, args);
    }
}
```

**Ilustración 88 Clase RestfulBlogConfigurationApplication del proyecto restful-blog-configuration**

Es importante para iniciar haber versionado los archivos de configuración que se van a utilizar en el servidor de configuración, se necesita crear un nuevo repositorio en el servidor GitStack previamente instalado y corriendo (Referirse al anexo final), se ingresará a la página web <http://localhost/gitstack/> y desde allí se creará un nuevo usuario desde la interfaz Users



**Ilustración 89 Formulario de creación de usuarios de gitstack**

Luego se debe de dirigir a la interfaz repositories y se cree un nuevo repositorio llamado restful\_blog\_configuration

**Repositories**

Name	Command to clone	Action
restful_blog_configuration	git clone http://localhost/restful_blog_configuration.git	

**Create repository**

**Name**

The name of your repository

**Create**

Ilustración 90 Formulario de creación de repositorio de gitstack

Además se deben de conceder permisos al usuario para leer y modificar el repositorio que se creó

## restful\_blog\_configuration permissions

Add and remove user access to this repository

**Users**

Name	Read	Write	Action
juanwalker	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
everyone	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

**Add user**

Ilustración 91 Formulario de asignación de usuario a repositorios en gitstack

Posteriormente se procederá a crear una carpeta localizada en la ruta D:\microservicios\ejemplo-metodologia-microservicios\repositorios\restful\_blog\_configuration\_local y se abrirá una consola de git ubicada en la misma ruta, luego se ejecutarán los comandos en el siguiente orden:

```
git clone http://localhost/restful\_blog\_configuration.git
```

esto permitirá clonar el repositorio en el ambiente local, después se crearán 3 archivos credential.properties, author.properties y monolith.properties, los tres archivos tendrá el siguiente contenido:

```
basic-authentication.user=authUser  
basic-authentication.password=KEQ9bx5p  
  
basic-authentication.password-  
encoded=$2a$10$kEmC0j7JDihHsG2boXy94.e6m39tGffIz2x/luw9kn4JZVu.  
BqHxm
```

**Ilustración 92 Contenidos de archivos de propiedades del proyecto restful-blog-configuration**

Las cuales son credenciales para el uso de las api entre los microservicios.

Posteriormente se ejecutará en la consola

```
git add *  
  
git config --global user.name "Su nombre"  
  
git config --global user.email su@correo.com  
  
git commit -m "First commit"  
  
git push origin master
```

**Ilustración 93 Comandos git para hacer push en un repositorio remoto**

Luego de haber realizar commit de los archivos se procederá a implementar los clientes del spring cloud configuration y el uso de sus propiedades. Se comenzará por adicionar la ruta de spring cloud server en el archivo bootstrap.yml de los servicios *credentials* y *authors*

```
cloud:  
config:  
uri: http://localhost:8888
```

**Ilustración 94 Archivo bootstrap.yml del proyecto restful-blog-configuration**

Después de haber configurado el servidor en cada microservicio se procederá a utilizar las propiedades y eliminar las cadenas donde aparecen quemadas en código las contraseñas.

En los proyectos *restful-blog-authors* y *restful-blog-credentials* debe de incluirse las propiedades serviceAuthUser y serviceAuthPassword en la clase UserDetailsServiceImpl las propiedades serviceAuthUser y serviceAuthPassword y reemplazar la contraseña quemadas existentes como cadenas de texto.

```
@Service  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
    @Value("${basic-authentication.user}")  
    String serviceAuthUser;  
  
    @Value("${basic-authentication.password-encoded}")  
    String serviceAuthPassword;  
  
    ...  
    return new User(serviceAuthUser, serviceAuthPassword, true, true);  
}
```

**Ilustración 95 Modificación de clase UserDetailsServiceImpl para la utilización de propiedades**

**existentes en spring cloud config**

En el proyecto *restful-blog-authors*:

```
@Configuration  
public class FeingClientConfiguration {  
  
    @Value("${basic-authentication.user}")  
    String serviceAuthUser;  
  
    @Value("${basic-authentication.password}")  
    String serviceAuthPassword;  
  
    ...  
}
```

```
public class ServiceAuthUser {
```

**Ilustración 96 Modificación de clase FeingClientConfiguration para la utilización de propiedades existentes en spring cloud config**

```
    -  
    encodedPassword) {  
        this.user = user;  
        this.password = password;  
        this.encodedPassword= encodedPassword;  
    }  
    public String getUser() {  
        return user;  
    }  
    public void setUser(String user) {  
        this.user = user;  
    }  
    public String getPassword() {  
        return password;  
    }  
    public void setPassword(String password) {  
        this.password = password;  
    }  
    public String getCredentials(){  
        return this.user.concat(":").concat(this.password);  
    }  
    public String getEncodedPassword() {  
        return encodedPassword;  
    }  
    public void setEncodedPassword(String encodedPassword) {  
        this.encodedPassword = encodedPassword;  
    }  
}
```

En el proyecto *restful-blog* debido a que no se tiene un cliente de *spring cloud configuration* se debe de crear uno propio de tal manera que se puedan obtener las propiedades y utilizarlas dentro del monolito, para este propósito se creo la clase *Spring CloudConfigClient* la cual genera un objeto del tipo *ServiceAuthUser*.

En las clases AutorClient, CredentialClient y UserDetailsServiceImpl se inyecta el cliente SpringCloudConfigClient desde donde se podrán acceder a las propiedades del spring cloud config

**Ilustración 97 Clase ServiceAuthUser del proyecto restful-blog**

```

@Service
public class SpringCloudConfigClient{
    @Autowired
    private ApplicationContext applicationContext;
    public ServiceAuthUser getServiceAuthUser() {
        RestTemplate rest = new RestTemplate();
        HttpEntity<String> request = new HttpEntity<String>(new HttpHeaders());
        return buildServiceAuthUser(rest.getForObject(getConfigurationBase(), String.class));
    }
    private ServiceAuthUser buildServiceAuthUser(String str) {
        ObjectMapper objectMapper = new ObjectMapper();
        try {
            JsonNode node = objectMapper.readValue(str, JsonNode.class);
            JsonNode userNode = node.get("propertySources")
                .get(0).get("source").get("basic-authentication.user");
            JsonNode passwordNode = node.get("propertySources").get(0).
                get("source").get("basic-authentication.password");
            JsonNode encodedPasswordNode = node.get("propertySources").get(0)
                .get("source").get("basic-authentication.password-encoded");
            return new ServiceAuthUser(userNode.getTextValue(), passwordNode.getTextValue(),
                encodedPasswordNode.getTextValue());
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    private String getConfigurationBase() {
        String fooResourceUrl;
        if (System.getenv("SPRING_PROFILES_ACTIVE") != null) {
            String host = System.getenv("CONFIGURATION_PORT_8888_TCP_ADDR").concat(":8888");
            fooResourceUrl= "http://".concat(host).concat("/monolith/development/master/");
        }else{
            fooResourceUrl= "http://localhost:8888/monolith/development/master";
        }
        return fooResourceUrl;
    }
}

```

Ilustración 98 Clase SpringCloudConfigClient del proyecto restful-blog

```

public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private CredentialClient credentialService;

    @Autowired SpringCloudConfigClient springCloudConfigClient;

    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        ServiceAuthUser serviceAuthUser =
springCloudConfigClient.getServiceAuthUser();
        ...
        return new User(serviceAuthUser.getUser(),
serviceAuthUser.getEncodedPassword(),
true, true, true, true, authorities);
        ...
    }
}

```

**Ilustración 99 Inyección de la clase SpringCloudConfigClient en la clase UserDetailsServiceImpl de restful-blog**

```

@Service
public class AuthorClient{
    @Autowired
    SpringCloudConfigClient springCloudConfigClient;
    ...

    private HttpHeaders buildHeaders(){
        byte[] plainCredsBytes =
springCloudConfigClient.getServiceAuthUser().
getCredentials().getBytes();
        ...
    }
}

```

**Ilustración 100 Inyección de la clase SpringCloudConfigClient en la clase AuthorClient de restful-blog**

```

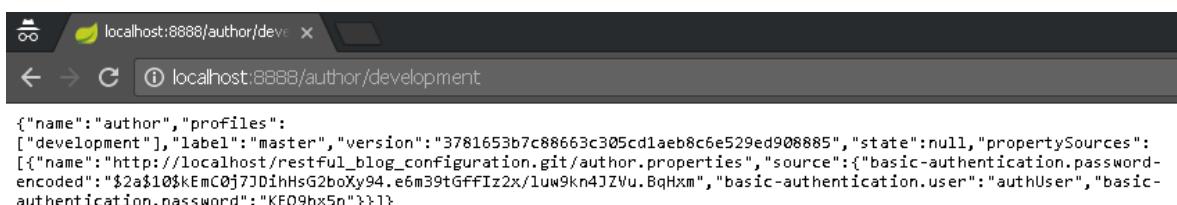
@Service
public class CredentialClient{
    @Autowired
    SpringCloudConfigClient springCloudConfigClient;
    ...

    private HttpHeaders buildHeaders() {
        byte[] plainCredsBytes =
        springCloudConfigClient.getServiceAuthUser() .
            getCredentials().getBytes();
        ...
    }
}

```

**Ilustración 101 Inyección de la clase SpringCloudConfigClient en la clase CredentialClient de restful-blog**

Para probar el funcionamiento del servidor de configuración bastará con reiniciar todos los microservicios y ejecutar el comando `./gradlew bootRun` por línea de comandos y entrar al navegador desde donde se verificará que las propiedades se están exponiendo. Para realizar la prueba se ingresará a la siguiente url <http://localhost:8888/author/development> y debería de imprimir las propiedades del archivo `author.properties`



**Ilustración 102 Verificación de la exposición de las propiedades en el servicio de configuración**

## Creación del servicio y clientes de la bitácora

La bitácora o log es el mecanismo tradicional y de uso común en todos los componentes de la aplicación, que registra las acciones, fallos, procedimientos realizado en tiempo de ejecución permite a la personas de soporte poder hacer trazabilidad de los eventos o procedimientos realizados por una aplicación, en una aplicación monolítica el manejo de la bitácora se maneja centralizada y fuertemente acoplada entre todos los módulos, sin embargo en una aplicación distribuida para remover el acoplamiento debe de utilizarse una comunicación orientada a mensajes (Productor-consumidor). Spring cloud dispone de unas librerías denominadas spring cloud bus las cuales permiten comunicación entre sus componentes por medio de un broker, el broker funciona de intermedio para que los servicio suscriptores puedan dejar sus mensajes a los servicios suscriptores.

Spring cloud bus utilizar como broker RabbitMQ el cual debe ejecutarse de manera externa a los microservicios (Ver Anexo de RabbitMQ), se comenzará primero por crear los servicios productores en este caso serian los proyectos restful-blog-credentials y restful-blog-author.

En ambos archivos pom.xml de ambos proyectos es necesario incluir las modificar una dependencia e incluir otras más:

```
...
<!-- Spring jpa dependency -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
...
```

```
<!-- Spring cloud bus dependency -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<!-- Spring log4jv2 dependency -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

Ilustración 103 Dependencias log4j2 para proyectos restful-blog-credentials y restflu-blog-author

Adicionalmente se debe de copiar en ambos proyectos dentro del paquete resources el archivo log4j2.xml de tal manera que todo lo relacionado a logger sea enviado al exchange spring-

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="STDOUT" target="SYSTEM_OUT">
            <PatternLayout pattern="[${level} %c{2} - %msg %d{yyyy-MM-dd HH:mm:ss.SSS} ${n}" />
        </Console>
        <RabbitMQ name="rabbitmq">
            host="${env:RABBITMQ_PORT_5672_TCP_ADDR:-127.0.0.1}"
            port="5672" user="guest" password="guest" virtualHost="/"
            exchange="spring-boot-exchange" declareExchange="true"
            durable ="true" autoDelete="false"
            applicationId="testAppId"
            routingKeyPattern="log4j.routing.key"
            exchangeType="fanout"
            contentEncoding="UTF-8" generateId="false"
            deliveryMode="NON_PERSISTENT"
            charset="UTF-8" contentType="text/plain"
            senderPoolSize="2" maxSenderRetries="30">
                <PatternLayout pattern="[${level} %c{2} - %msg %d{yyyy-MM-dd HH:mm:ss.SSS} ${n}" />
            </RabbitMQ>
        </Appenders>
        <Loggers>
            <Logger name="com.benjsicam.restfulblog" level="info">
                <AppenderRef ref="rabbitmq" />
            </Logger>
            <Root>
                <AppenderRef ref="STDOUT" />
            </Root>
        </Loggers>
    </Configuration>
```

Ilustración 104 Archivo de configuración de log4j2.xml para los proyectos restful-blog-credentials y restful-blog-authors

Es necesario ahora crear el servicio consumidor que estará encargado de leer la información cargada en rabbitMQ y persistirla en una base de datos. Se ingresará al sitio web <http://start.spring.io/> y se creará un nuevo proyecto que se llamará restful-blog-log del grupo com.benjsicam.restfulblog

## Project Metadata

Artifact coordinates

Group

com.benjsicam.restfulblog

Artifact

restful-blog-log

Name

restful-blog-log

Description

Demo project for Spring Boot

Package Name

com.benjsicam.restfulblog

Packaging

Jar

Java Version

1.8

Language

Java

Ilustración 105 Formulario para la creación del proyecto restful-blog-log

Se importará el proyecto desde IntelliJ y se abrirá el archivo pom.xml y se modificará de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.benjsicam.restfulblog</groupId>
  <artifactId>restful-blog-log</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
```

```

<name>restful-blog-log</name>
<description>Demo project for Spring Boot</description>

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.3.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

Ilustración 106 Archivo pom.xml del proyecto restful-blog-log

En la clase RestfulBlogLogApplication se configurará la conexión con RabbitMq y la preparación del recibidor encargado de persistir los mensajes provenientes de la cola.

```
@SpringBootApplication
public class RestfulBlogLogApplication {
    final static String queueName = "spring-boot";
    final static String exchangeName = "spring-boot-exchange";
    @Bean
    Queue queue() {
        return new Queue(queueName, false);
    }
    @Bean
    FanoutExchange exchange() {
        return new FanoutExchange(exchangeName);
    }
    @Bean
    Binding binding(Queue queue, FanoutExchange exchange) {

        return BindingBuilder.bind(queue).to(exchange);
    }
    @Bean
    SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
                                              MessageListenerAdapter listenerAdapter) {
        SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.setQueueNames(queueName);
        container.setMessageListener(listenerAdapter);
        return container;
    }
    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }
    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogLogApplication.class, args);
    }
}
```

Ilustración 107 Clase RestfulBlogLogApplication del proyecto restful-blog-log

```

@Component
public class Receiver {

    @Autowired
    private LogRepository repository;

    public void receiveMessage(String message) {
        System.out.println(message);
        repository.save(new Log(message));
    }
}

```

**Ilustración 108 Receiver del proyecto restful-blog-log**

También es necesaria la creación de una nueva base de datos para la persistencia de los mensajes, se ejecutará el siguiente comando para la creación de la nueva base de datos

```

@Repository
public interface LogRepository extends JpaRepository<Log, Long> {
}

```

**Ilustración 109 Clase LogRepository del proyecto restful-blog-log**

*CREATE DATABASE restful\_blog\_log;*

Se seleccionará la base de datos

*USE restful\_blog\_log;*

Y posteriormente se ejecutará el siguiente script

```

CREATE TABLE IF NOT EXISTS `log` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `message` varchar(21000) DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=204 DEFAULT CHARSET=utf8;

```

**Ilustración 110 Script de tabla log para la base de datos restful\_blog\_log**

Para verificar el funcionamiento del Logger es necesario reiniciar todos los servicios e ingresar a la ruta del proyecto restful-blog-log el cual lo se iniciará por medio del comando `./gradlew bootRun`, luego de terminar su ejecución se ingresará a la url <http://localhost:8282/resources/author> el cual provocará registro de trazas en el microservicios authors y también en credentials.

Si se examina la tabla Log en la base de datos restful\_blog\_log se verá en el registro de las llamadas `loadUserByUserName` de la clase `RestfulBlogCredentialsApplication` y `find` de la clase `restfulBlogAuthorApplication` lo que indica que el mensaje se registró de manera exitosa

206	[INFO]	restfulblog.RestfulBlogAuthorApplication	- you called	find	2017-02-10 15:26:09.985					03:26 p.m.	

205	[INFO]	restfulblog.RestfulBlogCredentialsApplication	- you called	loadUserByUsername	2017-02-10 15:26:09.779					10/02/2017	
-----	--------	---	--------------	--------------------	-------------------------	--	--	--	--	------------	--

**Ilustración 111 Filas sql generadas en el momento en que se ejecuta el llamado al servicio authors**

## Implementación de patrón cortocircuito en los microservicios

El patrón cortacircuitos permite tener un mayor control en las llamadas que se realicen entre microservicios, pudiendo tener la opción de rutinas de programación que permitan manejar fallos en tiempo de ejecución. Se realizará la implementación de los cortocircuitos en los microservicios author y credentials.

Como primer paso es necesario incluir el cliente hystrix para el soporte de los cortocircuitos en los pom.xml de ambos proyectos.

```
<!-- Hytrix client -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

**Ilustración 112 Dependencia a adicionar en el archivo pom.xml de microservicios credentials y authors para soportar cortocircuitajes**

En el microservicio de credentials no hace ningún llamado remoto a excepción de la consulta a la base de datos, se creará un cortocircuitaje en caso que la base de datos no esté disponible.

```
@Service
public class CredentialsService {
    @Autowired
    CredentialsRepository credentialsRepository;
    @HystrixCommand(fallbackMethod = "getDefaultCredentials")
    public Credentials getCredentialsByUserName(String userName) {
        Credentials credentials = this.credentialsRepository.
            findByUsername(userName);
        credentials.getRoles().add("ROLE_USER");
        return credentials;
    }
    public Credentials getDefaultCredentials(String userName) {
        return new Credentials();
    }
    ...
```

**Ilustración 113 Modificación de la clase CredentialsService en el proyecto restful-blog-credentials para soporte de cortocircuitajes en el proyecto restful-blog-authors**

Para el microservicio Authors, se añadirá cortocircuitajes para los métodos getDefaultAuthors, getDefaultAuthorId, getDefaultPosts, getDefaultAuthorUsername. Cada método tendrá un método que se activará en caso de un error o excepción como se aprecia en el código fuente:

```
@Service
public class AuthorService {

    ...
    @HystrixCommand(fallbackMethod = "getDefaultAuthors")
    public List<Author> find() {
        return authorRepository.findAll();
    }
    public List<Author> getDefaultAuthors() {
        return new ArrayList<Author>();
    }
    @HystrixCommand(fallbackMethod = "getDefaultValue")
    public Author findById(Long id) {
        return authorRepository.findOne(id);
    }

    public Author getDefaultValue(Long id) {
        return new Author();
    }

    @HystrixCommand(fallbackMethod = "getDefaultPosts")
    public List<Post> findAuthorPosts(Long authorId) {
        return postsClientService.findByAuthor(authorId);
    }
    public List<Post> getDefaultPosts(Long authorId) {
        return new ArrayList<Post>();
    }

    ...
    @HystrixCommand(fallbackMethod = "getDefaultValue")
    public Author findByUsername(String userName) {
        return authorRepository.findByUsername(userName);
    }

    public Author getDefaultValue(String userName) {
        return new Author();
    }
}
```

**Ilustración 114 Modificación de la clase AuthorService en el proyecto restful-blog-credentials para soporte de cortocircuitajes en proyecto restful-blog-authors**

## Creación del servicio y clientes del monitoreo

El servicio de monitoreo permite analizar el tráfico de llamadas de un microservicio o un cluster de microservicios para realizar el monitoreo es necesario de un dashboard de hystrix que capture toda la información de los demás microservicios. Se creará un nuevo proyecto para este propósito el cual se denominará restful-blog-monitor con el grupo com.benjsicam.restfulblog, se añadirá la dependencia del dashboard de hystrix,el cliente de eureka y las librerías de turbine.

Project Metadata		Cloud Circuit Breaker
Artifact coordinates		<input type="checkbox"/> Hystrix Circuit breaker with spring-cloud-netflix Hystrix
Group	com.benjsicam.restfulblog	<input checked="" type="checkbox"/> Hystrix Dashboard Circuit breaker dashboard with spring-cloud-netflix Hystrix
Artifact	restful-blog-monitor	<input checked="" type="checkbox"/> Turbine Circuit breaker metric aggregation using spring-cloud-netflix with Turbine and server-sent events
Name	restful-blog-monitor	<input type="checkbox"/> Turbine Stream Circuit breaker metric aggregation using spring-cloud-netflix with Turbine and Spring Cloud Stream (choose a specific Stream binder implementation to complement this)
Description	Demo project for Spring Boot	
Package Name	com.benjsicam.restfulblog	<input checked="" type="checkbox"/> Eureka Discovery Service discovery using spring-cloud-netflix and Eureka
Packaging	Jar	<input type="checkbox"/> Eureka Server spring-cloud-netflix Eureka Server
Java Version	1.8	<input type="checkbox"/> Zookeeper Discovery Service discovery with Zookeeper and spring-cloud-zookeeper-discovery
Language	Java	<input type="checkbox"/> Cloud Foundry Discovery Service discovery with Cloud Foundry
		<input type="checkbox"/> Consul Discovery Service discovery with Hashicorp Consul

Ilustración 115 Formulario de start.spring.io para la creación del proyecto restful-blog-monitor

Se descargará el proyecto y se descomprimirá en el workspace, es necesario habilitar tanto el cliente de eureka como el el hystrix board y turbine para que cuando levante el servicio estos muestren correctamente la información .La clase RestfulBlogMonitorApplication tendrá las anotaciones asociadas a los 3 servicios.

```

@SpringBootApplication
@EnableEurekaClient
@EnableHystrixDashboard
@EnableTurbine
public class RestfulBlogMonitorApplication {

    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogMonitorApplication.class,
args);
    }
}

```

**Ilustración 116 Clase RestfulBlogMonitorApplication del proyecto restful-blog-monitor**

Posteriormente se modificará el archivo bootstrap.yml con el nombre del microservicio y el archivo application.yml con el puerto, dirección de eureka y configuración de clusters

```

spring:
  application:
    name: sample-monitor

```

**Ilustración 117 Archivo bootstrap.yml del proyecto restful-blog-monitor**

```

server:
  port: 8383

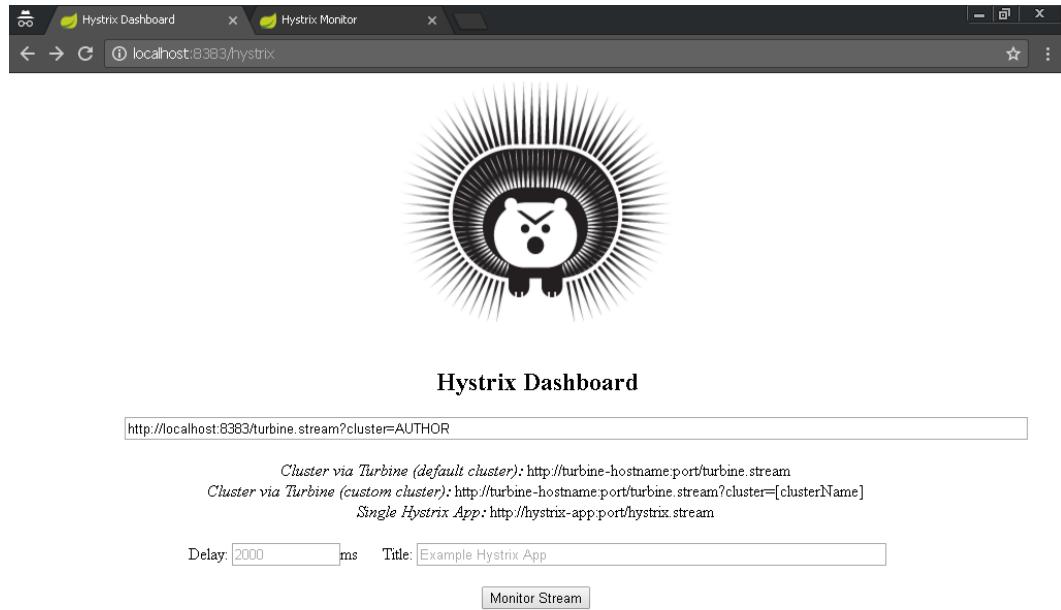
turbine:
  appConfig: author,credentials
  aggregator:
    clusterConfig: AUTHOR,CREDENTIALS

eureka:
  client:
    region: default
    registryFetchIntervalSeconds : 5
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/

```

**Ilustración 118 Archivo application.yml del proyecto restful-blog-monitor**

Para la ejecución del dashboard se puede iniciar el microservicio utilizando el comando `./gradlew bootRun` en la línea de comandos y entrando a la url en la máquina local <http://localhost:8383/hystrix> y desde allí se puede entrar a los cluster que se define en el microservicio: AUTHOR y CREDENTIALS. En ellos se pondrán ver las estadísticas de las llamadas exitosas, erróneas y cortocircuito efectuadas en cada microservicio.



**Ilustración 119 Página principal del microservicio hystrix**

El cluster CREDENTIALS lo se podrá encontrar en la siguiente url

<http://localhost:8383/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A8383%2Fturbine.stream%3Fcluster%3DCREDENTIALS>

<http://localhost:8383/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A8383%2Fturbine.stream%3Fcluster%3DCREDENTIALS>

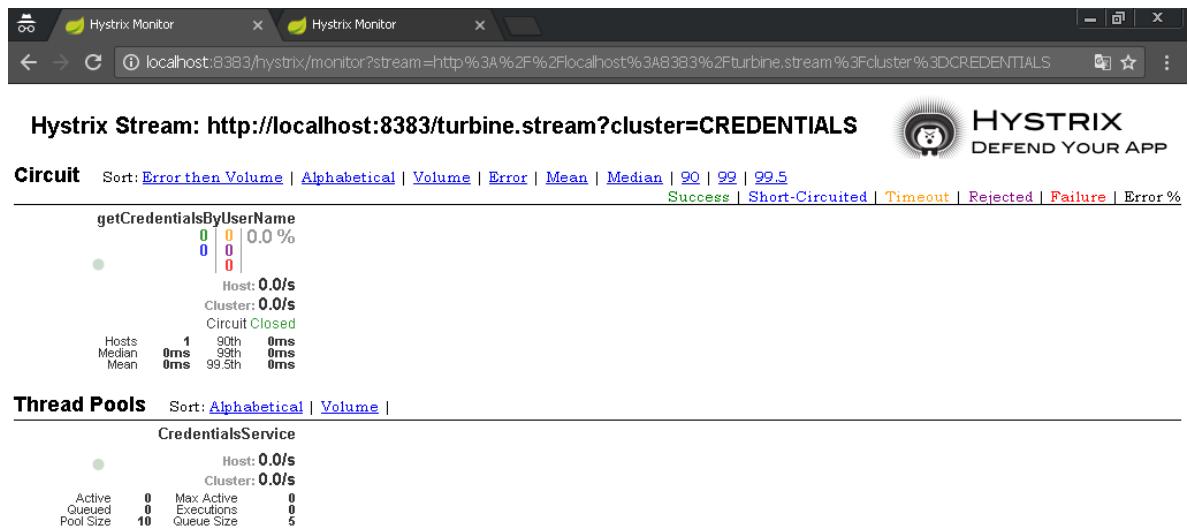


Ilustración 120 Estadísticas en hystrix del cluster CREDENTIALS

El cluster AUTHOR lo se podrá verificar en la siguiente url

<http://localhost:8383/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A8383%2Fturbine.stream%3Fcluster%3DAUTHOR>

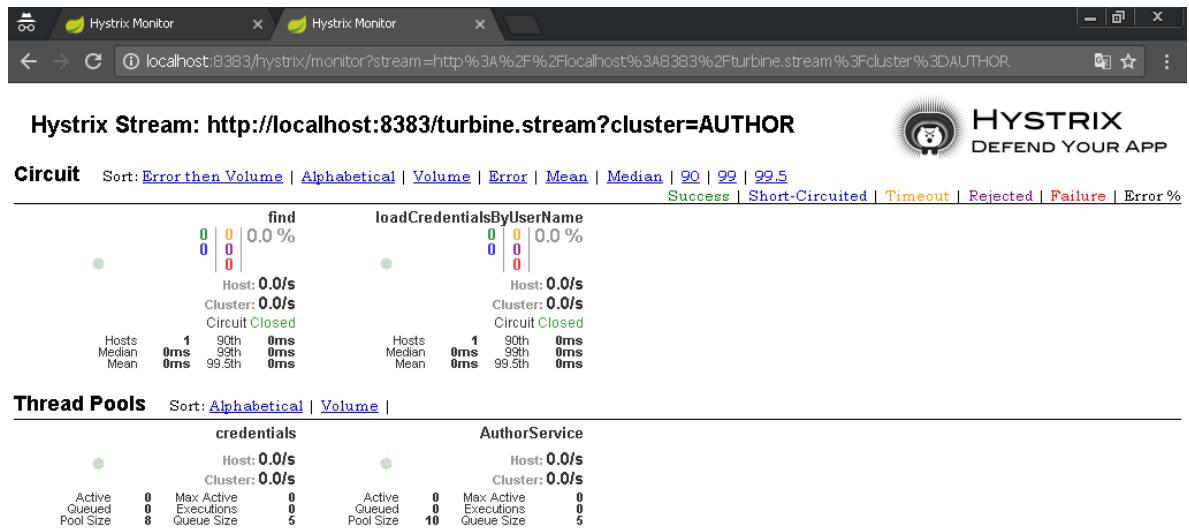


Ilustración 121 Estadísticas en hystrix del cluster AUTHOR

## Implementación de la fase II

En esta segunda fase se fragmentará el contexto de publicaciones y se creará un nuevo microservicio el cual interactuará con los demás microservicios y el monolito, el código fuente final de esta segunda fase puede ser revisado en el siguiente repositorio:  
<https://github.com/juanwalker/restful-blog-microservices-phase2>

### Inspección del código fuente de autenticación en el monolito

Se procede a inspeccionar la base de código del monolito para encontrar el código fuente relacionado con el contexto de publicaciones. A continuación se listan los archivos asociados:

Paquete	Clase	Notas
com.benjsicam.restfulblog.service	PostService	Contiene los métodos de consulta de la información de las publicaciones, toda la clase deberá de ser trasladada al nuevo microservicio
com.benjsicam.restfulblog.controller	PostController	Contiene todos los endpoint de los servicios rest de las publicaciones, la clase también será trasladada en su totalidad al microservicio
com.benjsicam.restfulblog.dao	PostRepository	Interfaz utilizada para las llamadas JPA hacia la base de datos. Deberá ser trasladada al nuevo microservicio
com.benjsicam.restfulblog.domain	Post	Clases de dominio, la clase Post se deberá de transcribir con todas las anotaciones JPA en tanto que en las otras será necesario remover las anotaciones JPA

Tabla 3 Resultados de inspección de código del contexto publicaciones

## Desprendimiento del código fuente del contexto de publicaciones en el monolito

A continuación se listan los fragmentos de código fuente asociado al contexto Posts

```
@Service
public class PostService {

    @Autowired
    private PostRepository postRepository;

    @Transactional
    public void create(Post post) {
        postRepository.saveAndFlush(post);
    }

    @Transactional
    public void update(Post post) {
        postRepository.saveAndFlush(post);
    }

    @Transactional
    public void delete(Long id) {
        postRepository.delete(id);
    }

    public List<Post> find() {
        return postRepository.findAll();
    }

    public Post findById(Long id) {
        return postRepository.findOne(id);
    }

    public Author findPostAuthor(Long id) {
        return postRepository.findPostAuthor(id);
    }

    public List<Post> findPostsByCategoryId(Long id) {
        return postRepository.findPostsByCategoryId(id);
    }
}
```

Ilustración 122 Clase PostService del proyecto restful-blog-posts

```

@Controller
@RequestMapping("/resources/post")
public class PostController {

    @Autowired
    private PostService postService;

    @Autowired
    private CategoryService categoryService;

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(value = HttpStatus.CREATED)
    public void create(@RequestBody Post post) {

        postService.create(post);
    }

    @RequestMapping(method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public void update(@RequestBody Post post) {

        postService.update(post);
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.OK)
    public void delete(@PathVariable("id") Long id) {

        postService.delete(id);
    }

    @RequestMapping(method = RequestMethod.GET)
    public @ResponseBody List<Post> find() {

        return postService.find();
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public @ResponseBody Post findPost(@PathVariable("id") Long id) {

        return postService.findById(id);
    }

    @RequestMapping(value = "/{id}/author", method = RequestMethod.GET)
    public @ResponseBody Author getPostAuthor(@PathVariable("id") Long id) {

        return postService.findPostAuthor(id);
    }

    @RequestMapping(value = "/{id}/category", method =
RequestMethod.GET)
    public @ResponseBody List<Category>
findPostCategories(@PathVariable("id") Long id ) {
        return categoryService.findCategoriesByPostId(id);
    }
}

```

Ilustración 123 Clase PostController del proyecto restful-blog-posts

```

public interface PostRepository extends JpaRepository<Post, Long> {
    @Query
    List<Post> findByAuthor(Author author);

    @Query
    List<Post> findByDate(Date date);

    @Query("SELECT post.id, post.date, post.content FROM
        Post post JOIN post.categories category WHERE
        category.id=?1")

    List<Post> findPostsByCategoryId(Long id);

    @Query("SELECT post.author FROM Post post WHERE post.id=?1")
    Author findPostAuthor(Long id);
}

```

**Ilustración 124 Clase PostRepository del proyecto restful-blog-posts**

La tabla Autor deberá de ser removida en su totalidad y trasladada a la nuevas base de datos del microservicio, no tendrá ninguna alteración en su estructura. Sin embargo será necesaria una refactorización de la tabla post\_category debido a que se necesita crear una clase

## Creación del servicio publicaciones

Se ingresará a la siguiente url <http://start.spring.io/> y se creará un nuevo proyecto maven con la última versión de spring-boot y depedencias web, que contendrán el código fuente removido del monolito asociado con el manejo de los autores.

Project Metadata

Artifact coordinates

Group

com.benjsicam.restfulblog

Artifact

restful-blog-posts

Name

restful-blog-posts

Description

Demo project for Spring Boot

Package Name

com.benjsicam.restfulblog

Packaging

Jar

Java Version

1.8

Language

Java

Web

- Web**  
Full-stack web development with Tomcat and Spring MVC
- Websocket**  
WebSocket development with SockJS and STOMP
- Web Services**  
Contract-first SOAP service development with Spring Web Services
- Jersey (JAX-RS)**  
RESTful Web Services framework
- Ratpack**  
Spring Boot integration for the Ratpack framework
- Vaadin**  
Vaadin java web application framework
- Rest Repositories**  
Exposing Spring Data repositories over REST via spring-data-rest-webmvc
- HATEOAS**  
HATEOAS-based RESTful services
- Rest Repositories HAL Browser**  
Browsing Spring Data REST repositories in your browser
- Mobile**  
Simplify the development of mobile web applications with spring-mobile
- REST Docs**  
Document RESTful services by combining hand-written and auto-generated documentation
- Stormpath**  
Stormpath default starter including Spring MVC, Thymeleaf and Spring Security

Ilustración 125 Formulario de start.spring.io para la creación dek microservicio restful-blog-posts

Es necesario hacer click en el botón *Generate Project* y se descargará el archivo .zip generado en el cual se descomprimirá en el directorio d:/microservicios/ejemplo-metodologia-microservicios. Y se modificará el archivo pom.xml de la siguiente manera:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.benjsicam.restfulblog</groupId>
  <artifactId>restful-blog-posts</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>restful-blog-posts</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <!-- Spring boot dependency -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Security dependencies, http-basic -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- Mysql driver -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>

```

```

        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <!-- Jackson mapper dependency -->
    <dependency>
        <groupId>org.codehaus.jackson</groupId>
        <artifactId>jackson-mapper-asl</artifactId>
        <version>1.9.12</version>
    </dependency>
    <!-- Eureka Client -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>
    <!-- Feign client-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-feign</artifactId>
    </dependency>
    <!-- Cloud config -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-client</artifactId>
    </dependency>
    <!-- Hystrix client -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-hystrix</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- Spring retry dependencies -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-aop</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.retry</groupId>
        <artifactId>spring-retry</artifactId>
        <version>1.1.5.RELEASE</version>
    </dependency>
    <!-- Spring cloud bus dependency -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

```

```

        </exclusion>
    </exclusions>
</dependency>
<!-- Spring log4jv2 dependency --&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-log4j2&lt;/artifactId&gt;
&lt;/dependency&gt;
&lt;/dependencies&gt;
&lt;dependencyManagement&gt;
    &lt;dependencies&gt;
        &lt;dependency&gt;
            &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
            &lt;artifactId&gt;spring-cloud-dependencies&lt;/artifactId&gt;
            &lt;version&gt;Brixton.SR6&lt;/version&gt;
            &lt;type&gt;pom&lt;/type&gt;
            &lt;scope&gt;import&lt;/scope&gt;
        &lt;/dependency&gt;
    &lt;/dependencies&gt;
&lt;/dependencyManagement&gt;
&lt;build&gt;
    &lt;plugins&gt;
        &lt;plugin&gt;
            &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
            &lt;artifactId&gt;spring-boot-maven-plugin&lt;/artifactId&gt;
        &lt;/plugin&gt;
    &lt;/plugins&gt;
&lt;/build&gt;
</pre>

```

**Ilustración 126 Archivo pom.xml del proyecto restful-blog-posts**

La clase principal del microservicio posts es RestfulBlogPostsApplication la cual contiene todos los *endPoints* existentes en el monolito, desde esta clase hacen los llamados a la clase PostService la cual contiene la lógica del negocio.

```

@SpringBootApplication
@RestController
@ComponentScan
@RequestMapping("/resources/posts")
@EnableDiscoveryClient
@EnableFeignClients
@EnableCircuitBreaker
public class RestfulBlogPostsApplication {

    private static final Log logger =
LogFactory.getLog(RestfulBlogPostsApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogPostsApplication.class, args);
    }
    @Autowired
    private PostService postService;
}

```

```

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(value = HttpStatus.CREATED)
    public void create(@RequestBody Post post) {

        postService.create(post);
        logger.info("create called");
    }
    @RequestMapping(method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public void update(@RequestBody Post post) {

        postService.update(post);
        logger.info("update called");
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.OK)
    public void delete(@PathVariable("id") Long id) {

        postService.delete(id);
        logger.info("delete called");
    }

    @RequestMapping(method = RequestMethod.GET)
    public @ResponseBody List<Post> find() {

        return postService.find();
    }
    @RequestMapping(value = "/author/{authorId}", method =
RequestMethod.GET)
    public @ResponseBody List<Post>
findByAuthor(@PathVariable("authorId") Long authorId) {
        logger.info("findByAuthor called");
        return postService.findByAuthor(authorId);
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public @ResponseBody Post findPost(@PathVariable("id") Long id ) {
        logger.info("findById called");
        return postService.findById(id);
    }

    @RequestMapping(value = "/{id}/author", method = RequestMethod.GET)
    public @ResponseBody
Author getPostAuthor(@PathVariable("id") Long id) {
        Long postAuthorId = postService.findPostAuthor(id);
        logger.info("findUserId called");
        return postService.findById(postAuthorId);
    }

    @RequestMapping(value = "/{id}/category", method =
RequestMethod.GET)
    public @ResponseBody List<Category>

```

```

    findPostCategories(@PathVariable("id") Long id) {
        logger.info("findPostCategories called");
        return postService.findPostCategories(id);
    }
}

```

**Ilustración 127 Clase RestfulBlogPostsApplication del proyecto restful-blog-posts**

La clase PostService contiene los llamados al repositorio PostRepository de donde accede a las tablas de la base de datos restul\_blog\_posts, también hace uso del microservicio author y del monolito para acceder a las credenciales. En el monolito los métodos create y update estaban acoplados con tabla category por medio base de datos y necesitan desacoplarse, se esperará hasta que el siguiente microservicio sea creado para realizar el desacoplamiento.

```

@Service
public class PostService {

    private static final Log logger =
        LogFactory.getLog(PostService.class);
    final static String queueName = "restful-blog-category";
    final static String exchangeName =
        "restful-blog-category-exchange";

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Autowired
    private PostRepository postRepository;

    @Autowired
    private AuthorClientService authorClientService;

    @Autowired
    private CategoryClientService categoryClientService;

    @Transactional
    public void create(Post post) throws IOException {
        postRepository.saveAndFlush(post);
        for (Category category : post.getCategories()) {
            PostCategory postCategory = new PostCategory();
            postCategory.setPostId(post.getId());
            category.getPostCategories().add(postCategory);
        }
        this.sendRabbitMQMessage("create", post.getCategories());
        logger.info("you called create");
    }
}

```

```

@Transactional
public void update(Post post) throws IOException {
    postRepository.saveAndFlush(post);
    this.sendRabbitMQMessage("update", post.getCategories());
    logger.info("you called update");
}

private void sendRabbitMQMessage(String operation, Object body)
throws IOException {
    ObjectMapper mapper = new ObjectMapper();
    RabbitMQMessage message = new RabbitMQMessage("post",
        operation, mapper.writeValueAsString(body));
    rabbitTemplate.setExchange(exchangeName);
    rabbitTemplate.convertAndSend(queueName, message);
    rabbitTemplate.setMessageConverter(getJsonMessageConverter());
}

public MessageConverter getJsonMessageConverter() {
    return new Jackson2JsonMessageConverter();
}

@Transactional
public void delete(Long id) {
    postRepository.delete(id);
    logger.info("you called delete");
}

@HystrixCommand(fallbackMethod = "findDefault")
public List<Post> find() {
    logger.info("you called findAll");
    return postRepository.findAll();
}

public List<Post> findDefault(){
    logger.info(" called findDefault");
    return new ArrayList<Post>();
}

@HystrixCommand(fallbackMethod = "findByIdDefault")
public Post findById(Long id ) {
    logger.info("you called findAll");
    return postRepository.findOne(id);
}

public List<Post> findByIdDefault(Long id){
    logger.info(" called findDefault");
    return new ArrayList<Post>();
}

@HystrixCommand(fallbackMethod = "findPostAuthorDefault")
public Long findPostAuthor(Long id) {
    logger.info("you called findPostAuthor");
    return postRepository.findPostAuthor(id);
}

public List<Post> findPostAuthorDefault(Long id){
    logger.info(" called findPostAuthorDefault");
    return new ArrayList<Post>();
}

@HystrixCommand(fallbackMethod = "findByAuthorDefault")
public List<Post> findByAuthor(@PathVariable("authorId")

```

```

        Long authorId) {
    logger.info("you called findByAuthor");
    return postRepository.findByAuthor(authorId);
}
public List<Post> findByAuthorDefault(Long id) {
    logger.info(" called findByAuthorDefault");
    return new ArrayList<Post>();
}
@HystrixCommand(fallbackMethod = "findByUserIdDefault")
public Author findByUserId(Long postAuthorId) {
    logger.info("you called findByUserId");
    return authorClientService.findByUserId(postAuthorId);
}
public List<Post> findByUserIdDefault(Long id) {
    logger.info(" called findByUserIdDefault");
    return new ArrayList<Post>();
}
@HystrixCommand(fallbackMethod = "findPostCategoriesDefault")
public List<Category> findPostCategories(Long id) {
    logger.info("you called findPostCategories");
    return categoryClientService.findCategoriesByPostId(id);
}
public List<Post> findPostCategoriesDefault(Long id) {
    logger.info(" called findByUserIdDefault");
}

```

Ilustración 128 Clase PostService del proyecto restful-blog-posts

```

public interface PostRepository extends JpaRepository<Post, Long> {
    @Query
    List<Post> findByAuthor(Long authorId);

    @Query
    List<Post> findByDate(Date date);

    @Query("SELECT post.author FROM Post post WHERE post.id=?1")
    Long findPostAuthor(Long id);
}

```

Ilustración 129 Clase PostRepository del proyecto restful-blog-posts

El microservicio necesitará de tres cliente que le permitan interactúa con los microservicios cedentials, authors y con el monolito, a continuación se muestra el código fuente:

```
@FeignClient(name="monolith", configuration =
FeingClientConfiguration.class)
public interface CategoryClientService {
    @RequestMapping(method = RequestMethod.GET,
value="/resources/category/post/{postId}")
    List<Category> findCategoriesByPostId
    (@PathVariable("postId") Long postId);

    @RequestMapping(method = RequestMethod.GET,
value="/resources/category/{id}/posts")
    List<Post> findPostsByCategoryId(Long id);
}
```

Ilustración 130 Clase CategoryClientService del proyecto restful-blog-posts

```
@FeignClient(name="author", configuration =
FeingClientConfiguration.class)
public interface AuthorClientService {

    @RequestMapping(method = RequestMethod.GET,
value="/resources/author/{authorId}")
    Author findByUserId(@PathVariable("authorId") Long authorId); }
```

Ilustración 131 Clase AuthorClientService del proyecto restful-blog-posts

```
@FeignClient(name="credentials", configuration =
FeingClientConfiguration.class)
public interface CredentialsClientService {

    @RequestMapping(method = RequestMethod.POST,
        value="/resources/credentials/", produces =
        MediaType.APPLICATION_JSON_VALUE, consumes =
        MediaType.APPLICATION_JSON_VALUE)
    void saveCredentials(@RequestBody Credentials credentials);

    @RequestMapping(method = RequestMethod.GET,
        value="/resources/credentials/{userName}")
    Credentials loadCredentialsByUserName(@PathVariable("userName")
String userName);

    @RequestMapping(method = RequestMethod.DELETE,
value="/resources/credentials/{id}")
    void deleteCredentials(@PathVariable("id") Long userId);
}
```

Ilustración 132 Clase CredentialsClientService del proyecto restful-blog-posts

Se realizará la configuración de los archivos propiedades para en archivo bootstrap.yml y posteriormente con application.yml

```

spring:
  application:
    name: post
  cloud:
    config:
      uri: http://localhost:8888

```

**Ilustración 133 Archivo bootstrap.yml del proyecto restful-blog-posts**

```

server:
  port: 8383

spring:
  rabbitmq:
    port: 5672
    host: 'localhost'
    username: guest
    password: guest
  datasource:
    url: jdbc:mysql://localhost/restful_blog_post
    username: root
    password: root
    driverClassName: com.mysql.jdbc.Driver
  jpa:
    show-sql : true
    hibernate:
      ddl-auto: update
      naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5Dialect
  eureka:
    client:
      serviceUrl:
        defaultZone: http://localhost:8761/eureka/

```

**Ilustración 134 Archivo application.yml del proyecto restful-blog-posts**

Las clases UserDetailsServiceImpl y SecurityConfiguration y el archivo de configuración log4j2.xml se copiarán dentro del proyecto restful-blog-posts desde el proyecto restful-blog-author con el mismo contenido y paquete sin ninguna modificación adicional. Adicionalmente será necesario agregar un archivo en el repositorio local, igual que se hizo con credentials y author llamado post.properties el cual contendrá los mismo valores que

los demás archivo y se tendrá que realizar un nuevo commit. En el proyecto restful-blog-monitor, se modificarán la siguientes dos propiedades en el archivo application.yml

```
turbine:  
  appConfig: author,credentials,post  
  aggregator:  
    clusterConfig: AUTHOR,CREDENTIALS,POST
```

Ilustración 135 Archivo de configuración restful-blog-monitor incluyendo post

También será necesario crear la nueva base de datos para el servicio en el servidor mysql de la máquina local, se ejecutará el siguiente comando desde la consola mysql se creará la base de datos *restful\_blog\_post*, y se seleccionará la base datos y se ejecutará el siguiente script

```
CREATE TABLE IF NOT EXISTS `post` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `content` text,  
  `date` date DEFAULT NULL,  
  `author` bigint(20) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `post_author` (`author`)  
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;
```

Ilustración 136 Script sql de tabla post para base de datos restful\_blog\_post

Y se insertarán los siguientes datos en la tabla anteriormente creada

```
INSERT INTO `post` (`id`, `content`, `date`, `author`) VALUES
(1, 'Lorem ipsum dolor sit amet', '2013-05-07', 1),
(2, 'Sed id nunc in nisi ', '2013-05-08', 1),
(3, 'Ut neque purus, convallis ', '2013-05-09', 1),
(4, 'Nulla id turpis ipsum', '2013-05-10', 2),
(5, 'Curabitur vel massa nec ', '2013-05-11', 2),
(6, 'Praesent ac dui risus. ', '2013-05-12', 2);
```

Ilustración 137 Valores para la tabla post para base de datos restful\_blog\_post

```
@Entity
@Table(name = "category")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Category {

    ...

    @JsonIgnore
    @OneToMany(mappedBy = "category", fetch = FetchType.EAGER)
    private Set<PostCategory> postCategories;

    public Set<PostCategory> getPostCategories() {
        return postCategories;
    }
}
```

Ilustración 138 Clase Category del proyecto restful-blog-monolith, remoción de etiqueta  
ManyToMany y adición de lista de objetos tipo PostCategory

En monolito también requiere cambios que permita mantener la comunicación entre ambos lados para ello se necesitará modificar las clase Category e incluir la clase PostCategory de tal manera que permita utilizar la tabla post\_category removiendo la anotación @ManyToMany que existia en la clase Category. También será necesario escribir un nuevo cliente que permita realizar la búsqueda de un post en el microservicio post y posteriormente hacer la codificación en la clase CategoryService y CategoryController para que utilza el nuevo cliente.

```

@Entity
@Table(name = "post_category")
@JsonIgnoreProperties(ignoreUnknown = true)
public class PostCategory {
    @Id
    @Column
    private Long id;

    @Column(name="post_id")
    private Long postId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name="category_id")
    private Category category;

    public Long getPostId() {
        return postId;
    }
    public void setPostId(Long postId) {
        postId = postId;
    }
    public Category getCategory() {
        return category;
    }
    public void setCategory(Category category) {
        category = category;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
}

```

Ilustración 139 Clase PostCategory del proyecto restful-blog

```

@Service
public class PostClient {

    @Autowired
    private ApplicationContext applicationContext;

    @Autowired
    SpringCloudConfigClient springCloudConfigClient;

    public Post findPostById(Long id) {

        RestTemplate restTemplate = new RestTemplate();
        HttpEntity<String> request =
                new HttpEntity<String>(buildHeaders());
        return restTemplate.exchange(getPostBase().concat("/");
                concat(String.valueOf(id)), HttpMethod.GET, request,
                Post.class).getBody();
    }

    private HttpHeaders buildHeaders() {
        byte[] plainCredsBytes =
                springCloudConfigClient.getServiceAuthUser().
                getCredentials().getBytes();
        byte[] base64CredsBytes =
                Base64.encodeBase64(plainCredsBytes);
        String base64Creds = new String(base64CredsBytes);
        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", "Basic " + base64Creds);
        return headers;
    }

    private String getPostBase() {
        String fooResourceUrl;
        if ("docker".equals(System.getenv("SPRING_PROFILES_ACTIVE"))){
            String host =
                    System.getenv("POST_PORT_8383_TCP_ADDR").
                    concat(":8282");
            fooResourceUrl= "http://".concat(host).
                    concat("/resources/author/");
        }else{
            fooResourceUrl= "http://localhost:8383/resources/posts/";
        }
        return fooResourceUrl;
    }
}

```

Ilustración 140 PostClient del proyecto restful-blog

```

@Service
public class CategoryService {
    @Autowired
    PostClient postClient;
    ...
    public Post findPost(Long id) {
        return postClient.findPostById(id);
    }
}

```

**Ilustración 141 CategoryService del proyecto restful-blog**

```

@Controller
@RequestMapping("/resources/category")
public class CategoryController {
    ...
    @RequestMapping(value = "/{id}/posts", method =
RequestMethod.GET)
    public @ResponseBody List<Post> findCategoryPosts(@PathVariable
("id") Long id) {
        List<Post> posts = new ArrayList<Post>();
        Category category = categoryService.findById(id);
        for (PostCategory pc: category.getPostCategories()) {
            posts.add(categoryService.findPost(pc.getPostId()));
        }
        logger.info("findCategoryPosts called");
        return posts;
    }

    @RequestMapping(value = "/post/{id}", method = RequestMethod.GET)
    public @ResponseBody List<Category> findCategoriesByPostId
        (@PathVariable("id") Long id) {
        logger.info("findCategoriesByPostId called");
        return categoryService.findCategoriesByPostId(id);
    }
}

```

**Ilustración 142 CategoryController del proyecto restful-blog**

### Implementación fase III

En esta fase el enfoque será extraer el código restante del monolito, perteneciente al contexto de folcsonomía, al microservicio encargado de las categorías. También se implementará una comunicación orientada a mensajes entre los microservicios haciendo uso del *broker* reemplazando las llamados entre microservicios que implican inserción, actualización o borrado de valores en las tablas existentes en la base de datos.

#### Inspección del código fuente de folcsonomía en el monolito

Paquete	Clase	Notas
com.benjsicam.restfulblog.service	CategoryService	Contiene los métodos de consulta de la información de las categorías, toda la clase deberá de ser trasladada al nuevo microservicio
com.benjsicam.restfulblog.controller	CategoryController	Contiene todos los endpoint de los servicios REST de las categorías, la clase también será trasladada en su totalidad al microservicio
com.benjsicam.restfulblog.dao	CategoryRepository	Interfaz utilizada para las llamadas JPA hacia la base de datos. Deberá ser trasladada al nuevo microservicio
com.benjsicam.restfulblog.domain	Category	Clases de dominio, la clase Post se deberá de transcribir con todas las anotaciones JPA en tanto que en las otras será necesario remover las anotaciones JPA
com.benjsicam.restfulblog.domain	PostCategory	

Tabla 4 Resultados de la inspección del código fuente del contexto folcsonomía en el monolito

## Desprendimiento del código fuente del contexto de publicaciones en el monolito

A continuación se listan los fragmentos de código fuente asociado al contexto folcsonomia

```
@Service
public class CategoryService {
    @Autowired
    CategoryRepository categoryRepository;

    @Transactional
    public void create(Category category) {
        categoryRepository.saveAndFlush(category);
    }

    @Transactional
    public void update(Category category) {
        categoryRepository.saveAndFlush(category);
    }

    @Transactional
    public void delete(Long id) {
        categoryRepository.delete(id);
    }

    public List<Category> find() {
        return categoryRepository.findAll();
    }

    public Category findById(Long id) {
        return categoryRepository.findOne(id);
    }

    public List<Category> findCategoriesByPostId(Long id) {
        return categoryRepository.findCategoriesByPostId(id);
    }
}
```

Ilustración 143 Clase CategoryService existente en el monolito

```

    @Controller
    @RequestMapping("/resources/category")
    public class CategoryController {

        @Autowired
        private CategoryService categoryService;

        @Autowired
        private PostService postService;

        @RequestMapping(method = RequestMethod.POST)
        @ResponseStatus(value = HttpStatus.CREATED)
        public void create(@RequestBody Category entity) {
            categoryService.create(entity);
        }

        @RequestMapping(method = RequestMethod.PUT)
        @ResponseStatus(HttpStatus.OK)
        public void update(@RequestBody Category entity) {
            categoryService.update(entity);
        }

        @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
        @ResponseStatus(HttpStatus.OK)
        public void delete(@PathVariable("id") Long id) {
            categoryService.delete(id);
        }

        @RequestMapping(method = RequestMethod.GET)
        public @ResponseBody List<Category> find() {
            return categoryService.find();
        }

        @RequestMapping(value = "/{id}", method = RequestMethod.GET)
        public @ResponseBody Category findCategory(@PathVariable("id") Long id) {
            return categoryService.findById(id);
        }

        @RequestMapping(value = "/{id}/posts", method = RequestMethod.GET)
        public @ResponseBody List<Post>
findCategoryPosts(@PathVariable("id") Long id ) {
            return postService.findPostsByCategoryId(id);
        }
    }
}

```

**Ilustración 144 Clase CategoryController del proyecto restful-blog**

```

@Entity
@Table(name = "category")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;
    @Column(name = "name", length = 50, nullable = false,
            unique = true)
    private String name;

    @JsonIgnore
    @ManyToMany(mappedBy = "categories")
    private Set<Post> posts;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Set<Post> getPosts() {
        return posts;
    }
    public void setPosts(Set<Post> posts) {
        this.posts = posts;
    }
}

```

Ilustración 145 Clase Category existente en el proyecto restful-blog

```

public interface CategoryRepository extends
JpaRepository<Category, Long> {

    @Query
    Category findByName(String name);

    @Query("SELECT category.id, category.name FROM Category
           category JOIN category.posts post WHERE post.id=?1")
    List<Category> findCategoriesByPostId(Long id);
}

```

Ilustración 146 Clase CategoryRepository del proyecto restful-blog

Las tablas category y post\_category necesitarán ser trasladadas a la nueva a la nueva base de datos juntas debido a que se encuentran acopladas entre ella quedan así la base de datos del monolito completamente vacía.

## Creación del servicio categories

Se ingresa a la siguiente url <http://start.spring.io/> y se creará un nuevo proyecto maven con la última versión de spring-boot y depedencias web, que contendrán el código fuente removido del monolito asociado con el manejo de los autores.

Project Metadata		Web
Artifact coordinates		<input checked="" type="checkbox"/> <b>Web</b> Full-stack web development with Tomcat and Spring MVC
Group		<input type="checkbox"/> <b>WebSocket</b> WebSocket development with SockJS and STOMP
Artifact		<input type="checkbox"/> <b>Web Services</b> Contract-first SOAP service development with Spring Web Services
Name		<input type="checkbox"/> <b>Jersey (JAX-RS)</b> RESTful Web Services framework
Description		<input type="checkbox"/> <b>Ratpack</b> Spring Boot integration for the Ratpack framework
Package Name		<input type="checkbox"/> <b>Vaadin</b> Vaadin java web application framework
Packaging		<input type="checkbox"/> <b>Rest Repositories</b> Exposing Spring Data repositories over REST via spring-data-rest-webmvc
Java Version		<input type="checkbox"/> <b>HATEOAS</b> HATEOAS-based RESTful services
Language		<input type="checkbox"/> <b>Rest Repositories HAL Browser</b> Browsing Spring Data REST repositories in your browser
		<input type="checkbox"/> <b>Mobile</b> Simplify the development of mobile web applications with spring-mobile
		<input type="checkbox"/> <b>REST Docs</b> Document RESTful services by combining hand-written and auto-generated documentation

Too many options? [Switch back to the simple version.](#)

**Ilustración 147 Formulario de start.spring.io para la creación del proyecto restful-blog-categories**

Se hace click en el botón Generate Project y se descargará el archivo .zip generado en el cual descomprimirá en el directorio d:/microservicios/ejemplo-metodologia-microservicios. Y se modificará el archivo pom.xml de la siguiente manera:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.benjsicam.restfulblog</groupId>
  <artifactId>restful-blog-categories</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>restful-blog-categories</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <!-- Spring boot dependency -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Security dependencies, http-basic -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- Mysql driver -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <scope>runtime</scope>

```

```

</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<!-- Jackson mapper dependency -->
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.12</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-hibernate4</artifactId>
    <version>2.4.4</version>
</dependency>

<!-- Eureka Client -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
<!-- Feign client-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
<!-- Cloud config -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-client</artifactId>
</dependency>
<!-- Hystrix client -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<!-- Spring retry dependencies -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.retry</groupId>
    <artifactId>spring-retry</artifactId>
    <version>1.1.5.RELEASE</version>
</dependency>
<!-- Spring cloud bus dependency -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>

```

```

<exclusions>
    <exclusion>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
</exclusions>
</dependency>
<!-- Spring log4jv2 dependency -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Brixton.SR6</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

Ilustración 148 Archivo pom.xml del proyecto restful-blog-categories

La clase principal del microservicio categories es RestfulBlogCategoriesApplication la cual contiene todos los del microservicio, desde esta clase se hacen los llamados a la clase CategoryService la cual contiene la lógica del negocio.

```
@SpringBootApplication
@RestController
@ComponentScan
@RequestMapping("/resources/categories")
@EnableDiscoveryClient
@EnableFeignClients
@EnableCircuitBreaker
public class RestfulBlogCategoriesApplication {

    private static final Log logger =
LogFactory.getLog(RestfulBlogCategoriesApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(RestfulBlogCategoriesApplication.class,
args);
    }

    @Autowired
    private CategoryService categoryService;

    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(value = HttpStatus.CREATED)
    public void create(@RequestBody Category entity) {
        logger.info("create called");
        categoryService.create(entity);
    }
    @RequestMapping(method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.OK)
    public void update(@RequestBody Category entity) {
        logger.info("update called");
        categoryService.update(entity);
    }
    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.OK)
    public void delete(@PathVariable("id") Long id) {
        logger.info("delete called");
        categoryService.delete(id);
    }
    @RequestMapping(method = RequestMethod.GET)
    public @ResponseBody List<Category> find() {
        logger.info("find called");
        return categoryService.find();
    }
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public @ResponseBody Category findCategory(@PathVariable("id") Long
```

```

    id ) {
        logger.info("findCategory called");
        return categoryService.findById(id);
    }
    @RequestMapping(value = "/{id}/posts", method = RequestMethod.GET)
    public @ResponseBody List<Post>
findCategoryPosts(@PathVariable("id") Long id ) {
    List<Post> posts = new ArrayList<Post>();
    Category category =
categoryService.findCategoriesByPostId(id).get(0);
    category.getPostCategories().size();
    for (PostCategory pc: category.getPostCategories()) {
        posts.add(categoryService.findPost(pc.getPostId()));
    }
    logger.info("findCategoryPosts called");
    return posts;
}
    @RequestMapping(value = "/post/{id}", method = RequestMethod.GET)
    public @ResponseBody List<Category>
findCategoriesByPostId(@PathVariable("id") Long id ) {
    logger.info("findCategoriesByPostId called");
    return categoryService.findCategoriesByPostId(id);
}
}

```

**Ilustración 149 Clase RestfulBlogCategoriesApplication del proyecto restful-blog-categories**

La clase CategoryService contiene los llamados al repositorio CategoryRepository de donde accede a las tablas category y post\_category de la base de datos restul\_blog\_categories . De igual manera el servicio también hace uso del microservicio Post para consultar la publicación asociada a una categoría.

```
@Service
public class CategoryService {

    private static final Log logger =
LogFactory.getLog(CategoryService.class);

    @Autowired
    PostClientService postClient;

    @Autowired
    CategoryRepository categoryRepository;

    @Transactional
    public void create(Category category) {
        categoryRepository.saveAndFlush(category);
        logger.info("you called create");
    }

    @Transactional
    public void update(Category category) {
        categoryRepository.saveAndFlush(category);
        logger.info("you called update");
    }

    @Transactional
    public void delete(Long id) {
        categoryRepository.delete(id);
        logger.info("you called delete");
    }

    @HystrixCommand(fallbackMethod = "findDefault")
    public List<Category> find() {
        logger.info("you called findDefault");
        return categoryRepository.findAll();
    }

    private List<Category> findDefault(){
        logger.info("called findDefault");
        return new ArrayList<Category>();
    }
}
```

Ilustración 150 Clase CategoryService del proyecto restful-blog-category

```
public interface CategoryRepository extends  
JpaRepository<Category, Long> {  
  
    @Query  
    Category findByName(String name);  
  
    @Query("SELECT category FROM Category category JOIN FETCH  
          category.postCategories pc WHERE pc.postId=?1")  
    List<Category> findCategoriesByPostId(Long id);  
}
```

Ilustración 151 Clase CategoryRepository del proyecto restful-blog-category

Las clases del modelo utilizadas Category y PostCategory no necesitan de ser modificadas, se debe de transcribirla de la misma manera como quedaron en el monolito y copiarlas en los mismos paquetes:

```
@Entity
@Table(name = "category")
@JsonIgnoreProperties(ignoreUnknown = true)
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "name", length = 50, nullable = false, unique =
true)
    private String name;

    @OneToMany(mappedBy = "category", fetch = FetchType.LAZY, cascade =
CascadeType.ALL)
    private Set<PostCategory> postCategories;
    /**
     * @return the id
     */
    public Long getId() {
        return id;
    }
    /**
     * @param id the id to set
     */
    public void setId(Long id) {
        this.id = id;
    }
    /**
     * @return the name
     */
    public String getName() {
        return name;
    }
    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }
    public Set<PostCategory> getPostCategories() {
        return postCategories;
    }
    public void setPostCategories(Set<PostCategory> postCategories) {
        this.postCategories = postCategories;
    }
}
```

Ilustración 152 Clase PostCategory transcrita al proyecto restful-blog-categories

```

@Entity
@Table(name = "post_category")
@JsonIgnoreProperties(ignoreUnknown = true)
public class PostCategory {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    @Column(name="post_id")
    private Long postId;

    @ManyToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="category_id", referencedColumnName = "id")
    private Category category;

    public Long getPostId() {
        return postId;
    }

    public void setPostId(Long postId) {
        this.postId = postId;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}

```

Ilustración 153 Clase Category transcrita al proyecto restful-blog-categories

El microservicio necesitará de dos clientes que le permitan interactuar con los microservicios credentials, y con el post, a continuación se muestra el código fuente de los clientes:

```
@FeignClient(name="credentials", configuration =
FeingClientConfiguration.class)
public interface CredentialsClientService {

    @RequestMapping(method = RequestMethod.POST,
value="/resources/credentials/", produces =
MediaType.APPLICATION_JSON_VALUE, consumes =
MediaType.APPLICATION_JSON_VALUE)
    void saveCredentials(@RequestBody Credentials credentials);

    @RequestMapping(method = RequestMethod.GET,
value="/resources/credentials/{userName}")
    Credentials loadCredentialsByUserName(@PathVariable("userName")
String userName);

    @RequestMapping(method = RequestMethod.DELETE,
value="/resources/credentials/{id}")
    void deleteCredentials(@PathVariable("id") Long userId);
}
```

Ilustración 154 Cliente del microservicio credentials en el proyecto restful-blog-categories

```
@FeignClient(name="post", configuration =
FeingClientConfiguration.class)
public interface PostClientService {

    @RequestMapping(method = RequestMethod.GET,
value="/resources/posts/{postId}")
    Post findPostById(@PathVariable("postId") Long authorId);
}
```

Ilustración 155 Cliente del microservicio posts en el proyecto restful-blog-categories

Se realizará la configuración de los archivos propiedades para la correcta ejecución del microservicio, se comenzará con el archivo bootstrap.yml y posteriormente con application.yml

```
spring:
  application:
    name: category
  cloud:
    config:
      uri: http://localhost:8888
```

Ilustración 156 Archivo bootstrap.yml del proyecto restful-blog-categories

```
server:
  port: 8484

spring:
  rabbitmq:
    port: 5672
    host: 'localhost'
    username: guest
    password: guest
  datasource:
    url: jdbc:mysql://localhost/restful_blog_category
    username: root
    password: root
    driverClassName: com.mysql.jdbc.Driver
  jpa:
    show-sql : true
    hibernate:
      ddl-auto: update
      naming-strategy: org.hibernate.cfg.ImprovedNamingStrategy
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5Dialect
  eureka:
    client:
      serviceUrl:
        defaultZone: http://localhost:8761/eureka/
```

Ilustración 157 Archivo application.yml del proyecto restful-blog-categories

Las clases UserDetailsServiceImpl y SecurityConfiguration y el archivo de configuración log4j2.xml se copiarán dentro del proyecto restful-blog-categories desde el proyecto restful-blog-author con el mismo contenido y paquete sin ninguna modificación adicional. Adicionalmente será necesario agregar un archivo en el repositorio local, igual que se hizo con credentials y author llamado categories.properties el cual contendrá los mismo valores que los demás archivo y se tendrá que realizar un nuevo commit.

En el proyecto restful-blog-monitor, se modificará el archivo application.yml con los siguientes valores

```
turbine:  
  appConfig: author,credentials,post,category  
  aggregator:  
    clusterConfig: AUTHOR,CREDENTIALS,POST,CATEGORY
```

Ilustración 158 Archivo application.yml del proyecto restful-blog-monitor

También se necesitará crear la nueva base de datos para el servicio en el servidor mysql de la máquina local, se creará la base de datos *restful\_blog\_category*, se seleccionará la base datos y se ejecutará el siguiente script:

```
CREATE TABLE IF NOT EXISTS `category` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `name` varchar(50) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO_INCREMENT=46 DEFAULT CHARSET=utf8;  
  
CREATE TABLE IF NOT EXISTS `post_category` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `post_id` bigint(20) NOT NULL,  
  `category_id` bigint(20) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `FK4BC509BD1CD41CA0` (`category_id`)  
)
```

Ilustración 159 Scripts tablas category y post\_category del microservicio categories

Y posteriormente se insertará los valores en la base de datos:

```
INSERT INTO `category` (`id`, `name`) VALUES
(1, 'General'),
(2, 'News'),
(3, 'Announcements');

INSERT INTO `post_category` (`id`, `post_id`, `category_id`) VALUES
(1, 1, 1),
(2, 4, 1),
(3, 2, 2),
(4, 5, 2),
(5, 3, 3),
(6, 6, 3);
```

Ilustración 160 Scripts valores tablas category y post\_category del microservicio categories

## Integración de los servicios por medio de mensajería

Existe un fuerte acoplamiento entre los microservicios Author y Credentials donde la creación, actualización o eliminación de un autor tiene una incidencia directa en las credenciales existentes en el sistema. También existen un acoplamiento entre los microservicios Post y Categories debido a que la creación o actualización de un post implica la creación de nuevas categorías. La clase que se ha definido para la comunicación entre los microservicios se llama RabbitMQMessage la cual será serializada y de serializada en formato json.

```
public class RabbitMQMessage {

    private String microserviceName;
    private String operation;
    private String body;
    public RabbitMQMessage() {

    }
    public RabbitMQMessage(String microserviceName, String operation,
String body) {
        this.microserviceName = microserviceName;
        this.operation = operation;
        this.body = body;
    }
    public String getMicroserviceName() {
        return microserviceName;
    }
    public void setMicroserviceName(String microserviceName) {
        this.microserviceName = microserviceName;
    }
    public String getOperation() {
        return operation;
    }
    public void setOperation(String operation) {
        this.operation = operation;
    }
    public String getBody() {
        return body;
    }
    public void setBody(String body) {
        this.body = body;
    }
}
```

Ilustración 161 Clase RabbitMQMessage como modelo para utilizar la comunicación de mensajes entre microservicios

Se comenzará por desacoplar los llamados create, update y delete del microservicio authors con el microservicio credentials, para ello en el proyecto restful-blog-posts en la clase AuthorService se creará el método sendRabbitMQMessage, que funcionara como productor, el cual se encarga de serializar la clase RabbitMQMessage y enviarlo al servidor rabbitMq haciendo uso de la conexión definida en el archivo application.yml.

```
@Service
public class AuthorService {
    private final static String queueName = "restful-blog-credentials";
    private final static String exchangeName =
        "restful-blog-credentials-exchange";
    private static final Log logger =
        LogFactory.getLog(AuthorService.class);
    @Autowired
    private RabbitTemplate rabbitTemplate;
    ...

    @Transactional
    public void create(Author author) throws IOException {
        author.setPassword(passwordEncoder.encode(author.getPassword()));
        authorRepository.saveAndFlush(author);
        Credentials authorCredentails =
            this.createAuthorCredentails(author);
        this.sendRabbitMQMessage("create", authorCredentails);
        logger.info("you called create");
    }

    @Transactional
    public void update(Author author) throws IOException {
        if (this.findById(author.getId()) != null) {
            author.setPassword(passwordEncoder.encode(author.getPassword()));
            authorRepository.saveAndFlush(author);
            Credentials authorCredentails =
                this.createAuthorCredentails(author);
            this.sendRabbitMQMessage("update", authorCredentails);
        }
        logger.info("you called update");
    }
}
```

```

    @Transactional
public void delete(Long id) throws IOException {
    authorRepository.delete(id);
    this.sendRabbitMQMessage("delete", id);
    logger.info("you called delete");
}

    @Transactional
public void updatePassword(Author author) throws IOException {
    logger.info("you called updatePassword");

    author.setPassword(passwordEncoder.encode(author.getPassword()));
    Credentials authorCredentails =
        this.createAuthorCredentails(author);
    this.sendRabbitMQMessage("update", authorCredentails);
}

...
private void sendRabbitMQMessage(String operation, Object body)
    throws IOException {
ObjectMapper mapper = new ObjectMapper();
RabbitMQMessage message = new
    RabbitMQMessage("author", operation,
        mapper.writeValueAsString(body));

rabbitTemplate.setExchange(exchangeName);
rabbitTemplate.setMessageConverter(getJsonMessageConverter());
this.rabbitTemplate.convertAndSend(message);
}

public MessageConverter getJsonMessageConverter() {
    return new Jackson2JsonMessageConverter();
}

```

**Ilustración 162 Fragmento de la clase AuthorService que incluye llamados por mensajería al servidor RabbitMQ**

En el proyecto restful-blog-credentials se creará el consumidor encargado de escuchar los mensajes provenientes de la cola restful-blog-credentials la cual manejará los mensajes provenientes del microservicio *authors*. Se creará una clase cliente llamada RabbitMQClient que permita al microservicio conectarse al servidor RabbitMq, además de otra clase consumidora encargada de procesar los mensaje y ejecutarlos en el microservicio *credentials*.

```

@Component
@Configuration
public class RabbitMQClient {

    final static String queueName = "restful-blog-credentials";
    final static String exchangeName =
            "restful-blog-credentials-exchange";

    @Bean
    Queue queue() {
        return new Queue(queueName, false);
    }

    @Bean
    FanoutExchange exchange() {
        return new FanoutExchange(exchangeName);
    }

    @Bean
    Binding binding(Queue queue, FanoutExchange exchange) {

        return BindingBuilder.bind(queue).to(exchange);
    }

    @Bean
    SimpleMessageListenerContainer container(
            ConnectionFactory connectionFactory,
            MessageListenerAdapter listenerAdapter)
    {

        SimpleMessageListenerContainer container =
                new SimpleMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.setQueueNames(queueName);
        container.setMessageListener(listenerAdapter);
        return container;
    }

    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }
}

```

**Ilustración 163 Clase RabbitMQClient en el proyecto restful-blog-credentials**

La clase receiver del microservicio credentials se encarga de hacer llamado a la clase CredentialsService y hacer las modificaciones necesarias en la base de datos, no devuelve ningún valor al broker.

```

@Component
public class Receiver {

    @Autowired
    CredentialsService credentialsService;

    public void receiveMessage(byte[] bytes) throws IOException {

```

```

ObjectMapper mapper = new ObjectMapper();
RabbitMQMessage message = mapper.readValue(new String(bytes),
RabbitMQMessage.class);
switch (message.getOperation()) {

    case "create":
        this.credentialsService.saveAndFlush(mapper.
            readValue(message.getBody(), Credentials.class));
        break;

    case "update":
        this.credentialsService.saveAndFlush(mapper.
            readValue(message.getBody(), Credentials.class));
        break;
    case "delete":
        this.credentialsService.delete
            (mapper.readValue(message.getBody(), Long.class));
        break;
}
}
}

```

Ilustración 164 Clase Receiver del proyecto restful-blog-credentials

Luego de hacer estas modificaciones el microservicio *Authors* queda desacoplado de los llamados que hacen modificaciones en base de datos del microservicio *Credentials*. El siguiente punto de desacoplamiento son la llamadas de escritura de base de datos entre el microservicio Post y el microservicio Category. En el proyecto restful-blog-posts se modificará la clase PostService y se incluirá el método sendRabbitMQMessage que servirá de productor para hacer envío de los mensajes al microservicio Category haciendo uso de la conexión de RabbitMQ configurada previamente en el archivo application.yml

```

@Service
public class PostService {

    private static final Log logger =
        LogFactory.getLog(PostService.class);
    final static String queueName = "restful-blog-category";
    final static String exchangeName =
        "restful-blog-category-exchange";
    @Autowired
    private RabbitTemplate rabbitTemplate;
}

```

Ilustración 165 Fragmento de la clase PostService del proyecto restful-blog-post

En el proyecto restful-blog-categories se creará la clase consumidora de los mensajes de RabbitMq. RabbitMQClient será responsable de hacer la conexión con el servidor RabbitMQ, y la clase Receiver se encargará de procesar los mensajes y de hacer uso de la clase CategoryService para la interacción con la base de datos.

```
@Component
@Configuration
public class RabbitMQClient {

    final static String queueName = "restful-blog-category";
    final static String exchangeName = "restful-blog-category-
exchange";
    @Bean
    Queue queue() {
        return new Queue(queueName, false);
    }
    @Bean
    FanoutExchange exchange() {
        return new FanoutExchange(exchangeName);
    }
    @Bean
    Binding binding(Queue queue, FanoutExchange exchange) {
        return BindingBuilder.bind(queue).to(exchange);
    }
    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }
    @Bean
    SimpleMessageListenerContainer container(ConnectionFactory
connectionFactory,
                                              MessageListenerAdapter
listenerAdapter) {
        SimpleMessageListenerContainer container = new
SimpleMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.setQueueNames(queueName);
        container.setMessageListener(listenerAdapter);
        return container;
    }
}
```

Ilustración 166 Clase RabbitMQClient del proyecto restful-blog-categories

Con este cambios los métodos crear y actualizar del microservicio category quedaron desacoplados con el microservicio authors haciendo uso del broker para su comunicación asincrónica.

```
@Component
public class Receiver {

    @Autowired
    CategoryService categoryService;

    public void receiveMessage(byte[] bytes) throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        RabbitMQMessage message = mapper.readValue(new String(bytes),
RabbitMQMessage.class);
        List<Category> categories =
Arrays.asList(mapper.readValue(message.getBody(), Category[].class));
        if ("post".equals(message.getMicroserviceName())){
            switch (message.getOperation()){

                case "create":
                    for(Category category :categories){
                        setCategoryPostCategory(category);
                        categoryService.create(category);
                    }
                    break;

                case "update":
                    for(Category category :categories){
                        setCategoryPostCategory(category);
                        categoryService.update(category);
                    }
                    break;
            }
        }
    }

    private void setCategoryPostCategory(Category category) {
        for(PostCategory postCategory : category.getPostCategories()){
            postCategory.setCategory(category);
        }
    }
}
```

Ilustración 167 Clase Receiver del proyecto restful-blog-categories

## **Implementación de la fase IV**

Docker es una aplicación que permite hacer uso de la virtualización de contenedores que permite ejecutar los archivos distribuibles de aplicaciones o de los microservicios, antes de comenzar las labores de codificación se necesitará instalar docker en la máquina tal como se describe en la guía de instalación de docker. Los resultados de esta fase se pueden ver en la url <https://github.com/juanwalker/restful-blog-microservices-phase4>

A continuación se listan las labores que se deben de realizar para añadir el soporte de docker a los microservicios:

### **Creación de archivos Dockerfile para cada proyecto**

Es necesario construir un archivo Dockerfile el cual contendrá las instrucciones que se ejecutaran dentro de cada uno de los contenedores cada vez que se inicie un microservicio.

### **Creación de archivos docker-compose.yml**

El archivo docker-compose se encarga de hacer la orquestación de los microservicios, realizando empaquetamiento y arranque de los contenedores, ejecutando las instrucciones contenidas en cada archivo Dockerfile definido por proyecto.

### **Creación de nuevo perfil de configuración docker**

Por cada proyecto se necesitará definir un perfil de spring para la ejecución del microservicio dentro del contener docker, la parametrización desde de realizarse en los archivos de configuración.

### **Creación de archivos DockerFile por proyecto**

La base de cada proyecto necesita incluir un archivo llamado DockerFile, en este archivo se programará el contenedor para que copie el jar (archivo distribuible) de cada microservicio

y lo ejecute haciendo uso el JDK. La plantilla que se usuará para cada DockerFile será la siguiente:

```
FROM java:8
VOLUME /tmp
ADD target/restful-blog-configuration*.jar app.jar
RUN bash -c 'touch /app.jar'
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-
jar","/app.jar"]
```

**Ilustración 168 Plantilla de archivo DockerFile para los microservicios**

En cada proyecto se reemplazará la cadena de texto target/restful-blog-configuration por la correspondiente al nombre del jar generado en cada proyecto.

## Creación de archivo docker-compose.yml:

El archivo docker-compose-yml contiene un mapa de todos los contenedores necesarios para el correcto funcionamiento de la aplicación, en el se definen las rutas de construcción, mapeo de puertos, nombres de contenedores, enlaces entre contenedores, volúmenes de directorios, variables de entornos e imágenes. A continuación se muestra el contenido de este archivo.

```
rabbitmq:
    image: rabbitmq:management
    ports:
        - "5672:5672"
        - "15672:15672"

eureka:
    build: ../restful-blog-eureka
    ports:
        - "8761:8761"
    container_name: restful-blog-eureka
    environment:
        - JVM_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m

gogs:
    image: gogs/gogs
    container_name: gogs
    ports:
        - "8022:22"
        - "3000:3000"
    volumes:
        - "gogs:/data"

configuration:
    build: ../restful-blog-configuration
    ports:
        - "8888:8888"
    environment:
        - SPRING_PROFILES_ACTIVE=docker
        - JVM_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
    links:
        - eureka
        - gogs
    container_name: restful-blog-configuration

log:
    build: ../restful-blog-log
    environment:
        - SPRING_PROFILES_ACTIVE=docker
    links:
        - rabbitmq
        - restful_blog_log
```

```

credentials:
  build: ../restful-blog-credentials
  ports:
    - "8181:8181"
  environment:
    - SPRING_PROFILES_ACTIVE=docker
    - JVM_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
  container_name: restful-blog-credentials
  links:
    - eureka
    - configuration
    - restful_blog_auth
    - rabbitmq

  author:
  build: ../restful-blog-author
  ports:
    - "8282:8282"
  container_name: restful-blog-author
  environment:
    - SPRING_PROFILES_ACTIVE=docker
    - JVM_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
  links:
    - eureka
    - credentials
    - configuration
    - posts
    - restful_blog_author
    - rabbitmq

  posts:
  build: ../restful-blog-posts
  ports:
    - "8383:8383"
  container_name: restful-blog-posts
  environment:
    - SPRING_PROFILES_ACTIVE=docker
    - JVM_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
  links:
    - eureka
    - credentials
    - configuration
    - categories
    - restful_blog_posts
    - rabbitmq

  categories:
  build: ../restful-blog-categories
  ports:
    - "8484:8484"
  container_name: restful-blog-categories
  environment:
    - SPRING_PROFILES_ACTIVE=docker
    - JVM_OPTS=-Xmx512m -Xms512m -XX:MaxPermSize=128m
  links:

```

```

        - eureka
        - credentials
        - configuration
        - restful_blog_categories
        - rabbitmq

hystrix:
  build: ../restful-blog-monitor
  ports:
    - "8383:8383"
  environment:
    - SPRING_PROFILES_ACTIVE=docker
  links:
    - credentials
    - author
    - eureka

restful_blog_auth:
  image: mysql:latest
  container_name: restful_blog_auth
  ports:
    - 3307:3306
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: restful_blog_auth
  volumes:
    - "restful_blog_auth:/var/lib/mysql"

restful_blog_author:
  image: mysql:latest
  container_name: restful_blog_author
  ports:
    - 3308:3306
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: restful_blog_author
  volumes:
    - "restful_blog_author:/var/lib/mysql"

restful_blog_log:
  image: mysql:latest
  container_name: restful_blog_log
  ports:
    - 3309:3306
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: restful_blog_log
  volumes:
    - "restful_blog_log:/var/lib/mysql"

restful_blog_posts:
  image: mysql:latest
  container_name: restful_blog_posts
  ports:
    - 3310:3306
  environment:

```

```
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: restful_blog_posts
volumes:
    - "restful_blog_posts:/var/lib/mysql"
restful_blog_categories:
    image: mysql:latest
    container_name: restful_blog_categories
ports:
    - 3311:3306
environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: restful_blog_categories
volumes:
    - "restful_blog_categories:/var/lib/mysql"
```

**Ilustración 169 Archivo docker-compose.yml para la ejecución de los microservicios**

En este archivo se ha referenciado las imágenes binarias publicadas en docker-hub como lo es gogs (Servidor git), Mysql server y RabbitMQ. El resto de imágenes son construidas a partir de la compilación de los microservicios, también se definen los enlaces o *links* que permiten la comunicación entre contenedores y crean variables de entorno, cada contenedor se configura para el perfil docker por medio del parámetro SPRING\_PROFILES\_ACTIVE=docker

### Creación de perfiles docker en archivos de configuración

Los perfiles docker deberán concentrarse en la parametrización del descubrimiento de recursos externos a los microservicios como los son las instancias de bases de datos,el servidor de versionamiento, el servidor eureka, el servidor de configuración y el broker.A continuación se verá la manera como quedaron definidos los perfiles de docker en cada uno de los proyectos:

**restful-blog-eureka:** Se reemplazó la propiedad hostName por la variable que inyecta el link por medio de docker-composer.

```
spring:  
  profiles: docker  
eureka:  
  instance:  
    hostname: ${EUREKA_PORT_8761_TCP_ADDR:localhost}
```

**Ilustración 170 Inclusión de perfil docker en el archivo application.yml del proyecto restful-blog-eureka**

**restful-blog-configuration** : Se parametrizará el repositorio git y la url de eureka haciendo uso de las variables provistas por docker-composer.

```
spring:
  profiles: docker
  cloud:
    config:
      server:
        git:
          uri: http://${GOGS_PORT_3000_TCP_ADDR}:3000/
              juanwalker/restful_blog_configuration.git
eureka:
  client:
    serviceUrl:
      defaultZone: http://${EUREKA_PORT_8761_TCP_ADDR}:8761/eureka/
instance:
  hostname: ${EUREKA_PORT_8761_TCP_ADDR:localhost}
  preferIpAddress: true
```

Ilustración 171 Perfil docker en archivo application.yml del proyecto restful-blog-configuration

```
spring:
  profiles: docker
  rabbitmq:
    port: 5672
    host: ${RABBITMQ_PORT_5672_TCP_ADDR}
    username: guest
    password: guest
  datasource:
    url:
      jdbc:mysql://${RESTFUL_BLOG_LOG_PORT_3306_TCP_ADDR}/restful_blog_log
      username: root
      password: root
eureka:
  client:
    serviceUrl:
      defaultZone: http://${EUREKA_PORT_8761_TCP_ADDR}:8761/eureka/
instance:
  preferIpAddress: true
```

Ilustración 172 Inclusión de perfil docker en el archivo application.yml para los microservicios log, credential,author, post y category

**restful-blog-log, restful-blog-credentials, restful-blog-author, restful-blog-posts, restful-blog-categories:** Se parametrizarán los enlaces de conexión al broker, eureka y a la base de datos haciendo uso de las variables inyectadas por docker-composer

También se modificará la política de reintentos para verificar el spring cloud configuration de tal manera que cada microservicio espere a que el servidor de configuración esté arriba y corriendo

```
spring:  
  profiles: docker  
  cloud:  
    config:  
      uri: http://restful-blog-configuration:8888  
      failFast: true  
      retry:  
        maxAttempts: 9999  
      discovery:  
        enabled: true
```

**Ilustración 173 Inclusión de perfil docker en el archivo bootstrap.yml para los microservicios log, credential,author, post y category**

Luego de haber modificado los archivos se procederá a preparar la base de datos mysql y el servidor git, para ello se abrirá una consola del docker toolbox y será necesario ubicarse dentro del folder docker del workspace y se ejecutará los siguientes comandos.

```
docker-compose build  
  
docker-compose up gogs restful_blog_auth  
restful_blog_author restful_blog_log restful_blog_posts  
restful_blog_categories
```

**Ilustración 174 Comandos utilizados para ejecutar bases de datos por medio de docker-compose**

Cada una de las bases de datos levantadas estarán disponibles desde la máquina de docker, por defecto su ip es 192.168.99.100 sin embargo se podrá verificarla por medio del

comando *docker-machine ip*. Se accederá desde el mysql workbench a cada base de datos de manera separada y se ejecutarán los script sql que se usan en las anteriores fases en la máquina local, dejando de esta manera listas las bases de datos antes de ejecutar todos los microservicios.

También será necesario hacer un push de los archivos git en el nuevo servidor git tal como se hizo en las anteriores fases (Un archivo .properties por cada microservicio) de tal manera que el servidor de configuración quede preparado para ser usado por todos los microservicios.

Finalmente se detendrá la anterior ejecución deteniéndola desde la consola, y se ejecutarán nuevamente el comando *docker-compose up* lo que levantara todos los microservicios.

## Implementación de la fase V

Hasta ahora se ha utilizado solamente dos ambientes para la ejecución de los microservicios, el ambiente default sirve para hacer las pruebas y ejecuciones en la máquina local, el ambiente docker el cual permite hacer despliegues al vuelo y hacer pruebas funcionales de la correcta comunicación entre microservicios. Sin embargo es necesario ofrecer otros ambientes que permitan compartir los artefactos a otros stakeholders con la intención de ser usados o validados antes de su salida a producción. Para este caso de estudio vamos a definir 4 nuevos ambientes de despliegue que se explican a continuación:

Identificador	Nombre	Descripción
dev	Development	Servirá para que el equipo de desarrollo pueda sincronizar el código fuente (Subidas o descargas) y realizar pruebas de integración entre los diferentes microservicios.
test	Testing	Los artefactos encontrados aquí servirán para que los <i>testers</i> del proyecto puedan verificar que el sistema cumple con la especificación requerimiento
qa	Quality Assurance	Este entorno será por el cliente para verificar que las funcionalidades solicitadas en el contrato se estén cumpliendo en el desarrollo.

prod	Production	Entorno final de los despliegues, es donde la aplicación ejecutará sus operaciones reales.
------	------------	--

**Tabla 5 Ambientes de despliegue de la aplicación**

Los entornos estarán conectados en forma de cascada lo que hará obligatorio el paso por cada uno antes de su salida a producción, **dev** será el entorno de punto de partida y necesita por lo tanto los accesos a los repositorios del código fuente de cada microservicio que utilizará para compilar el código fuente.

Será necesario por lo tanto que crear un repositorio local git para cada uno de los microservicios desde la interfaz gráfica de gitstack definiéndolos de la siguiente manera:

Nombre del Proyecto	Url del repositorio
<i>restful_blog_author</i>	<a href="http://localhost/restful_blog_author.git">http://localhost/restful_blog_author.git</a>
<i>restful_blog_categories</i>	<a href="http://localhost/restful_blog_categories.git">http://localhost/restful_blog_categories.git</a>
<i>restful_blog_configuration</i>	<a href="http://localhost/restful_blog_configuration.git">http://localhost/restful_blog_configuration.git</a>
<i>restful_blog_credentials</i>	<a href="http://localhost/restful_blog_credentials.git">http://localhost/restful_blog_credentials.git</a>
<i>restful_blog_eureka</i>	<a href="http://localhost/restful_blog_eureka.git">http://localhost/restful_blog_eureka.git</a>
<i>restful_blog_log</i>	<a href="http://localhost/restful_blog_log.git">http://localhost/restful_blog_log.git</a>
<i>restful_blog_monitor</i>	<a href="http://localhost/restful_blog_monitor.git">http://localhost/restful_blog_monitor.git</a>
<i>restful_blog_posts</i>	<a href="http://localhost/restful_blog_posts.git">http://localhost/restful_blog_posts.git</a>

**Tabla 6 Repositorios por proyectos necesarios para la entrega continua**

Posteriormente se subirá el código fuente al repositorio ejecutando los siguientes comandos desde cada proyecto del workspace por medio de la consola.

```
git init  
git add *  
git commit -m "first commit"  
git remote add origin remote repository url del  
repositorio  
git push origin master
```

**Ilustración 175 Comandos git usados para la inicialización de un repositorio remoto**

En este punto se debe de tener instaladas, configurados y corriendo las aplicaciones Jenkins, Nexus ya que son necesarias para la el proceso de entrega continua.

Será necesario ingresar a la url de nexus <http://localhost:8081/nexus/> y crear 4 nuevos repositorios, luego loguearse como administrador y hacer click en la opción add Hosted Repository. Cada uno de los repositorios deberán de crearse con el *Repository Name* y el *Repository Type* con base en los datos que se encuentran en la tabla X, deberan ser creado con el *Repository Policy* como *Release* y con el Deployment Policy con valor *Allow Redeploy*.

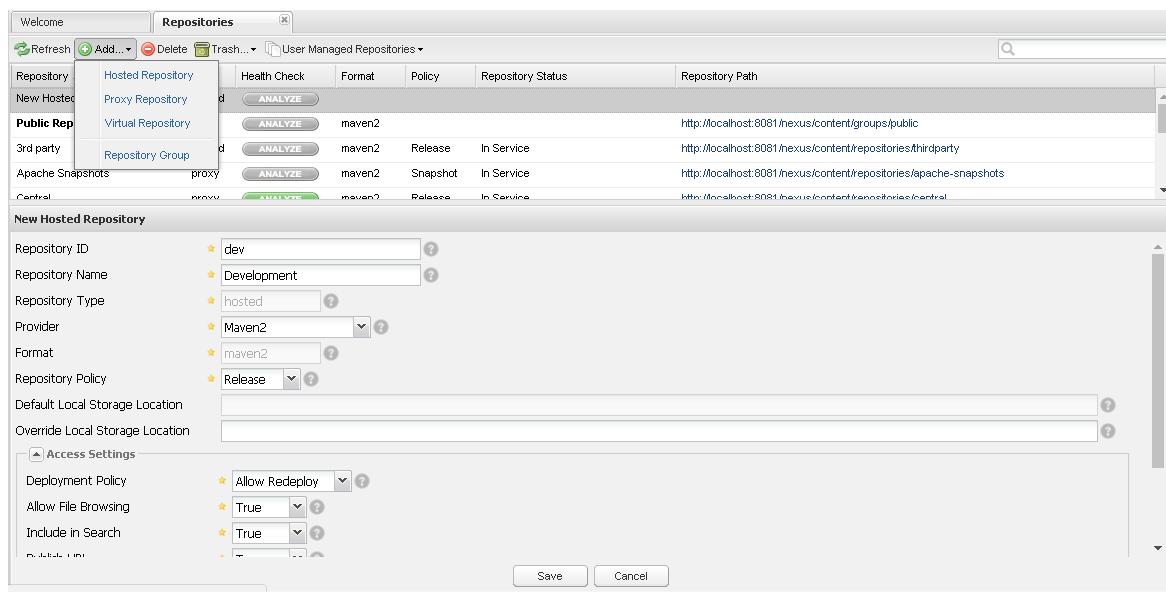


Ilustración 176 Repositorios dev,test,qa y prod en nexus

Después de haber creado los repositorios se debe de crear los trabajos en jenkins que construirán y promoverán los artefactos para que sean copiados en los repositorios.

Se debe de ingresar a la url de jenkins <http://localhost:8080/> y crear una tarea de tipo *build* y otra de tipo *promote* por proyecto las cual serán las encargadas de construir y de promocionar el artefacto a los diferentes ambientes de despliegue. La tarea tipo build se nombrará *restful-blog-proyecto-build* y tendrá la siguiente configuración:

Proyecto nombre	<input type="text" value="restful-blog-author-build"/>
Descripción	<input type="text" value=""/>
<a href="#">[Plain text]</a> <a href="#">Visualizar</a>	

**Ilustración 177 Creación de tarea en jenkins tipo build : definición de nombre**

Será necesario parametrizar el repositorio del código fuente que será utilizado por la tarea, se usarán los repositorios utilizados por cada microservicio

### Configurar el origen del código fuente

Ninguno  
 Git

**Repositories**

Repository URL	<input type="text" value="http://localhost/restful_blog_author.git"/>	<a href="#">?</a>
Credentials	<input type="text" value="juanwalker***** (GitStack localhost)"/>	<a href="#">Add</a>
<a href="#">Avanzado...</a>		
<a href="#">Add Repository</a>		

**Branches to build**

Branch Specifier (blank for 'any')	<input type="text" value="*/master"/>	<a href="#">X</a>	<a href="#">?</a>
<a href="#">Add Branch</a>			

**Navegador del repositorio**

<input type="text" value="(Auto)"/>	<a href="#">?</a>
-------------------------------------	-------------------

**Additional Behaviours**

<a href="#">Añadir</a>	<a href="#">▼</a>	<a href="#">?</a>
------------------------	-------------------	-------------------

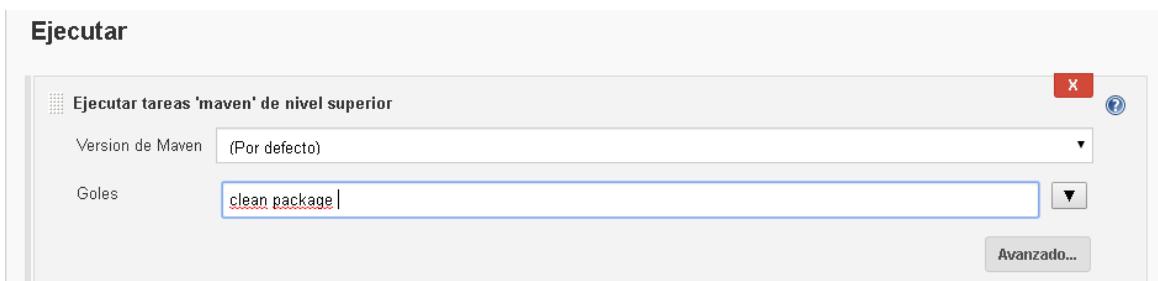
**Ilustración 178 Creación de tarea en jenkins tipo build: asociación de repositorio git**

Se debe de configurar la tarea para que verifique cada 5 minutos si ha existido algún cambio en el repositorio y de ser así que construya un nuevo artefacto



**Ilustración 179 Creación de tarea en jenkins tipo build: disparador de ejecuciones de repositorio git**

En cada ejecución de la tarea esta ejecutará los objetivos clean y package para generar el archivo jar, es necesario tener los microservicio de configuración arriba, de lo contrario se debe de adicionar en los objetivos la bandera –DskipTests.



**Ilustración 180 Creación de tarea en jenkins tipo build: Ejecución de tareas tipo maven**

Será necesario asociar el repositorio nexus deployment con la generación de los artefactos, también se debe de definir el id del artefacto, el grupo y la ruta de donde se carga el artefacto.

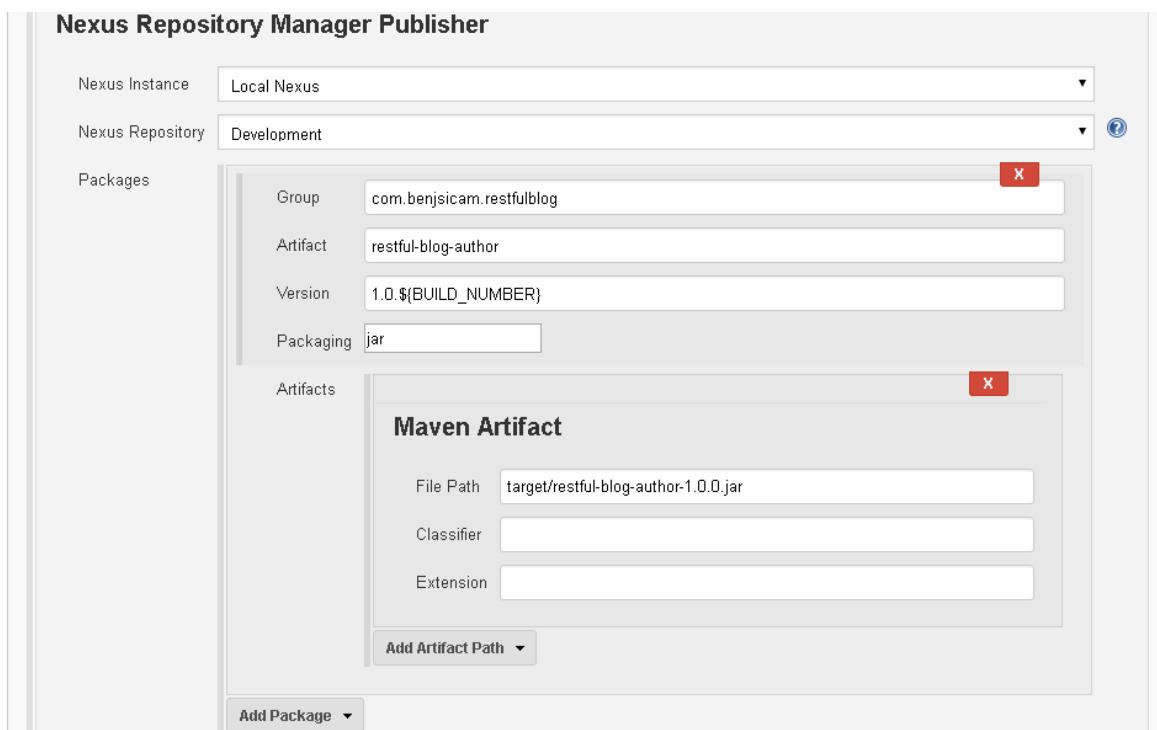


Ilustración 181 Creación de tarea en jenkins tipo build: Administrador de repositorios nexus

Será necesario definir la tarea encargada de hacer las promociones de los artefactos construidos Y se denominará restful-blog-proyecto-promote

The screenshot shows the initial configuration screen for a Jenkins job. It includes fields for 'Proyecto nombre' (set to 'restful-blog-author-promote') and 'Descripción'. At the bottom, there are links for '[Plain text]' and '[Visualizar]'. The entire form is enclosed in a light gray border.

**Ilustración 182 Creación de tarea en jenkins tipo promoción: Definición de nombre de tarea**

The screenshot shows two configuration sections for a Jenkins promotion job. The top section, titled 'Elección', defines a dropdown parameter named 'DEST\_ENV' with options 'test', 'qa', and 'prod'. The bottom section, titled 'Parámetro de cadena', defines a string parameter named 'VERSION' with a default value field and a description field. Both sections have a red 'X' button in the top right corner and a 'Plain text' link at the bottom.

**Ilustración 183 Creación de tarea en jenkins tipo promoción: Definición de lista desplegable de entornos**

Se debe de definir las propiedades GROUP\_ID y ARTIFACT\_ID necesarias para el script bash



Ilustración 184 Creación de tarea en jenkins tipo promoción: Definición de propiedades de contenido

Se define un script que permita hace la promoción de los artefactos

```
GROUP_PATH=`echo ${GROUP_ID} | sed 's/\./\//g'`
if [ $DEST_ENV == 'test' ]
then
    SOURCE_ENV=dev
fi
if [ $DEST_ENV == 'qa' ]
then
    SOURCE_ENV=test
fi
if [ $DEST_ENV == 'prod' ]
then
    SOURCE_ENV=qa
fi

wget -nd -np -k -r
$NEXUS_URL/content/repositories/$SOURCE_ENV/$GROUP_PATH/$ARTIFACT_ID/$VERSION/

for entry in *
do
    if [ "$entry" != "script.sh" ]
    then
        curl --insecure -u $NEXUS_USER:$NEXUS_PASSWORD --
upload-file "$entry"
$NEXUS_URL/content/repositories/$DEST_ENV/$GROUP_PATH/$ARTIFACT_ID/$VERSION/"$entry"
        rm "$entry"
    fi
done
echo "Promotion successfully."
exit 0
```

Ilustración 185 Tarea jenkins tipo promoción: Script de promoción de artefactos

Para verificar el funcionamiento se lanzará un *build* y posteriormente una tarea de *promotion* en cada uno de los ambientes (dev,test,qa y prod) de tal manera que se pueda verificar el funcionamiento de la entrega continua correctamente. Para dicha labor se tomará un microservicio de ejemplo, para este caso se tomará restful-blog, y se realizará un build ejecutando la tarea restful-blog-author-build y se procederá a construir.

The screenshot shows the Jenkins interface for the project "restful-blog-author-build". The left sidebar includes links for "Volver al Panel de Control", "Estado Actual", "Cambios", "Zona de Trabajo", "Construir ahora", "Borrar Proyecto", "Configurar", "Git Log de consultas", and "Move". The main content area displays the title "Proyecto restful-blog-author-build" and "Enlaces permanentes" with icons for "Espacio de trabajo" and "Cambios recientes". To the right is a "Historia de tareas" panel showing the following build history:

Número	Fecha
#8	27-feb-2017 18:06
#7	26-feb-2017 11:11
#6	26-feb-2017 3:01
#5	26-feb-2017 2:58
#4	26-feb-2017 2:54
#3	26-feb-2017 2:51
#2	26-feb-2017 2:14
#1	25-feb-2017 23:39

At the bottom of the history panel are links for "RSS Para Todos" and "RSS para los errores".

Ilustración 186 Tarea restful-blog-author-build antes de ser ejecutada

Al terminar la tarea, se encontrará que la ejecución se realizó exitosamente, agregando una nueva ejecución con identificador 9:

This screenshot shows the same Jenkins project page after a new build was executed. The build history now includes an additional entry at the top:

Número	Fecha
#9	04-mar-2017 22:07
#8	27-feb-2017 18:06
#7	26-feb-2017 11:11
#6	26-feb-2017 3:01
#5	26-feb-2017 2:58
#4	26-feb-2017 2:54
#3	26-feb-2017 2:51
#2	26-feb-2017 2:14
#1	25-feb-2017 23:39

Ilustración 187 Tarea restful-blog-author-build después de ser ejecutada

Se encontrará que el artefato que se acaba de construir se encuentre en el repositorio dev con versión 1.0.9

### **Index of /repositories/dev/com/benjsicam/restfulblog/restful-blog-author/1.0.9**

Name	Last Modified	Size	Description
<a href="#">Parent Directory</a>			
<a href="#">restful-blog-author-1.0.9.jar</a>	Sat Mar 04 22:08:50 COT 2017	56155889	
<a href="#">restful-blog-author-1.0.9.jar.md5</a>	Sat Mar 04 22:08:51 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.jar.sha1</a>	Sat Mar 04 22:08:51 COT 2017	40	
<a href="#">restful-blog-author-1.0.9.pom</a>	Sat Mar 04 22:08:50 COT 2017	480	
<a href="#">restful-blog-author-1.0.9.pom.md5</a>	Sat Mar 04 22:08:50 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.pom.sha1</a>	Sat Mar 04 22:08:50 COT 2017	40	

**Ilustración 188 Artefacto restful-blog-author con versión 1.0.9 en el repositorio dev de nexus**

Posteriormente se realizará la promoción del artefacto con versión 1.0.9 a los ambientes test luego al ambiente qa y finalmente al ambiente prod y de esta manera verificar que la entrega continua se esté realizando de manera exitosa.

### **Proyecto restful-blog-author-promote**

Esta ejecución requiere parámetros adicionales:

DEST_ENV	<input type="text" value="test"/>
VERSION	<input type="text" value="1.0.9"/>
<b>Ejecución</b>	

### **Index of /repositories/test/com/benjsicam/restfulblog/restful- blog-author/1.0.9**

Name	Last Modified	Size	Description
<a href="#">Parent Directory</a>			
<a href="#">index.html</a>	Sun Mar 05 10:54:33 COT 2017	2902	
<a href="#">restful-blog-author-1.0.9.jar</a>	Sun Mar 05 10:54:34 COT 2017	56155889	
<a href="#">restful-blog-author-1.0.9.jar.md5</a>	Sun Mar 05 10:54:35 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.jar.sha1</a>	Sun Mar 05 10:54:36 COT 2017	40	
<a href="#">restful-blog-author-1.0.9.pom</a>	Sun Mar 05 10:54:36 COT 2017	480	
<a href="#">restful-blog-author-1.0.9.pom.md5</a>	Sun Mar 05 10:54:36 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.pom.sha1</a>	Sun Mar 05 10:54:37 COT 2017	40	

**Ilustración 189 Artefacto restful-blog-author con versión 1.0.9 en el repositorio test de nexus**

Luego de haber promovido el artefacto al repositorio test y de haber cumplido satisfactoriamente las pruebas funcionales se procederá a promover el artefacto al repositorio de QA.

**Proyecto restful-blog-author-promote**

Esta ejecución requiere parámetros adicionales:

DEST\_ENV  VERSION

**Ejecución**

**Index of /repositories/qa/com/benjsicam/restfulblog/restful-blog-author/1.0.9**

Name	Last Modified	Size	Description
<a href="#">Parent Directory</a>			
<a href="#">index.html</a>	Sun Mar 05 11:18:25 COT 2017	3173	
<a href="#">index.html_1</a>	Sun Mar 05 11:18:26 COT 2017	2902	
<a href="#">restful-blog-author-1.0.9.jar</a>	Sun Mar 05 11:18:26 COT 2017	56155889	
<a href="#">restful-blog-author-1.0.9.jar.md5</a>	Sun Mar 05 11:18:28 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.jar.sha1</a>	Sun Mar 05 11:18:28 COT 2017	40	
<a href="#">restful-blog-author-1.0.9.pom</a>	Sun Mar 05 11:18:29 COT 2017	480	
<a href="#">restful-blog-author-1.0.9.pom.md5</a>	Sun Mar 05 11:18:29 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.pom.sha1</a>	Sun Mar 05 11:18:29 COT 2017	40	

**Ilustración 190 Artefacto restful-blog-author con versión 1.0.9 en el repositorio qa de nexus**

El ambiente QA le permitirá al cliente verificar que las funcionalidades desplegadas y probadas en los ambientes anteriores cumplen de manera satisfactoria con lo contratado. De ser certificada la entrega por parte del cliente, se realizará el despliegue a producción.

**Proyecto restful-blog-author-promote**

Esta ejecución requiere parámetros adicionales:

DEST\_ENV  VERSION

**Ejecución**

**Index of /repositories/prod/com/benjsicam/restfulblog/restful-blog-author/1.0.9**

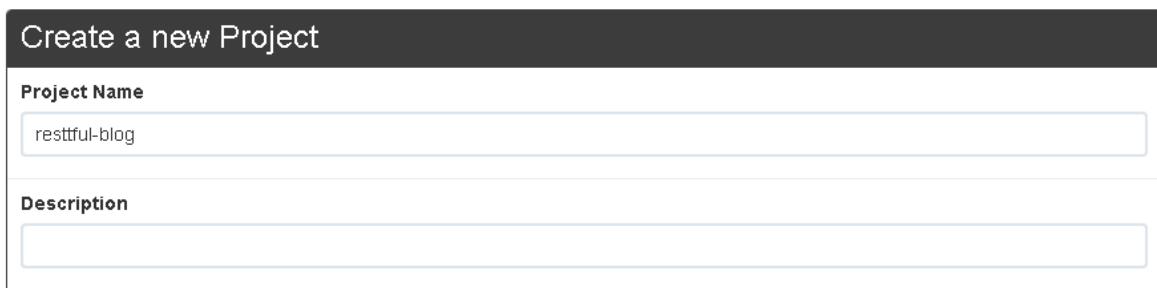
Name	Last Modified	Size	Description
<a href="#">Parent Directory</a>			
<a href="#">index.html</a>	Sun Mar 05 11:22:31 COT 2017	3439	
<a href="#">index.html_1</a>	Sun Mar 05 11:22:31 COT 2017	3175	
<a href="#">index.html_1.1</a>	Sun Mar 05 11:22:32 COT 2017	2902	
<a href="#">restful-blog-author-1.0.9.jar</a>	Sun Mar 05 11:22:32 COT 2017	56155889	
<a href="#">restful-blog-author-1.0.9.jar.md5</a>	Sun Mar 05 11:22:33 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.jar.sha1</a>	Sun Mar 05 11:22:34 COT 2017	40	
<a href="#">restful-blog-author-1.0.9.pom</a>	Sun Mar 05 11:22:34 COT 2017	480	
<a href="#">restful-blog-author-1.0.9.pom.md5</a>	Sun Mar 05 11:22:34 COT 2017	32	
<a href="#">restful-blog-author-1.0.9.pom.sha1</a>	Sun Mar 05 11:22:34 COT 2017	40	

**Ilustración 191 Artefacto restful-blog-author con versión 1.0.9 en el repositorio prod de nexus**

## Instalación de los artefactos en los entornos de despliegue

Hasta ahora se ha preparado para que de manera automática se realice la construcción de los artefactos y la asistencia en la promoción de los mismos a los repositorios de los diferentes ambientes. Sin embargo aún es necesario completar la última etapa de la entrega continua que permita hacer los despliegues directamente a los servidores. Para ello se hará uso de la aplicación rundeck especializada en ejecutar procedimientos que se pueden parametrizar desde una interfaz web.

Se debe de ingresar a la url local <http://localhost:4440> y crear un proyecto nuevo, haciendo click en el botón *New Project +*, el cual se llamará restful-blog y se dejarán todas las opciones por defecto con las que vienen en el formulario finalizando la generación del mismo haciendo click en el botón *create* al final del formulario.



The screenshot shows a 'Create a new Project' dialog box. At the top, it says 'Create a new Project'. Below that, there are two input fields. The first field is labeled 'Project Name' and contains the value 'restful-blog'. The second field is labeled 'Description' and is currently empty. Both fields have a thin grey border. The overall background of the dialog is white.

Ilustración 192 Creación de nuevo proyecto en rundeck

Luego de crear el proyecto, se creará un nuevo trabajo que permita a los usuarios autorizados poder hacer despliegues desde la interfaz web, para ellos es necesario click en el nombre del proyecto y posteriormente seleccionar la opción *+New Job* desde el menú desplegable *Job Actions*.

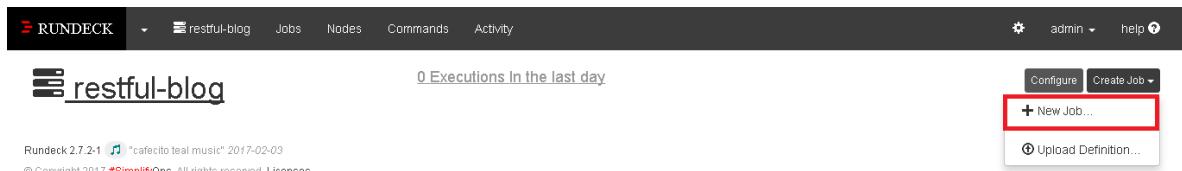


Ilustración 193 Navegación para crear nuevo trabajo en rundeck

Para este *job* se creará un formulario desde los usuarios puedan seleccionar la aplicación, el entorno desde donde desean desplegar y la versión que van a desplegar. Para ello se parametrizará los 3 valores, los cuales son fijos a excepción de la versión del artefacto la cual se cargarán desde el servidor nexus que compartirá por medio de un servicio rest de consulta las versiones disponibles de una aplicación en un repositorio determinado. Para definir el job con las características anteriormente mencionadas, se ingresarán los siguientes valores en las plantillas:

The screenshot shows the 'Edit Job' interface. The 'Job Name' field contains the value 'Restful Blog Microservices Deployment'. The 'Description' section is currently empty. A note at the bottom states: 'The first line of the description will be shown in plain text, the rest will be rendered with Markdown. More >'.

**Ilustración 194 Asignación de nombre al nuevo trabajo**

The screenshot shows the 'Edit Option' interface. The 'Option Name' is 'version'. The 'Description' field contains the value '1'. A note at the bottom states: 'The description will be rendered with Markdown.' Below the description, there are sections for 'Default Value' (empty), 'Input Type' (set to 'Plain text'), and 'Allowed Values' (set to 'List'). The URL listed is 'http://localhost:8081/nexus/service/local/rundeck/options/version?a=\${option.application.value}&r=\${option.environment.value}'. The 'Restrictions' section is empty.

**Ilustración 195 Creación del parámetro versión y asignación de url de servicio rest trabajo restful-blog-deploy en rundeck**

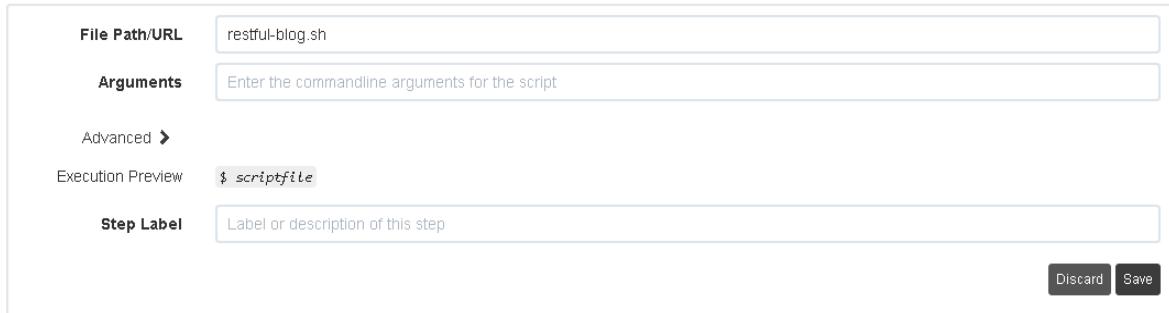
<b>Option Name</b>	application
<b>Description</b>	<p>1</p> <p>The description will be rendered with Markdown. <a href="#">?</a></p>
<b>Default Value</b>	restful-blog-eureka
<b>Input Type</b>	<input type="radio"/> Plain text <input type="radio"/> Date The date will pass to your job as a string formatted this way: mm/dd/yy HH:MM <input type="radio"/> Secure <small>†</small> Password input, value exposed in scripts and commands. <input type="radio"/> Secure Remote Authentication <small>†</small> Password input, value not exposed in scripts or commands, used only by Node Executors for authentication. <small>†</small> Secure input values are not stored by Rundeck after use. If the exposed value is used in a script or command then the output log may contain the value.
<b>Allowed Values</b>	<input type="radio"/> List <input checked="" type="radio"/> Remote URL restful-blog-author,restful-blog-categories,restful-blog-configuration,restful-blog-credentials,restful-blog-eureka,restful-blog-log,restful-blog-project,restful-blog-user
<b>Restrictions</b>	<input type="radio"/> None Any values can be used <input type="radio"/> Enforced from Allowed Values <input type="radio"/> Match Regular Expression
<b>Required</b>	<input type="radio"/> No <input checked="" type="radio"/> Yes

**Ilustración 196 Creación de parámetro application el el trabajo restful-blog-deploy en rundeck**

<b>Option Name</b>	enviroment
<b>Description</b>	<p>1</p> <p>The description will be rendered with Markdown. <a href="#">?</a></p>
<b>Default Value</b>	dev
<b>Input Type</b>	<input type="radio"/> Plain text <input type="radio"/> Date The date will pass to your job as a string formatted this way: mm/dd/yy HH:MM <input type="radio"/> Secure <small>†</small> Password input, value exposed in scripts and commands. <input type="radio"/> Secure Remote Authentication <small>†</small> Password input, value not exposed in scripts or commands, used only by Node Executors for authentication. <small>†</small> Secure input values are not stored by Rundeck after use. If the exposed value is used in a script or command then the output log may contain the value.
<b>Allowed Values</b>	<input type="radio"/> List <input checked="" type="radio"/> Remote URL dev,prod,qa,test
<b>Restrictions</b>	<input type="radio"/> None Any values can be used <input type="radio"/> Enforced from Allowed Values <input type="radio"/> Match Regular Expression
<b>Required</b>	<input type="radio"/> No <input checked="" type="radio"/> Yes

**Ilustración 197 Creación de parámetro enviroment el el trabajo restful-blog-deploy en rundeck**

Será necesario definir dentro de las tareas de ejecución o del workflow, la ejecución del script restful-blog.sh el cual contendrá el script de despliegue del artefacto de acuerdo a los parámetro ingresados por el usuario



**Ilustración 198 Referencia del script restful-blog-sh para ser ejecutado**

El script restful-blog.sh se encargará de bajar el artefacto seleccionado desde el repositorio con el comando wget, copiarlo a la ubicación seleccionada y finalmente borrar el artefacto descargado en la ruta de descargas.

```
wget "http://localhost:8081/nexus/service/local/artifact/maven/content?r=%RD_OPTION_ENVIRONMENT%&g=com.benjsicam.restfulblog&a=%RD_OPTION_APPLICATION%&v=%RD_OPTION_VERSION%" --content-disposition

copy "%RD_OPTION_APPLICATION%-%RD_OPTION_VERSION%.jar"
"rundeck_deployments/%RD_OPTION_ENVIRONMENT%_server/"

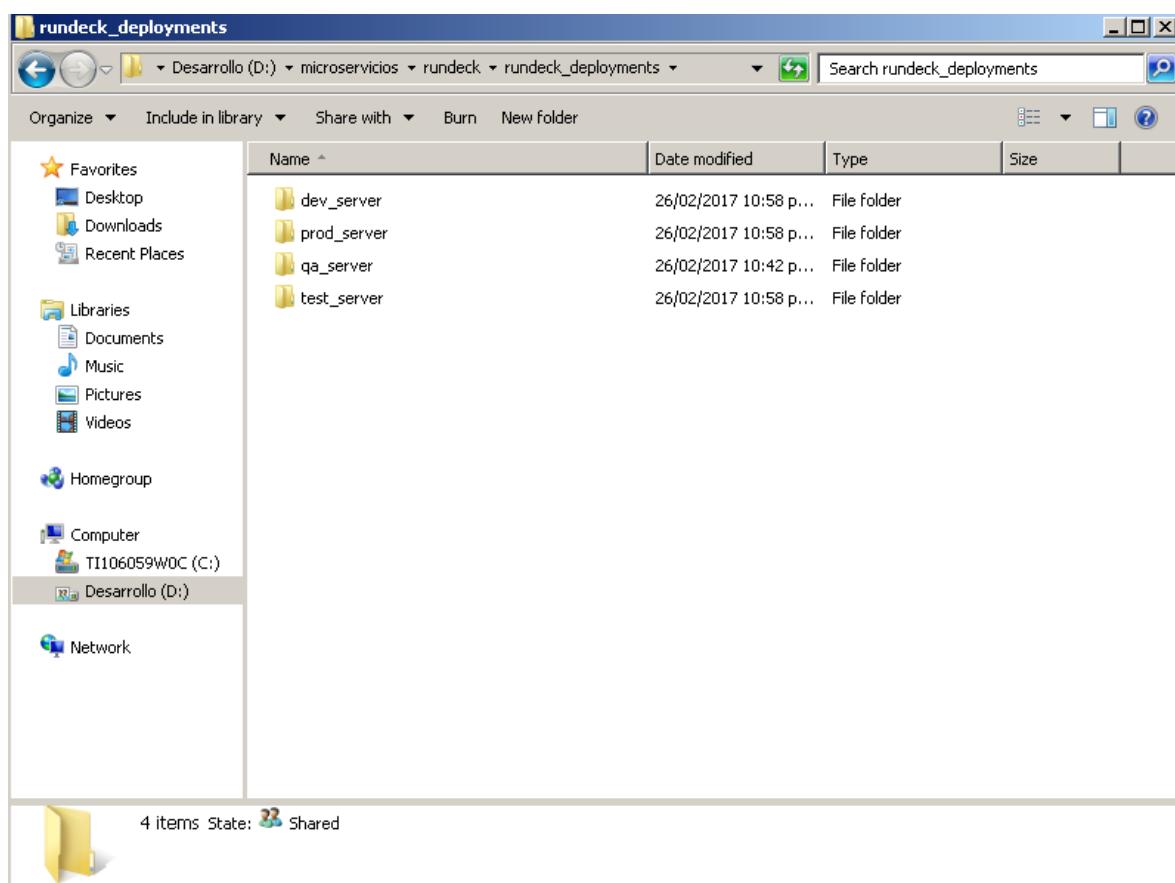
del "%RD_OPTION_APPLICATION%-%RD_OPTION_VERSION%.jar"

EXIT 0
```

**Ilustración 199 Contenido del script restful-blog.sh**

Para el correcto funcionamiento de este script se copió el archivo restful-blog.sh justamente en la base del directorio donde está instalado rundeck, además de una carpeta llamada rundeck\_deployments la cual contendrá la subcarpetas dev\_server, testserver.

Qa\_server y prod\_server que simulan sistemas de archivos de servidores remotos y que contendrán los despliegues. Para la configuración de despliegues en un sistema real probablemente ya el copiado de archivos no se realice por medio de un copiado de sistema de archivos sino por medio de un comando remoto como scp o sftp para el copiado de archivos a servidores remotos.



**Ilustración 200 Carpetas utilizadas por restful-blog.sh para la simulación de los despliegues**

De esta manera se terminar de configurar el proceso de entrega continua y despliegue que cumlmina el último paso para la implementación de la metodología.

## Anexos de la validación de la metodología

### Guía para la instalación de apache tomcat

1. Para iniciar es necesario haber descargado previamente un archivo .zip con la versión de Apache tomcat que se quiere utilizar. Se procederá a descomprimir en un directorio en el equipo, se utilizará para este propósito D:\microservicios junto con la versión Apache Tomcat 7.0.73

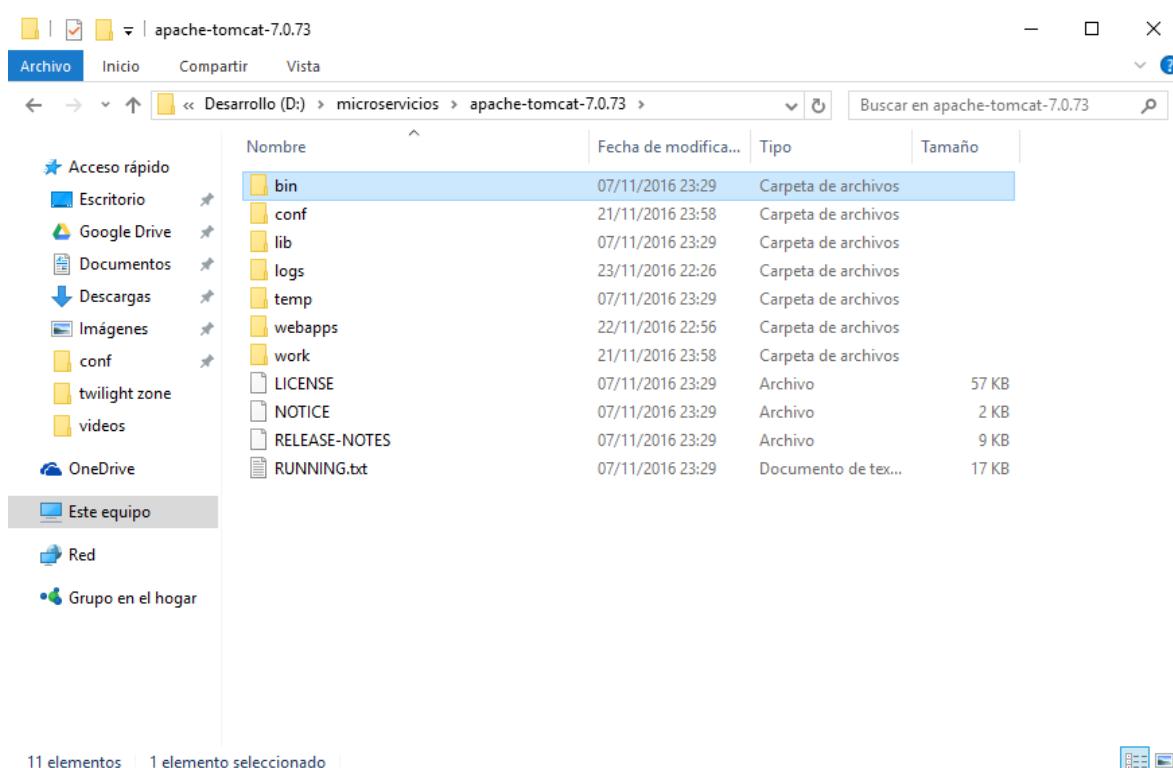


Ilustración 201 Carpeta base para instalación de apache tomcat

2. Será necesario crear un usuario administrador que tenga acceso a la consola de despliegue de aplicaciones, para ello se modificará el archivo D:\microservicios\apache-tomcat-7.0.73\conf\tomcat-users.xml

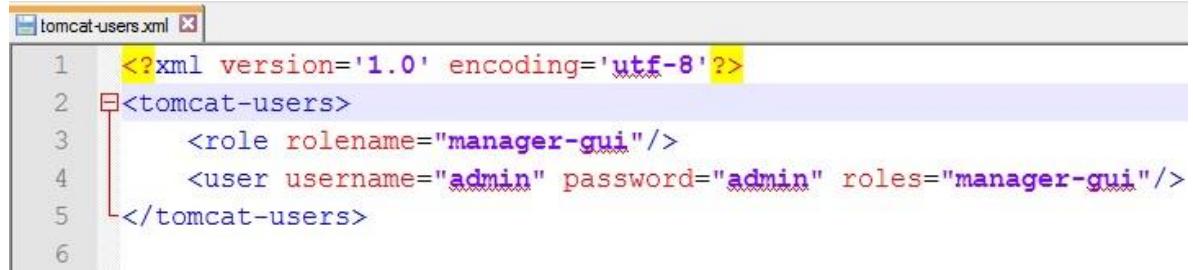
Haciendo uso del editor de texto de preferencia se adicionará al archivo D:\microservicios\apache-tomcat-7.0.73\conf\tomcat-users.xml las siguientes líneas xml:

```

<role rolename="manager-gui"/>
<user username="admin" password="admin" roles="manager-gui"/>

```

Lo cual permitirá adicionar un usuario admin con contraseña admin y con roles administrativos.

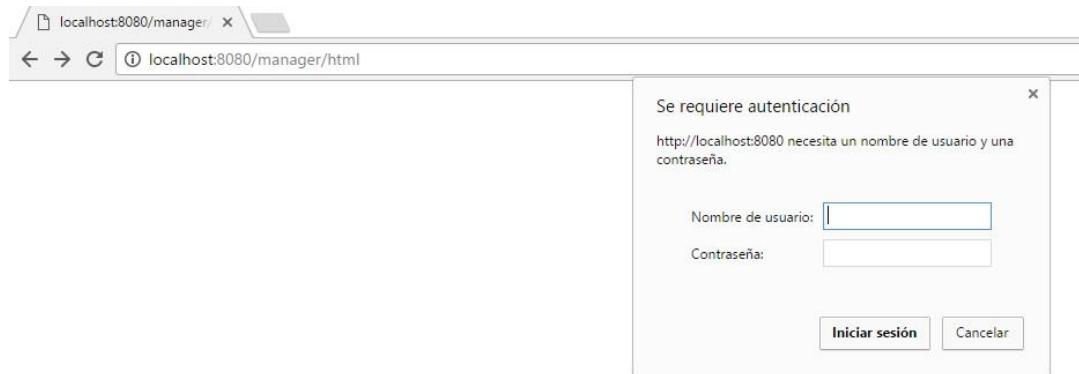


```

1  <?xml version='1.0' encoding='utf-8'?>
2  <tomcat-users>
3      <role rolename="manager-gui"/>
4      <user username="admin" password="admin" roles="manager-gui"/>
5  </tomcat-users>
6

```

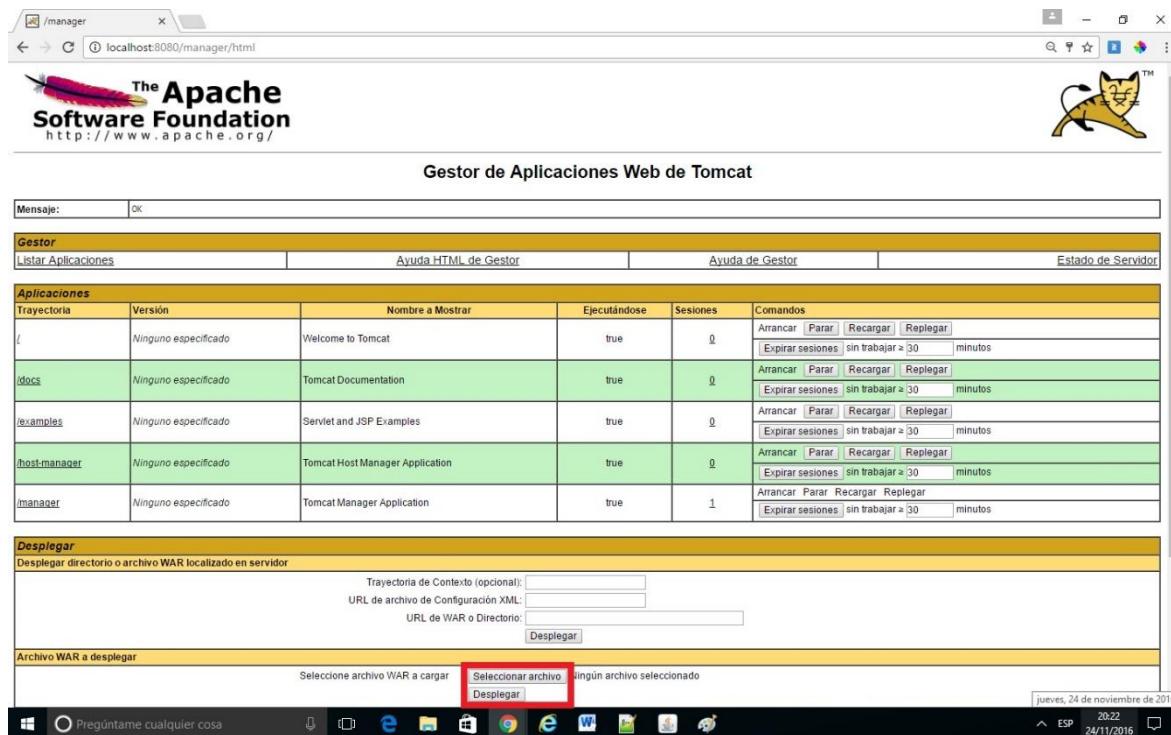
**Ilustración 202 Archivo de configuración de servidor tomcat, tomcat-users.xml**



**Ilustración 203 Autenticación de interfaz administrativa de apache tomcat**

Será necesario iniciar el servidor tomcat por medio del comando D:\microservicios\apache-tomcat-7.0.73\bin\startup.bat, lo cual iniciará el servidor y habilitará la consola para el manejo de los despliegues por medio de la siguiente url <http://localhost:8080/manager/html>, en caso que la url no sea accesible o no muestre una ventana de login deberá de verificar si existen otras aplicaciones que ya estén corriendo el puerto 8080 que viene por defecto.

3. En la ventana de login se deben de ingresar las credenciales anteriormente definidas en el archivo xml (admin/admin) lo que permitirá ingresar a la consola administrativa de apache tomcat.



**Ilustración 204 Interfaz administrativa de apache tomcat**

En esta ventana se podrá ver las aplicaciones web desplegadas en el servidor, su estado actual, los botones para el manejo de cada uno de los despliegues. También en los botones del recuadro rojo se podrá seleccionar el archivo .war o .ear y el botón desplegar que permitirá subir el archivo al servidor tomcat. Para hacer uso de él se subirá un archivo war y se desplegará y arrancará para verlo funcionar, se tomará de ejemplo el archivo que se ha usado como monolito .

/restful-blog-1.0	Ninguno especificado	restful-blog	true	0	Arrancar   Parar   Recargar   Replegar
					Expirar sesiones   sin trabajar ≥ 30 minutos

**Ilustración 205 Ejemplo de despliegue en apache tomcat de restful-blog**

4. Se procederá a probar la trayectoria que tomcat no genera en este caso es  
<http://localhost:8080/restful-blog-1.0/>



**Ilustración 206 Prueba de ejecución de aplicación restful-blog desplegada en apache tomcat**

## Guía para instalación de servidor y cliente Mysql

Será necesario bajar el instalador Mysql MSI desde la siguiente url <https://dev.mysql.com/downloads/mysql/> el cual proveerá de un asistente paso a paso para instalar el servidor y el cliente *workbench* para mysql. Se ejecutará el archivo instalador de mysql de donde se seleccionará una instalación tipo *Developer Default*

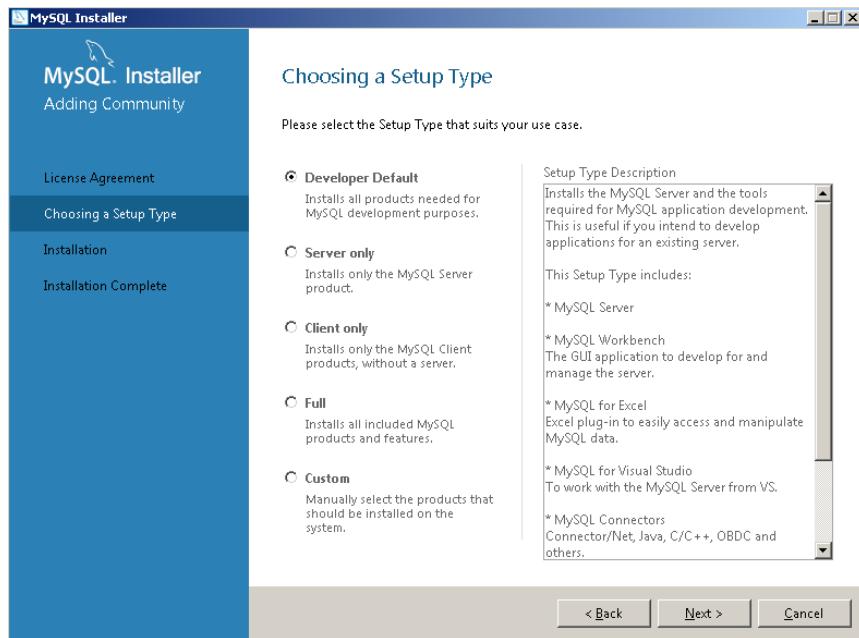


Ilustración 207 Asistente de instalación mysql: Escojer el tipo de instalación desarrollador

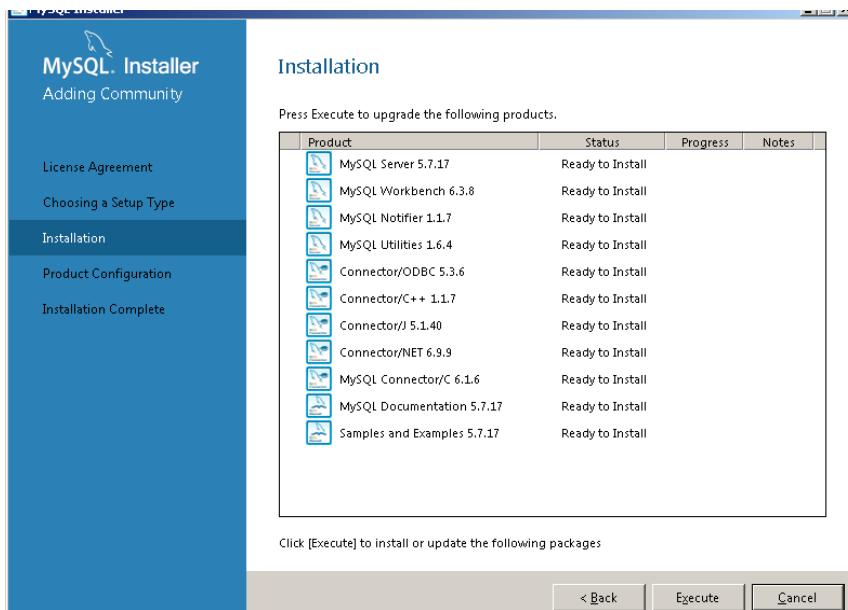
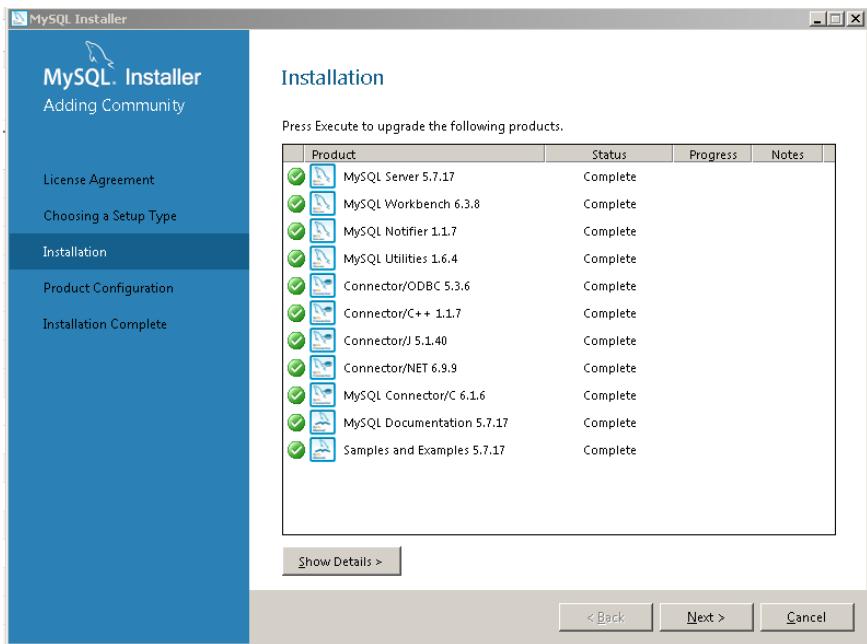


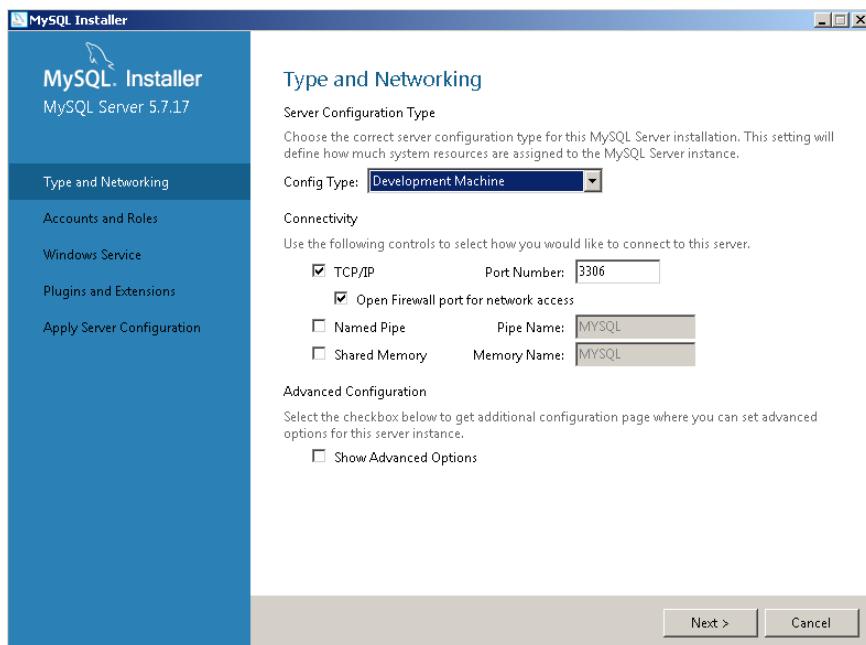
Ilustración 208 Asistente de instalación mysql: Componentes a instalar para el perfil desarrollador

Al finalizar la instalación, debe de salir la siguiente lista de chequeo



**Ilustración 209 Asistente de instalación mysql: Lista de chequeo después de la instalación de mysql**

Se configurará el servidor con las opciones por defecto y se definirá el password del usuario root con la palabra root.



**Ilustración 210 Asistente de instalación mysql: Configuración de red del servidor mysql**

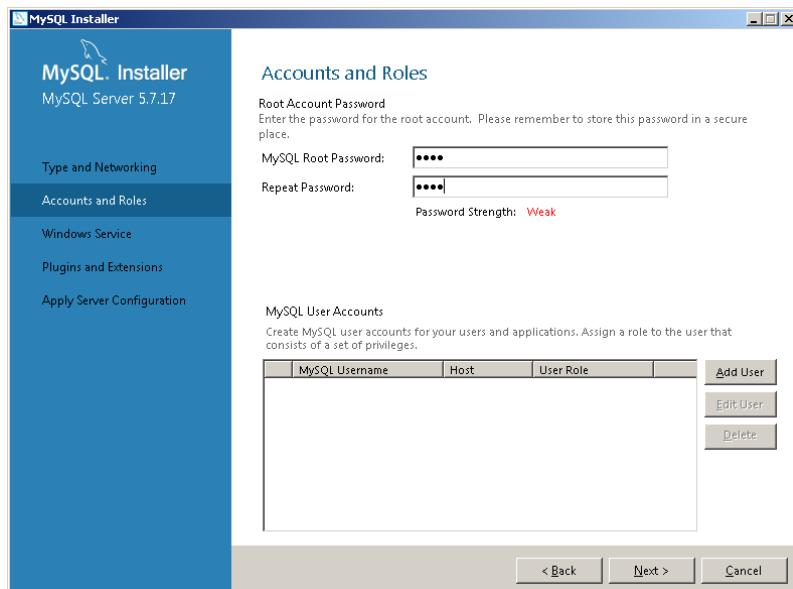


Ilustración 211 Asistente de instalación mysql: Asignación de contraseña de administrador

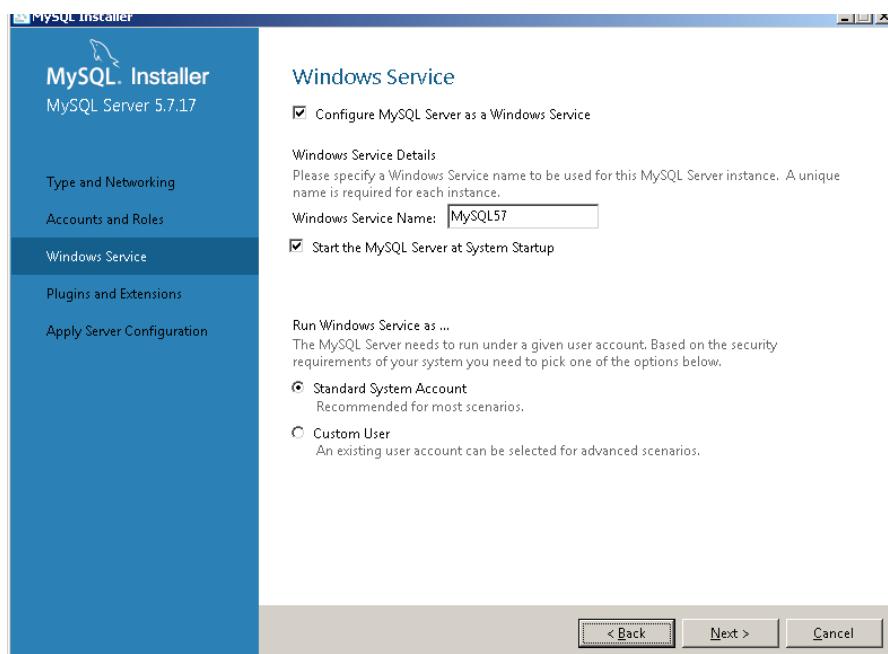


Ilustración 212 Asistente de instalación mysql: Configuración de servicio web de windows

Para verificar la conexión se abrirá el programa MySql Workbench se creará una nueva conexión para el servidor local y se verificará que la conexión creada se haga exitosamente.

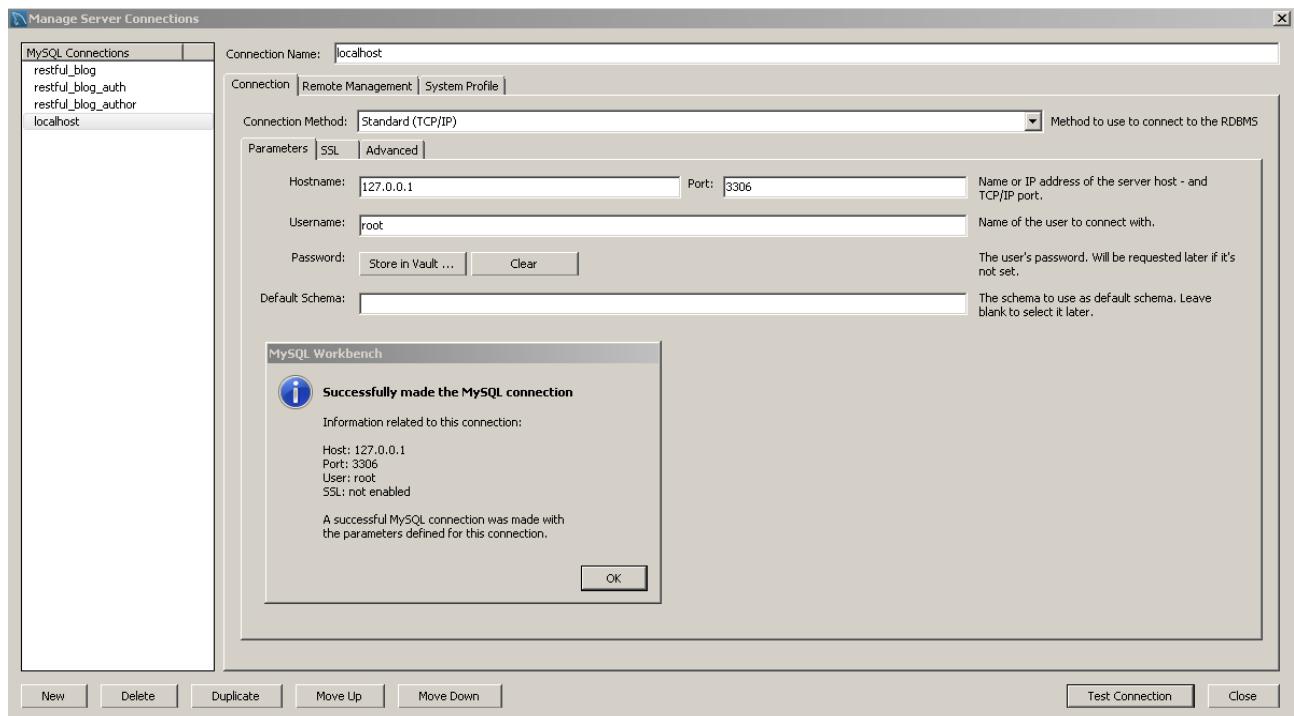


Ilustración 213 Configuración de conexión a servidor mysql desde wokbench

## Guía para instalación de GitStack

Es necesario bajar el instalador de gitstack desde la siguiente url  
<http://gitstack.com/download/> y ejecutar el asistente de instalación

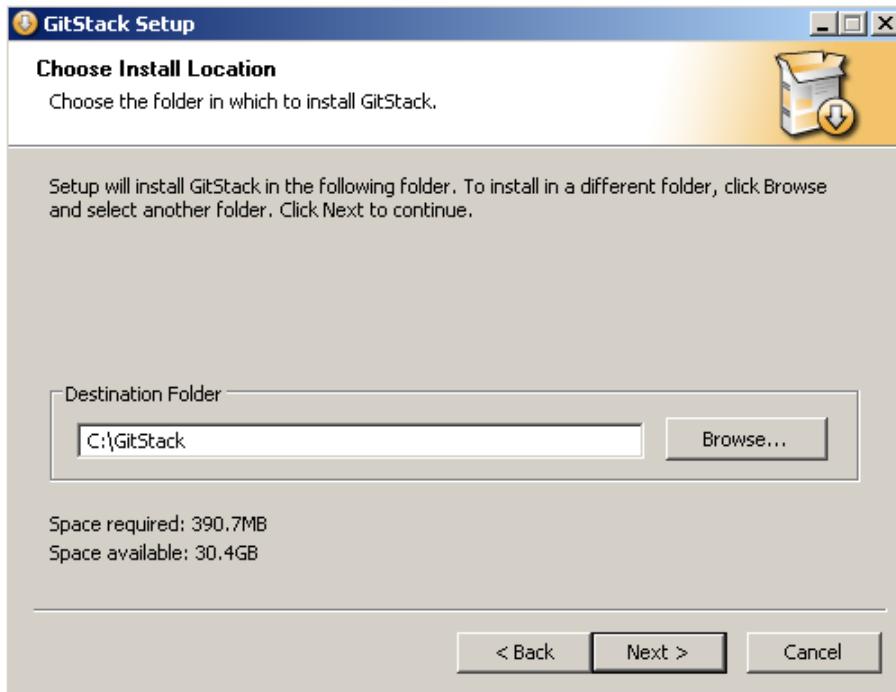


Ilustración 214 Asistente de instalación de gitstack: elección de la ruta

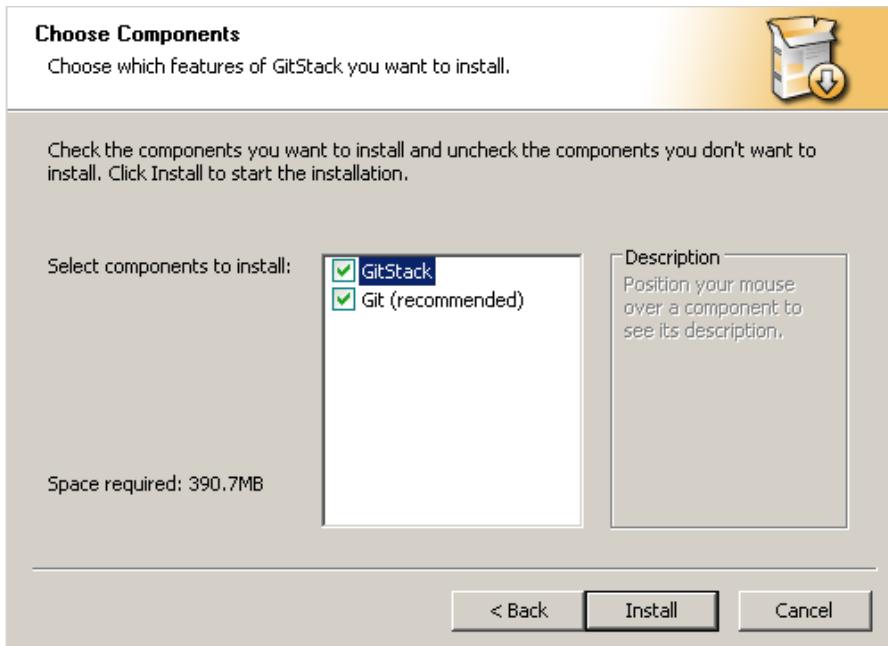


Ilustración 215 Asistente de instalación de gitstack: programas a instalar

Es necesario ingresar a la url <http://localhost/gitstack> al aplicativo usando las credenciales de administrador y proceder a crear un usuario que tenga permisos de lectura y escritura para el repositorio.

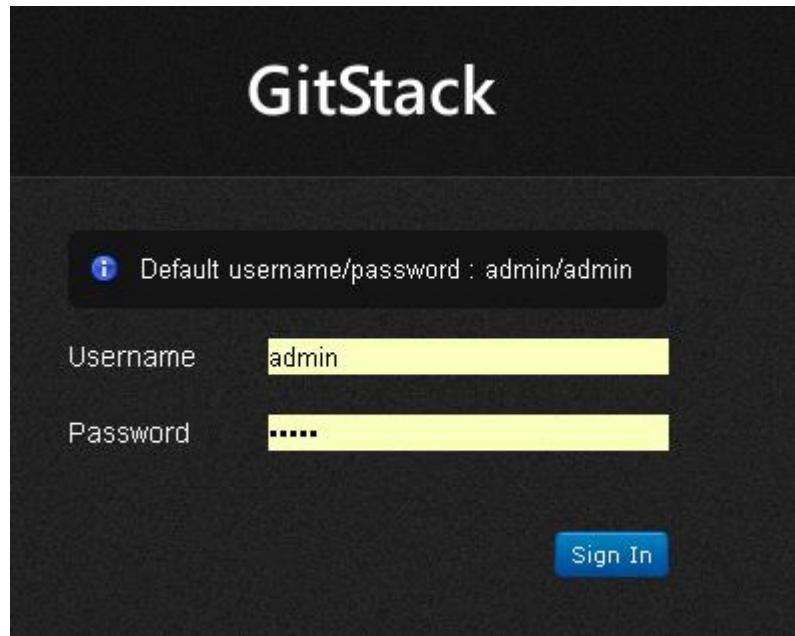


Ilustración 216 Ventana de inicio de sesión de gitstack

Después de autenticarse en el sistema se debe de ingresar a la interfaz administra y se creará un nuevo usuario con el que se podrá utilizar para crear y acceder a los repositorio que se vayan a utilizar.

The screenshot shows the GitStack application interface. On the left is a dark sidebar with a white header 'GitStack' and a 'Logout' link. Below the header are four menu items: 'Repositories' (blue background), 'Users & Groups' (white background), 'Users' (blue background, currently selected), and 'Groups' (white background). To the right of the sidebar is a light gray main area. At the top of the main area, a header says 'Users' with the sub-instruction 'Create and manage your users'. Below this is a table titled 'Users' with two columns: 'Username' and 'Action'. A single row is listed: 'juanwalker' with edit and delete icons. Below the table is a section titled 'Create user' containing fields for 'Username' (with 'admin' typed) and 'Password' (with '\*\*\*\*' typed). A blue 'Create' button is at the bottom of this section.

**Ilustración 217 Creación de usuarios en gitstack**

## Guía para instalación de RabbitMQ

El servidor RabbitMQ ofrece la comunicación por mensajes entre aplicaciones distribuidas, está basado en el lenguaje de programación y es necesario instalar la máquina virtual que permita su ejecución. Para ello se debe de ingresar al sitio de descargas de erlang <https://www.erlang.org/downloads> y bajar erlang para 64 bits, y posteriormente ejecutarlo y seguir el asistente paso a paso.

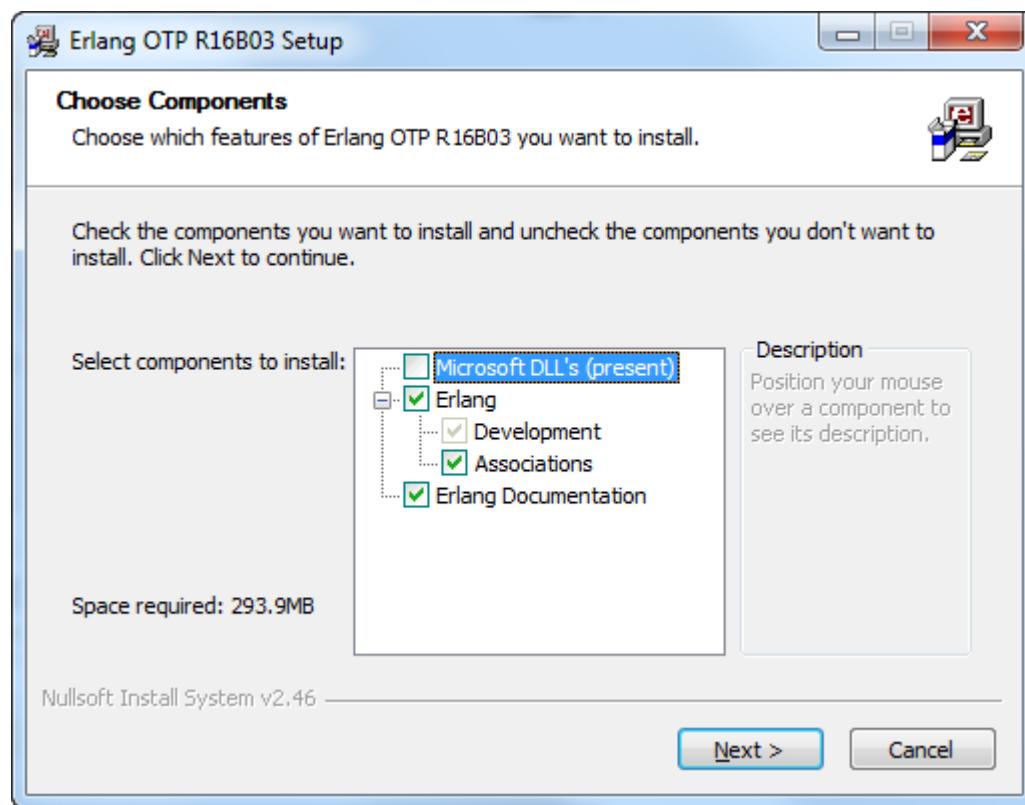


Ilustración 218 Asistente de instalación erlang: Componentes a instalar

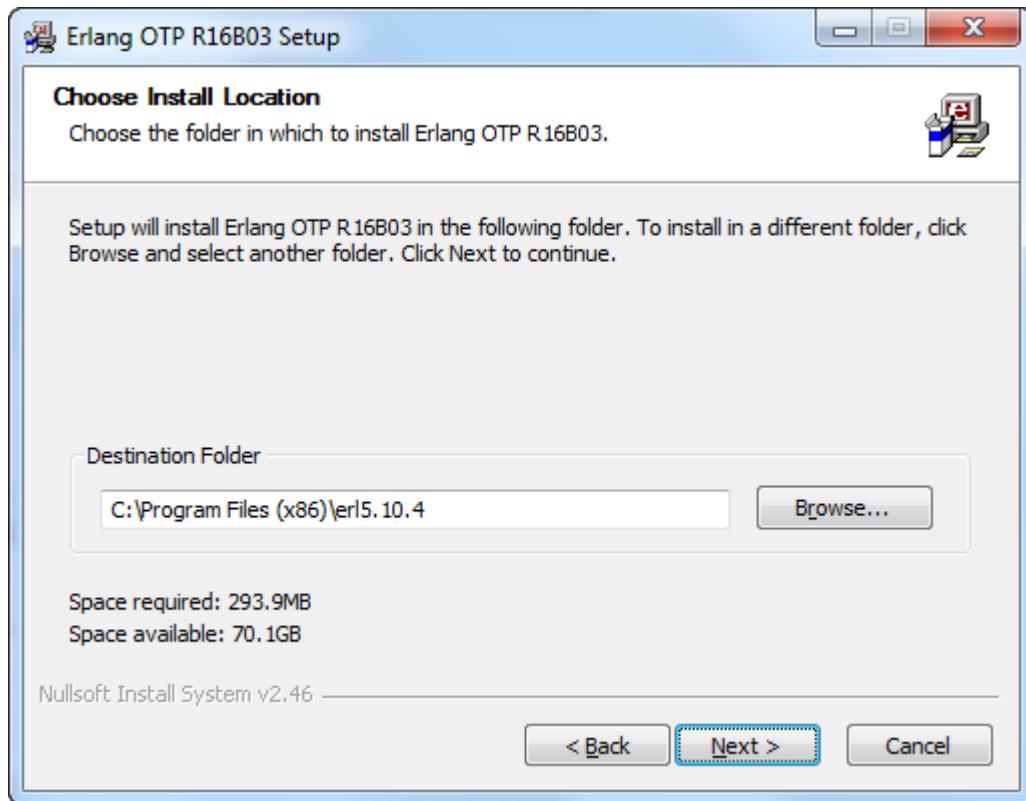


Ilustración 219 Asistente de instalación erlang: escojencia runta instalación

También es necesario definir la variable ERLANG\_HOME la cual tendrá como valor la ruta donde fue instalado erlang, por ejemplo ERLANG\_HOME= c:\Program Files (x86)\erl5.10.4\

Posteriormente es necesario descargar el instalador de RabbitMQ para Windows desde el sitio web <https://www.rabbitmq.com/download.html>, luego ejecutarlo y seguir el asistente paso por paso

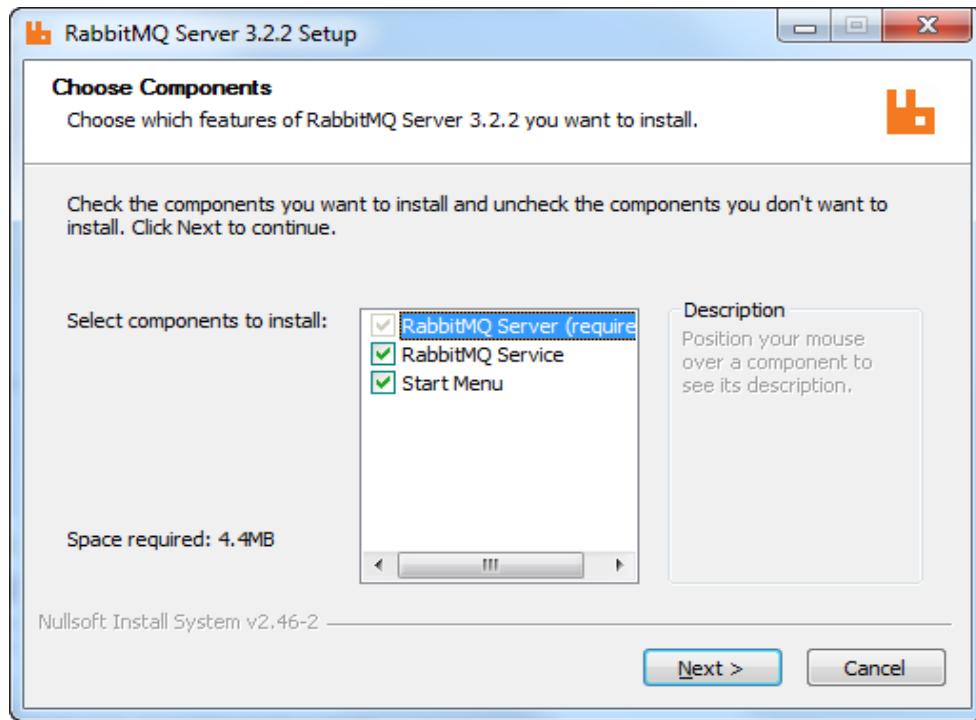


Ilustración 220 Asistente de instalación rabbitMQ: componentes a instalar

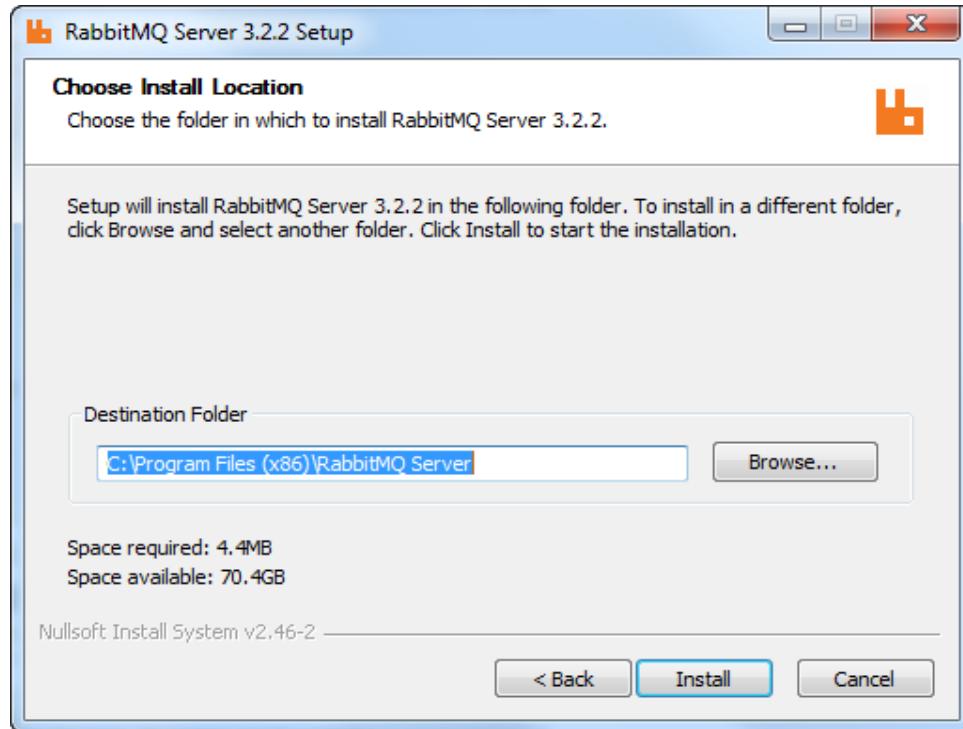


Ilustración 221 Asistente de instalación rabbitMQ: ruta de instalación

Se debe de abrir una consola y en el directorio donde se instaló rabbitMQ y dentro de la subcarpeta se deben de ejecutar los siguientes comandos:

```
rabbitmq-plugins.bat enable rabbitmq_management  
rabbitmq-service.bat stop  
  
rabbitmq-service.bat install  
rabbitmq-service.bat start
```

**Ilustración 222 Comandos necesarios para configurar rabbitMQ**

Finalmente verificar que el servidor rabbitMQ esté arriba entrando a la url <http://localhost:15672/> donde se puede encontrar una interfaz de inicio de sesión a la cual se debe de ingresar como *Username* guest y *Password* guest.



**Ilustración 223 Ventana de inicio de sesión de rabbitMQ**

Pudiendo acceder a la pantalla administrativa de RabbitMQ

The screenshot shows the RabbitMQ Management Interface. At the top, it displays the cluster information: Cluster: rabbit@The-Colombia-PC (dansen) RabbitMQ 3.6.6, Erlang 19.2. The navigation bar includes Overview, Connections, Channels, Exchanges, Queues, and Admin, with Overview selected. The main area is titled 'Overview' and contains two sections: 'Totals' and 'Node'. The 'Totals' section provides summary statistics: Queued messages (chart: last minute), Currently idle, Message rates (chart: last minute), Currently idle, Global counts (?), and specific counts for Connections: 0, Channels: 0, Exchanges: 12, Queues: 0, and Consumers: 0. The 'Node' section shows details for the node rabbit@The-Colombia-PC, including file descriptors (0 available, 7200 max), socket descriptors (0 available, 1048576 max), Erlang processes (250), Memory (50MB), Disk space (30GB), Rates mode (basic), and Info (Disc 1, Stats). Below these are sections for 'Paths', 'Ports and contexts', and 'Import / export definitions'.

Ilustración 224 Panel de control de RabbitMQ

## Guía de instalación de Docker

Docker es un sistema que permite manejar contenedores que no son más que máquinas virtuales simplificadas desde donde se pueden correr aplicaciones con el propósito de simular un ambiente distribuido para los desarrollos.

Para instalar docker en windows es recomendable instalar docker-toolbox el cual se puede descargar desde la siguiente url <https://www.docker.com/products/docker-toolbox> al descargarlo su instalación es bastante sencilla tan solo es necesario hacer click en el botón siguiente que aparece en el asistente de instalación

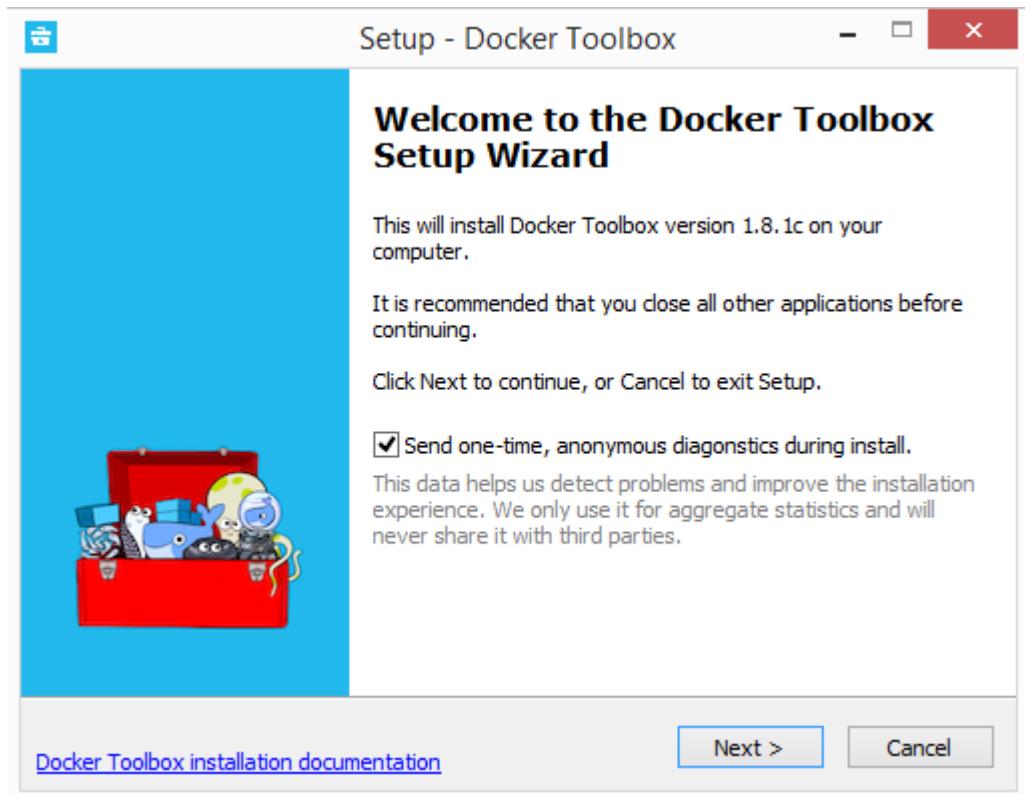


Ilustración 225 Asistente de instalación de docker toolbox

Al finalizar la instalación es necesario ejecutar el ícono en el escritorio llamado Docker Quickstart Terminal que se encargará de preparar una nueva terminal docker con toda la

A screenshot of a terminal window titled 'MINGW64:/c/Users/The Colombia'. The window shows a small ASCII art logo of a person sitting at a desk. Below it, the text reads: 'docker is configured to use the default machine with IP 192.168.99.100 For help getting started, check out the docs at https://docs.docker.com Start interactive shell'. The prompt shows 'The Colombia@The-Colombia-PC MINGW64 ~ \$'. The window has standard Windows-style borders and a scroll bar on the right.

Ilustración 226 Consola de docker usando docker toolbox

configuración necesaria para la ejecución de los comandos tipo unix y todos los propios de docker.

Es recomendable ofrecer una buena cantidad de memoria RAM del PC a docker de tal manera que la ejecución de los contenedores no se vea limitada, para ello se debe de ingresar por el ícono Oracle VM VirtualBox disponible en el escritorio y desde el Oracle VM VirtualBox Administrador y apagar la máquina default

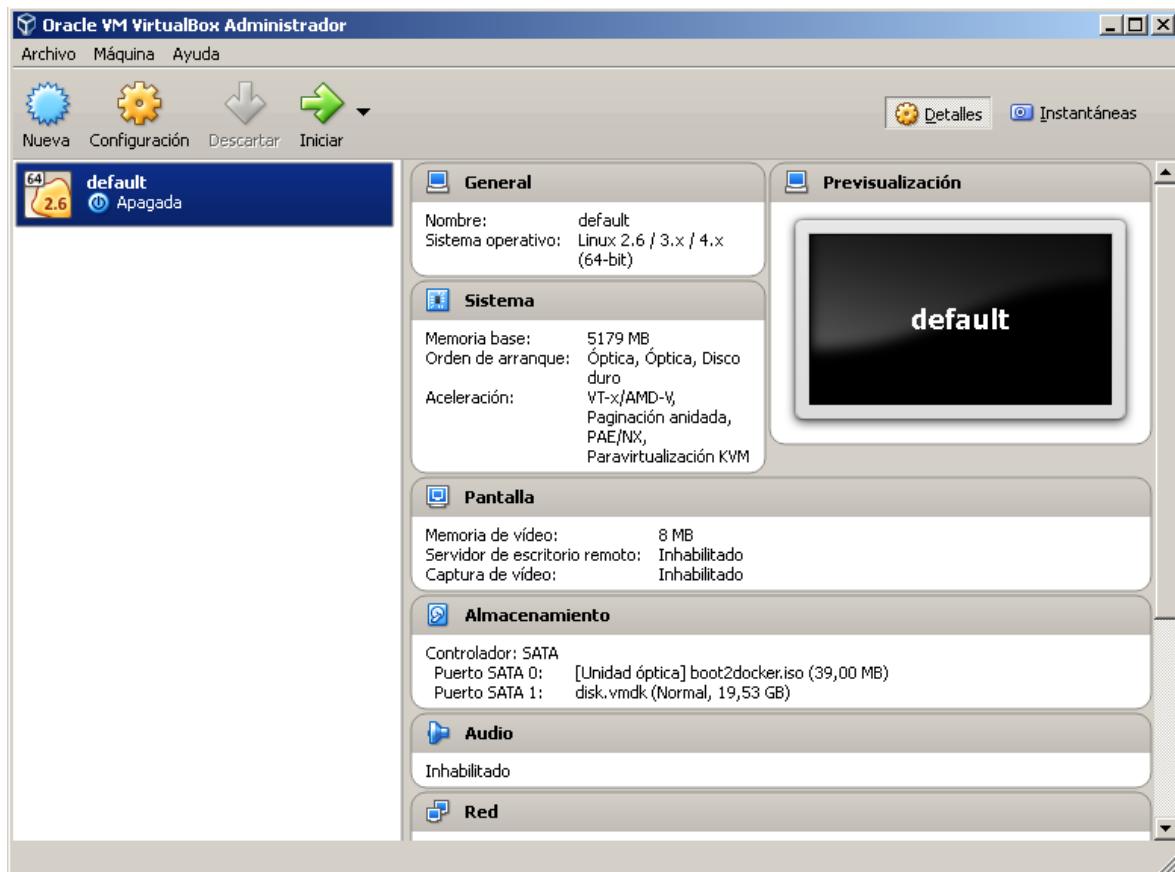


Ilustración 227 Panel de control de Oracle virtual box para docker

Y posteriormente hacer click derecho en la máquina default y seleccionar configuración y luego sistema donde se puede ver la cantidad de memoria RAM asignada, es sugerible que se ofrezca mínimo la mitad de memoria física disponible para la ejecución de los contenedores.

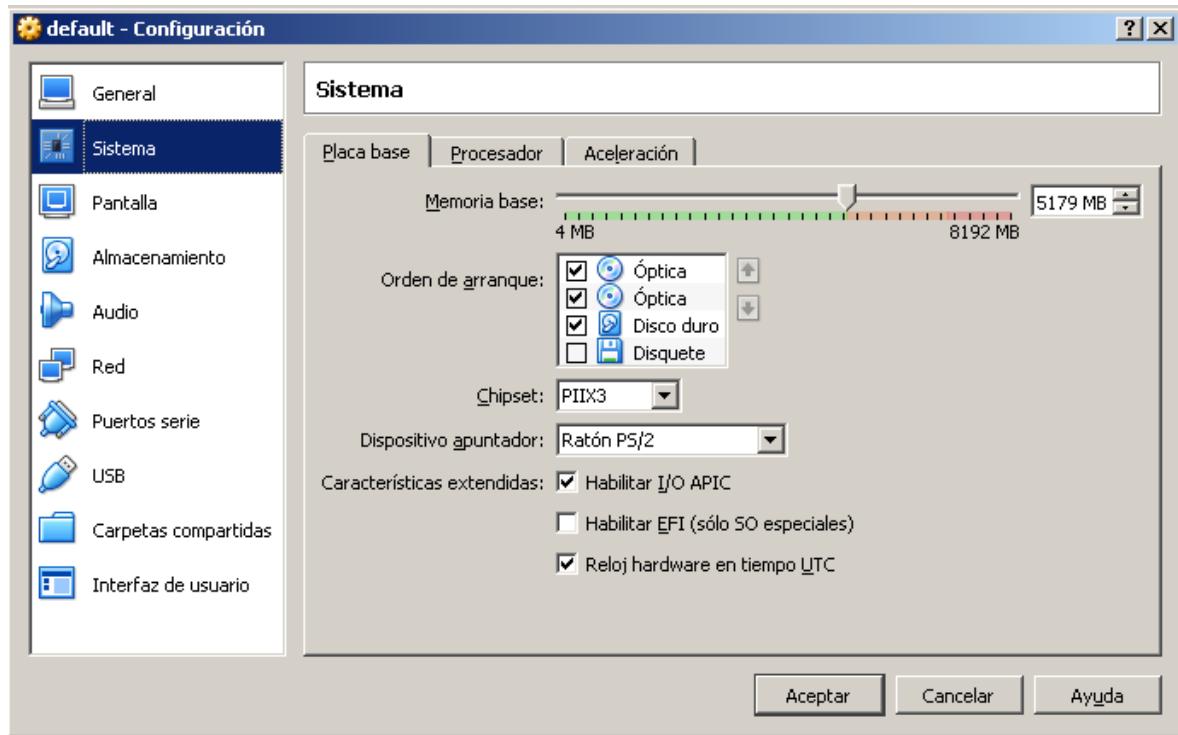


Ilustración 228 Ajuste de la memoria ram del Oracle Virtualbox para docker

Hasta este punto se ha realizado la instalación y configuración básica de docker para la ejecución de la práctica de microservicios.

## Guía de instalación de Jenkins

Para instalar jenkins es necesario descargar el archivo .war de la página oficial del proyecto <https://jenkins.io/>, posteriormente se creará una carpeta en el workspace llamada D:\microservicios\jenkins en donde se copiará el archivo .war y se abrirá una consola ubicada en la misma ruta donde se ejecutará el siguiente comando:

```
java -jar jenkins.war
```

Ilustración 229 Comando para la ejecución de jenkins desde línea de comandos

Es necesario ingresar a la página web principal de jenkins en la url <http://localhost:8080> y se realizará la configuración inicial, se debe de revisar el archivo initialAdminPassword donde se encuentra la cadena para desbloquear el ingreso de Jenkins.

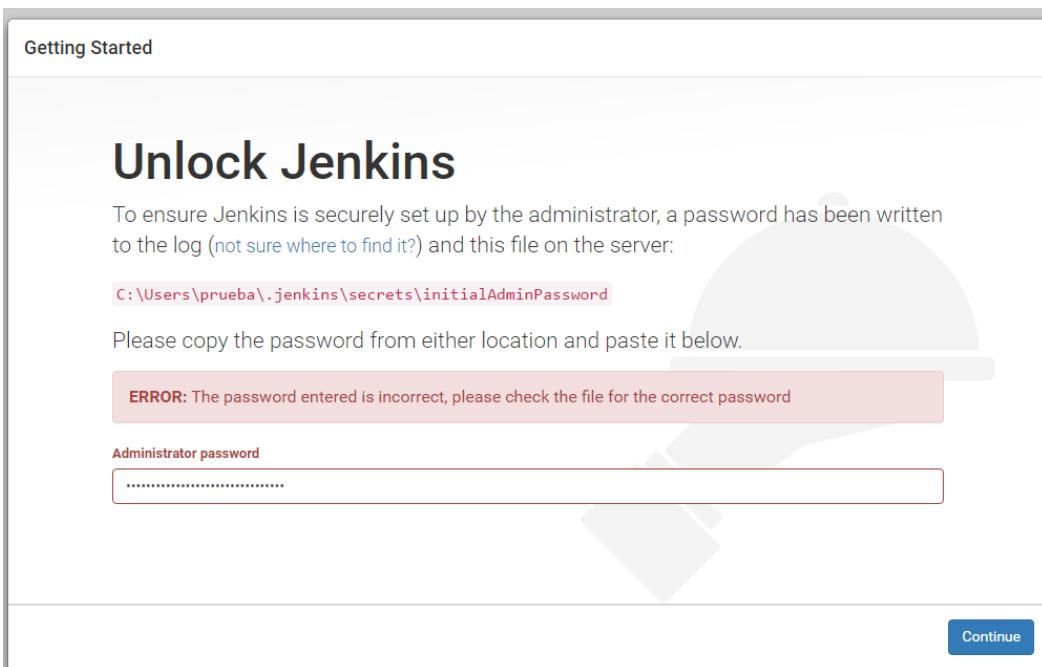
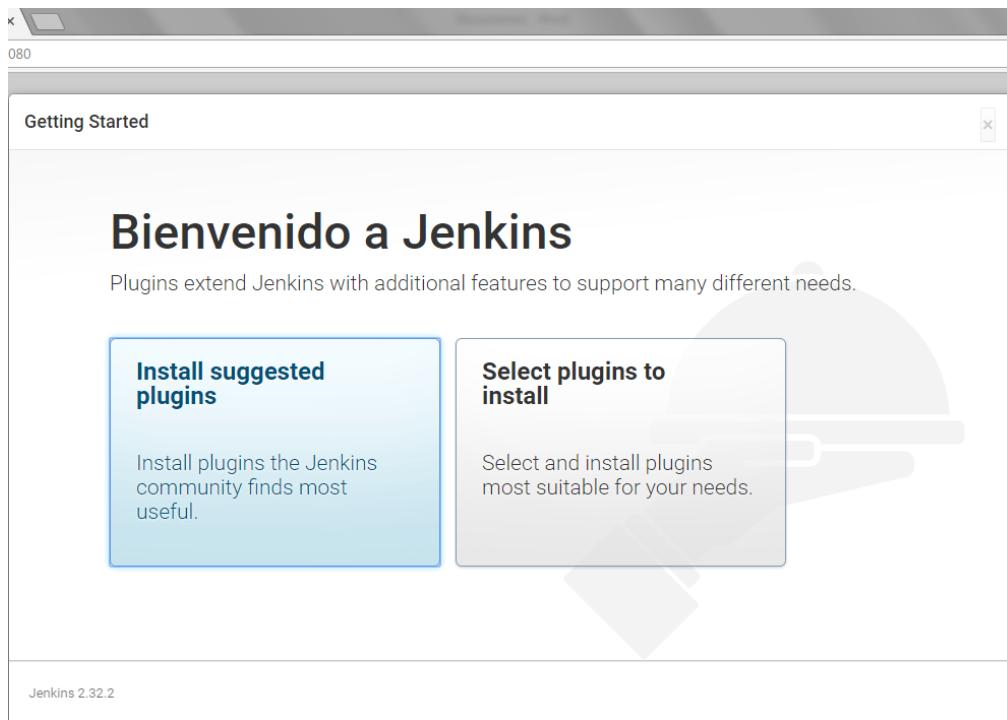
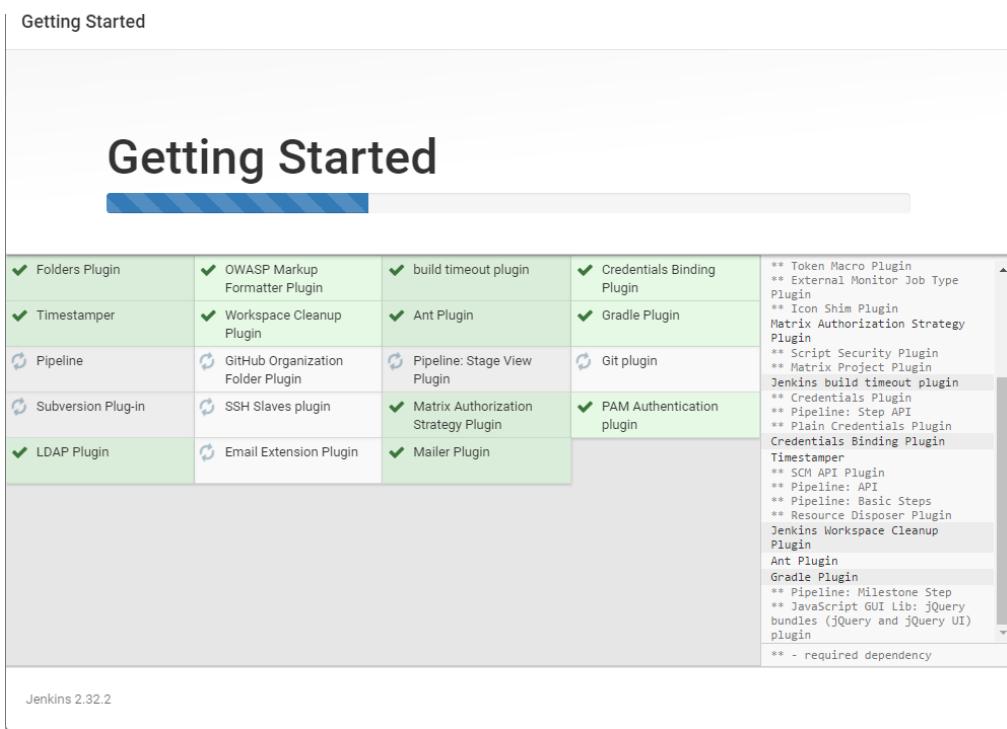


Ilustración 230 Pantalla inicial para configurar jenkins por primera vez

Posteriormente será necesario instalar los plugins de jenkins sugeridos para empezar su configuración y jenkins procederá a instalarlos automáticamente



**Ilustración 231 Escojencia de plugins para instalar en jenkins**



**Ilustración 232 Instalación de plugins en jenkins**

Jenkins también solicitará la creación del primer usuario administrador que utilizará para el uso de la interfaz administrativa.

Getting Started

## Create First Admin User

Usuario:

Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

Jenkins 2.32.2 Continue as admin **Save and Finish**

**Ilustración 233 Creación de primer usuario de jenkins**

Finalmente se podrá ingresar a la pantalla inicial de Jenkins.

S	W	Nombre	Último Éxito	Último Fallo	Última Duración
●	cloud	restful-blog-author-build	1 día 23 Hor - #40	8 días 18 Hor - #45	57 Seg
●	sun	restful-blog-author-promote	1 día 10 Hor - #41	1 día 10 Hor - #41	6,3 Seg
●	cloud	restful-blog-categories-build	8 días 10 Hor - #42	8 días 19 Hor - #41	20 Seg
●	sun	restful-blog-categories-promote	N/D	N/D	N/D
●	rain	restful-blog-configuration-build	8 días 10 Hor - #43	8 días 19 Hor - #42	15 Seg
●	sun	restful-blog-configuration-promote	N/D	N/D	N/D
●	sun	restful-blog-credentials-promote	N/D	N/D	N/D
●	sun	restful-blog-eureka-build	8 días 0 Hor - #44	9 días 7 Hor - #42	30 Seg
●	cloud	restful-blog-eureka-promote	8 días 0 Hor - #45	8 días 0 Hor - #47	6,1 Seg
●	cloud	restful-blog-log-build	7 días 23 Hor - #46	8 días 19 Hor - #42	15 Seg
●	cloud	restful-blog-log-credentials-build	8 días 10 Hor - #47	8 días 18 Hor - #42	27 Seg
●	cloud	restful-blog-log-promote	7 días 22 Hor - #48	7 días 22 Hor - #44	5,4 Seg
●	cloud	restful-blog-monitor-build	8 días 18 Hor - #49	8 días 19 Hor - #41	1 Min 50 Seg

**Ilustración 234 Pantalla principal de jenkins**

Jenkins Plugin interactúa con dos aplicaciones externas: Nexus y Gitstack por medio del uso de credenciales por ello será necesario entonces crear un par de nuevas llaves de tal

manera que pueda autenticarse en ambos servicios, se seguirá la siguiente ruta: Credentials, system y finalmente global credentials donde se seleccionará add credentials . Se debe de comenzar por crear las llaves para el servidor Nexus ingresando los valores **Username** como admin, **Password** como admin123 y el **ID** y **Description** tendrá como valor admin-nexus y será necesario presionar el botón guardar.

The screenshot shows the Jenkins Global Credentials configuration interface. It includes the following fields:

- Scope:** Global (Jenkins, nodes, items, all child items, etc)
- Username:** admin
- Password:** .....
- ID:** admin-nexus
- Description:** admin nexus

A blue "Save" button is located at the bottom left of the form.

**Ilustración 235 Creación de credenciales de nexus en jenkins**

Para la interacción con gitstack se debe de repetir el mismo procedimiento pero esta vez con los valores que se hallaron en el servidor gitstack, para el ejemplo **Username** será juanwalker, **Password** como juan123 y el **ID** y **Description** tendrá como valor gitStack\_local y se debe de presionar el botón guardar.

The screenshot shows the Jenkins Global Credentials configuration interface. It includes the following fields:

- Scope:** Global (Jenkins, nodes, items, all child items, etc)
- Username:** juanwalker
- Password:** .....
- ID:** gitStack\_local
- Description:** gitStack\_local

A blue "Save" button is located at the bottom left of the form.

**Ilustración 236 Creación de credenciales de git para jenkins**

Se requiere la instalación de dos plugins más que permitan injectar variables a las tareas de jenkins y otro más necesario para copiar los artefactos generados para ello desde la página

principal se debe de ingresar por la opción Administrar Jenkins, luego administrar plugins y de ahí en la pestaña Todos los plugin seleccionará Enviroment Injector Plugin y Nexus Jenkins Plugin y finalmente se hará click en el botón Instalar Sin Reiniciar.



**Ilustración 237 Instalación de plugin inyector de propiedades y nexus desde el administrador de jenkins**

El plugin de nexus jenkins es necesario parametrizarlo, al igual que la exposición de su configuración por variables globales accesibles por script para su correcto uso y funcionamiento ,para ello se debe de ingresar por a la opción Administrar Jenkins, configurar el sistema y modificar la sección propiedades globales escribiendo los siguientes valores teniendo en cuenta que el servidor nexus se instalará en la máquina local y el usuario de administración será admin con contraseña admin123 1 igual que la url <http://localhost:8081/nexus>

The screenshot shows the Jenkins Global Properties configuration page under the 'Variables de entorno' section. It lists three global variables:

- NEXUS\_PASSWORD**: Value: admin123. Action button: Borrar (Delete).
- NEXUS\_URL**: Value: <http://localhost:8081/nexus>. Action button: Borrar (Delete).
- NEXUS\_USER**: Value: admin. Action button: Borrar (Delete).

**Ilustración 238 Parametrización de variables globales de nexus en jenkins**

Se configurará el plugin en la sección Sonatype Nexus donde será necesario definir el id del servidor nexus, la url de nexus y asociar las credenciales previamente creadas. Antes de guardar la configuración es necesario probar la conexión.

The screenshot shows the Jenkins configuration interface for the Sonatype Nexus plugin. On the left, there's a sidebar with sections for 'Sonatype Nexus' and 'Nexus Repository Manager Servers'. The main area is titled 'Nexus Repository Manager 2.x Server' and contains the following fields:

- Display Name: Local Nexus
- Server ID: localNexus
- Server URL: http://localhost:8081/nexus/
- Credentials: admin/\*\*\*\*\* (admin nexus) (with an 'Add' button next to it)

Below these fields are two buttons: 'Test connection' (grey) and 'Borrar' (red). At the bottom, there are two dropdown menus: 'Add Nexus Repository Manager Server' and 'Add Nexus IQ Server'.

**Ilustración 239 Configuración de plugin nexus jenkins**

Los anteriores pasos servirán para hacer uso de jennkis en el caso de ejemplo donde se implementa la metodología.

## Guía de instalación de Nexus

Nexus es una aplicación que permitirá manejar repositorios para almacenar los artefactos producidos por la entrega continua. Para comenzar será necesario descargar el instalador desde el sitio web <https://www.sonatype.com/download-oss-sonatype> desde donde se podrá descargar el repository manager versión 2 para windows. Se creará el directorio nexus dentro del workspace y se debe descomprimir el archivo comprimido, abriendo una consola de windows y apuntando a la ruta donde nexus\nexus-2.X.X-XX\bin\jsw\windows-x86-64 se ejecutará el comando install-nexus.bat el cual instalará el wrapper necesario para su funcionamiento, posteriormente desde la ruta D:\microservicios\nexus\nexus-2.X.X-XX\bin\ será necesario ejecutar el comando nexus.bat el cual iniciará el servidor nexus donde se podrá ingresar por medio de la url <http://localhost:8081/nexus/> donde se ingresarán las credenciales que vienen por defecto Username **admin** y Password **admin123**

Pudiendo ingresar a la pantalla principal desde donde se podrá ver los repositorio activos

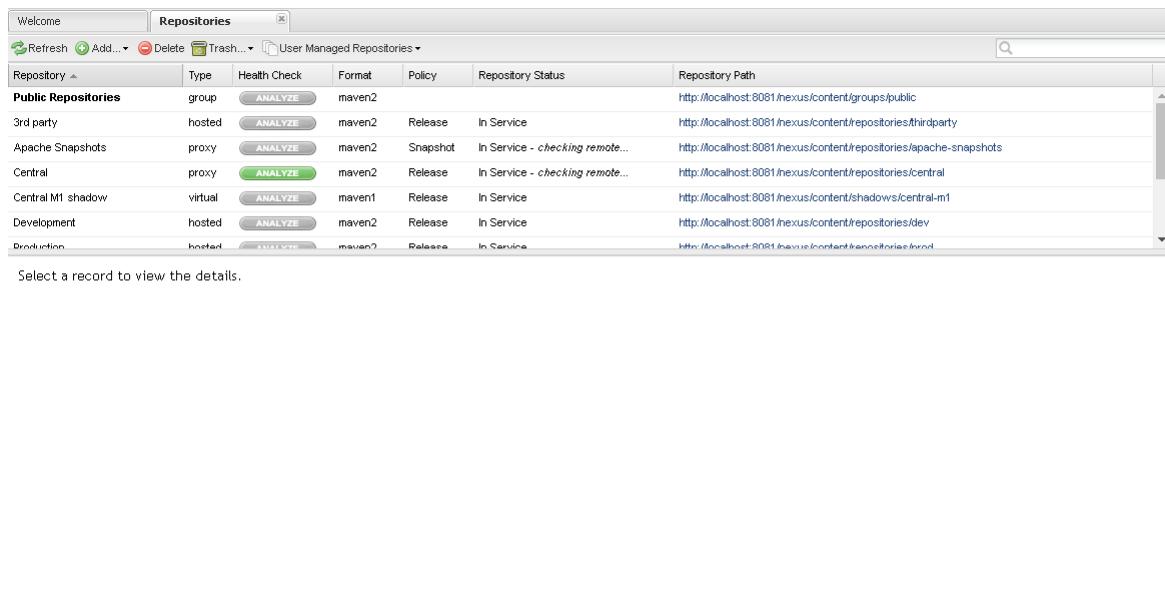
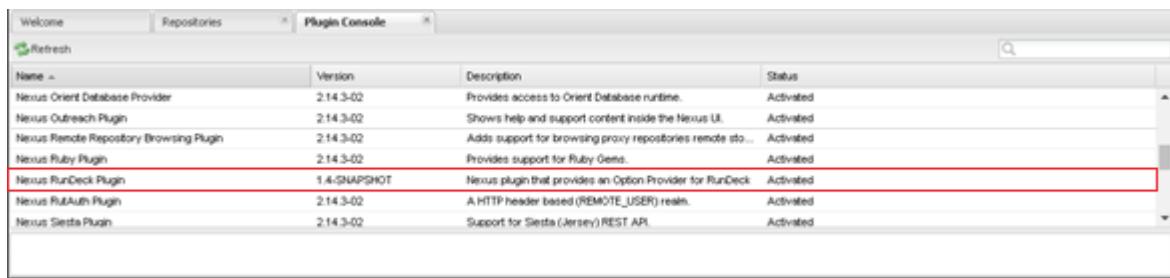


Ilustración 240 Explorador de repositorio de nexus

También será necesario configurar la integración de nexus con rundeck para ello se debe de descargar el código fuente del proyecto <https://github.com/rundeck/nexus-rundeck-plugin> el cual se necesitará descargar y compilar por línea de comandos por medio del comando mvn clean install, el cual generará un archivo llamado nexus-rundeck-plugin-1.4-SNAPSHOT-bundle.zip que se tiene que copiar dentro de la carpeta D:\microservicios\nexus\sonatype-work\nexus\plugin-repository\ y descomprimirla allí. Bastará con reiniciar el servidor para que el plugin sea utilizado por el servidor, para verificar que esté funcionando correctamente se hace desde la interfaz gráfica desde *administration* y luego *plugin console*.



The screenshot shows the Nexus Plugin Console interface. At the top, there are tabs for 'Welcome', 'Repositories', and 'Plugin Console'. The 'Plugin Console' tab is active. Below the tabs is a search bar. The main area is a table with columns: 'Name', 'Version', 'Description', and 'Status'. There are seven rows of data:

Name	Version	Description	Status
Nexus Orient Database Provider	2.14.3-02	Provides access to Orient Database runtime.	Activated
Nexus Outreach Plugin	2.14.3-02	Show help and support content inside the Nexus UI.	Activated
Nexus Remote Repository Browsing Plugin	2.14.3-02	Adds support for browsing proxy repositories remote sto...	Activated
Nexus Ruby Plugin	2.14.3-02	Provides support for Ruby Gems.	Activated
Nexus RunDeck Plugin	1.4-SNAPSHOT	Nexus plugin that provides an Option Provider for RunDeck.	Activated
Nexus RunAuth Plugin	2.14.3-02	A HTTP header based (REMOTE_USER) realm.	Activated
Nexus Siesta Plugin	2.14.3-02	Support for Siesta (Jersey) REST API.	Activated

Ilustración 241 Verificación de plugins instalados en nexus

Hasta aquí se ha configurado Nexus Repository Server para que funcione correctamente para realizar la práctica de la entrega continua.

## Guía de instalación de Rundeck

Rundeck es una aplicación web que permite parametrizar las tareas y ejecutarlas de acuerdo a los valores ingresados por los usuarios. Esta aplicación apoyará en las labores de despliegue de las entregas continuas.

Para comenzar es necesario descargar el instalador de rundeck <http://rundeck.org/downloads.html> y se copiará dentro de una nueva carpeta dentro del workspace la cual se debe de llamar rundeck, su ejecución es bastante sencilla solo se debe de abrir una consola en la ruta D:\microservicios\rundeck y ejecutar el comando:

Luego de ejecutar este comando se podrá ingresar desde el navegador a la ruta

```
java -jar rundeck-launcher-2.X.X.jar
```

Ilustración 242 Comando para ejecutar rundeck desde línea de comandos

<http://localhost:4440> y poder visualizar la pantalla de inicio de sesión de la aplicación, donde se introducirán las credenciales por defecto **Username** admin y **Password** admin

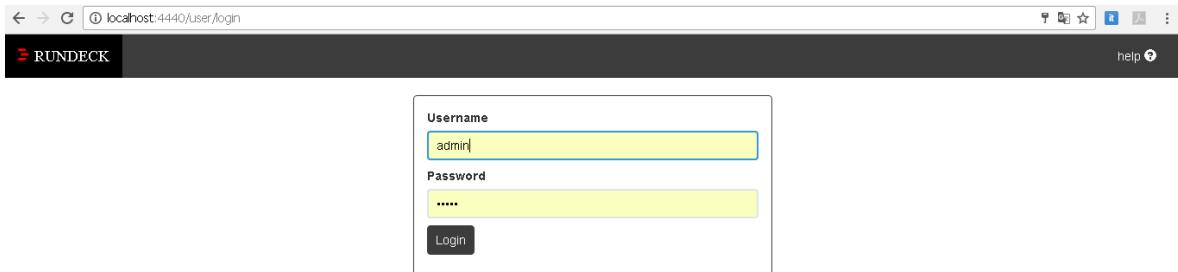
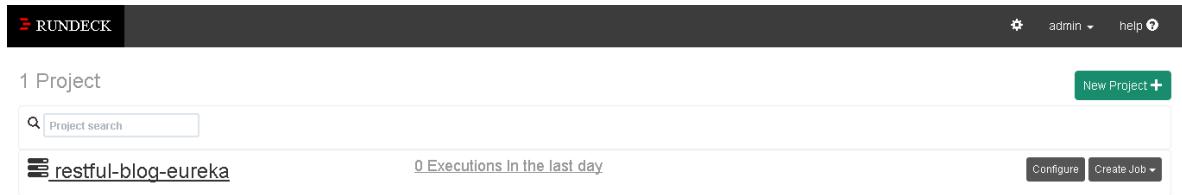


Ilustración 243 Ventana de inicio de sesión de rundeck

Y poder ver la pantalla de inicio de rundeck



**Ilustración 244 Ventana principal de rundeck**

Hasta aquí se ha configurado rundeck de la manera más simple para ser utilizado en la práctica de implementación de la metodología de microservicios

## CONCLUSIONES

Al validar la metodología en el caso práctico se logró evidenciar que lo planteado por la hipótesis se cumple a totalidad tal como se explica a continuación:

- Se implementó una adecuada semejanza entre el negocio y los microservicios ya que cada contexto de negocio tenía una equivalencia de microservicio.
- Mejoró la legibilidad del código fuente puesto que inicialmente todo el código se encontraba en un solo proyecto y se fragmentó en diferentes varios proyectos de código al igual que se implementaron varias base de código fuente.
- Se realizó una reducción casi total del acoplamiento en los componentes creando una base de datos por microservicio, separando su ejecución en servidores web separados y haciendo uso de comunicación por mensajería por medio de un *bróker*.
- Se implementó exitosamente el procedimiento de integración y entrega continua por medio de herramientas de software que permitían detectar las actualizaciones en las bases de código fuente, ejecutar la compilación, ejecución de pruebas y despliegue automáticamente.
- Cada microservicio implementó lógica propia que permitía dar un manejo a los fallos de ejecución del servicio previniendo lanzar errores no manejados. También cada uno de ellos publicó estadísticas de uso de red que permitieron una adecuado monitoreo.

Por lo anterior se podrá concluir que la arquitectura microservicios plantea un cambio de paradigma sobre como se están actualizando los sistemas legados en la actualidad ya que al proponer una transformación hacia una arquitectura distribuida la aplicación reduce el acoplamiento interno entre componentes simplificando la escalabilidad de los mismos y

facilitando la lectura de la base de código fuente que se reorganiza por contextos lo que disminuye los tiempos de reparación de errores o de actualizaciones del sistema .

Sin embargo todas estas bondades necesitan de ciertas condiciones para que puedan ser aprovechadas, entre ellas están una considerable madurez de la aplicación que garantice que sus funcionalidades se encuentran validadas por los usuarios, un dominio de la propiedad de la propiedad intelectual del código fuente que necesita ser transformado y un equipo de desarrolladores cualificados que tengan la experiencia en aplicaciones distribuidas.

Los microservicios son un cambio de paradigma que muchas empresas de renombre internacional han estado eligiendo como es el caso de facebook, netflix, uber entre otras, y que de ser construidos con pertinencia y en el momento preciso pueden aliviar muchos dolores de cabeza al departamento de sistemas cuando un negocio necesita atender una mayor demanda de clientes o realizar actualizaciones de su plataforma tecnológica.

## **Trabajos futuros**

- Para el caso de prueba no se profundizó en las pruebas de los microservicios, solo se hicieron pruebas funcionales, se podría desarrollar una metodología donde el tema central sea la realización efectiva de pruebas punto a punto o pre-producción en medio de la coreografía entre los microservicios del sistema.
- Realizar capacitaciones y promocionar el uso de microservicios realizando talleres que provoque a los programadores a aprender más sobre la programación distribuida y así cualificar el talento local.
- Desarrollar casos de ejemplo haciendo uso de otras tecnologías como .net php, ruby u otros lenguajes de programación.

## BIBLIOGRAFÍA

- [1] Fowler Martin, 2014, *BoundedContext* [Imagen], disponible en <http://martinfowler.com/bliki/BoundedContext.html> [Recuperado 27 Junio 2016]
- [2] Lewis James & Fowler Martin, 2014, *Microservices*. Imagen], disponible en <http://martinfowler.com/articles/microservices.html> [Recuperado Recuperado 27 Junio 2016]
- [3] Fowler Martin, 2015, *MicroservicePremium* [Imagen] disponible en <http://martinfowler.com/bliki/MicroservicePremium.html> [Recuperado 30 Junio 2016]
- [4] Clemson Toby, 2014, *Testing Strategies in a Microservices Architecture* [Imagen], disponible <http://martinfowler.com/articles/microservice-testing/> [Recuperado 10 Julio 2016]
- [5] Newman Sam. *Building microservices*. Primera edición. Sebastopol, California. O'Reilly Media Inc, 2015. ISBN 978-1-4919-5035-7
- [6] Evans Eric. *Domain-driven design*. Primera edición, Boston, Massachusetts. Addison-Wesley, 2004. ISBN 0-321-12521-5
- [7] Gitlevich,V. & Evans, E. (2007, March). What is Domain Driven Design? Retrieved from [http://dddcommunity.flywheelsites.com/learning-ddd/what\\_is\\_ddd/](http://dddcommunity.flywheelsites.com/learning-ddd/what_is_ddd/)
- [8] Richardson Chris, 2016. *Refactoring Monolith into Microservices* .[Imagen] disponible en <https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/> [Recuperado 12 Septiembre 2016]

[9] Larman Craig, *Agile and Iterative Development: A Manager's Guide* Primera Edición, Boston, Massachusetts,Pearson Education Inc, 2004,ISBN 0-13-1111155-8

[10] Lerman Julie, 2014, *Entity Framework Model Partitioning in Domain-Driven Design Bounded Contexts*, [Video], disponible en

[https://www.youtube.com/watch?v=rGA\\_FNew-6g#t=23m39s](https://www.youtube.com/watch?v=rGA_FNew-6g#t=23m39s) [Recuperado 2 Octubre 2016]

[11] Bryant Daniel, 2015, *Microservice Maturity Model Proposal* [documento], disponible en <https://taidevcouk.files.wordpress.com/2015/02/microservice-maturity-model-proposal-daniel-bryant-danielbryantuk.pdf> [Recuperado 15 Octubre 2016]

[12] Brown Kyle 2016, *Refactoring to microservices, Part 1: What to consider when migrating from a monolith* [Imagen] disponible en <https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-1/> [Recuperado 20 Octubre 2016]

[13] Haddad Chris 2012, *Netflix Has a Eureka Moment* [Imagen], disponible en <https://www.techwell.com/techwell-insights/2012/09/netflix-has-eureka-moment> [Recuperado 27 Octubre 2016]

[14] Sastrasinh Paul 2016, *API Infrastructure at Knewton: What's in an Edge Service?* [Imagen], disponible en <https://tech.knewton.com/blog/2016/05/api-infrastructure-knewton-whats-edge-service/> [Recuperado 29 Octubre 2016]

[15] Fowler Martin, 2014, *CircuitBreaker*, [Imagen] disponible en <http://martinfowler.com/bliki/CircuitBreaker.html> [Recuperado 30 Octubre 2016]

[16] Stine Matt, 2015, *Build self-healing distributed systems with Spring Cloud*, [Imagen], disponible en <http://www.infoworld.com/article/2925047/application->

development/build-self-healing-distributed-systems-with-spring-cloud.html [Recuperado 10 Noviembre 2016]

[17] Fowler Martin 2006, *Continuous Integration*, disponible en <http://martinfowler.com/articles/continuousIntegration.html> [Recuperado 15 Noviembre 2016]

[18] Humble Jez, Farley David, 2011, *Continous Delivery*, Primera Edición, Pearson Education Inc, 2011, ISBN 978–0–321–60191–9

[19] Folwer Martin, 2013, *Testing Strategies in a Microservices Architecture* [Imagen], disponible <http://martinfowler.com/articles/microservice-testing/> [Recuperado 10 Febrero 2017]

[20] Wake Bill, 2003, *INVEST in Good Stories, and SMART Tasks* disponible <http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/> [Recuperado 15 Febrero 2017]

[21] Josuttis Nicolai M., *SOA in practice the art of a distributed system design*, Primera edición. Sebastopol, California. O'Reilly Media Inc, 2007. ISBN 978-0-596-52955-0

[22] Wolff Eberhard, *Microservices: Flexible Software Architecture*, Primera edición. Crawfordsville, Indiana. Addison-Wesley, 2016. ISBN 978-0-134-60241-7

[23] Miller Ron, 2016, *How AWS came to be* disponible en <https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/> [Recuperado 21 Marzo 2017]

[24] Mauro Tony, 2016, *Adopting Microservices at Netflix: Lessons for Architectural Design* disponible en <https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/> [Recuperado 21 Marzo 2017]

[25] RV Rajesh, 2016, *Spring microservices: Build scalable microservices with Spring, Docker and Mesos*, Primera Edición Birmingham, UK 978-1-78646-668-6, PACKT Publishing

[26] Krueger Ken, 2015, *Microservices with Spring Cloud: Lessons for Architectural Design* disponible en <https://www.udemy.com/microservices-with-spring-cloud/learn/v4/t/lecture/2953328?start=0> [Recuperado 21 Marzo 2017]

[27] Stenberg Jay, 2014, *Martin Fowler on Characteristics of Microservices* disponible en <https://www.infoq.com/news/2014/11/gotober-fowler-microservices> [Recuperado 21 Marzo 2017]

[28] Albaladejo Xavier, 2008, *Qué es SCRUM* disponible en <https://proyectosagiles.org/que-es-scrum/> [Recuperado 21 Marzo 2017]

[29] Silverthorne Valierie, 2015, *What is the most popular agile process?* <http://searchsoftwarequality.techtarget.com/photostory/4500251249/VersionOne-surveys-Agile-development-for-2015/5/How-popular-is-the-Scrum-process> [Recuperado 21 Marzo 2017]

[30] Sato Danilo, 2015, *CanaryRelease* <https://martinfowler.com/bliki/CanaryRelease.html> [Recuperado 21 Marzo 2017]

[31] Massacci F , Redwine S., Zannone N.,2009, *Engineering Secure Software and Systems*, Primera Edición Berlin,Germany 3-642-00198-X, Springer International

[32] Cohn Mike, 2004, *User Stories Applied*, Primera Edición Boston,MA 0-321-20568-5, Addison-Wesley

[33] Long Josh, 2015, *Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka* <https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka> [Recuperado 21 Marzo 2017]

[34] Fowler Martin, 2014, *CircuitBreaker* <https://martinfowler.com/bliki/CircuitBreaker.html> [Recuperado 21 Marzo 2017]

[35] Stine Matt, 2015, *Migrating to cloud-native application architectures* <https://www.oreilly.com/ideas/migrating-to-cloud-native-application-architectures/page/3/migration-cookbook> [Recuperado 21 Marzo 2017]

[36] Salerno Rafael, 2015, *Spring Cloud Netflix: Load Balancer with Ribbon/Feign* <https://dzone.com/articles/spring-cloud-netflix-load-balancer-with-ribbonfeig> [Recuperado 21 Marzo 2017]

[37] Sackman Matthew, 2010, RabbitMQ: Messaging in the Cloud , [http://www.rabbitmq.com/resources/RabbitMQ\\_MessagingInTheCloud\\_VMworld\\_2010\\_MS.pdf](http://www.rabbitmq.com/resources/RabbitMQ_MessagingInTheCloud_VMworld_2010_MS.pdf) [Recuperado 21 Marzo 2017]

[38] Celesti A.,Leitner P. , 2015, *Advances in Service-Oriented and Cloud Computing*, Primera Edición, Switzerland, 978-3-319-33312-0, Springer International

[39] Richardson Chris, 2016 Refactoring a Monolith into Microservices <https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/inthecloud> [Recuperado 21 Marzo 2017]

[40] Stenberg Jan, 2016 Using Domain-Driven Design When Creating Microservices <https://www.infoq.com/news/2016/02/ddd-microservices> [Recuperado 21 Marzo 2017]

[41] Belcham Donald, 2016 Microservices and isolation  
<http://www.westerndevs.com/Microservices-and-isolation/> [Recuperado 21 Marzo 2017]

[42] Cruz Cristian, Castro Gustavo, 2015, Metodología para la adquisición y gestión de requerimientos en el desarrollo de software para pequeñas y medianas empresas (pymes) del departamento de risaralda (tesis maestría), Universidad Tecnológica de Pereira, Pereira, Colombia

[42] Cruz Cristian, Castro Gustavo, 2015, Metodología para la adquisición y gestión de requerimientos en el desarrollo de software para pequeñas y medianas empresas (pymes) del departamento de risaralda (tesis maestría), Universidad Tecnológica de Pereira, Pereira, Colombia

[43] Lainez Fuentes José Rubén, Desarrollo de Software Ágil: Extreme Programming y Scrum, Primera edición, Createspace Independent Publishing Platform ,2014 ISBN 978-1502952226

[44] Richardson Chris, 2016, Refactoring a Monolith into Microservices,  
<https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/> [Recuperado 8 de Mayo 2017]

[45] Rosa Eduardo, Nguyen Duy V, Florez Luis, De Santis Sandro, Evolve the Monolith to Microservices with Java and Node, Primera edición, Publisher: IBM Redbooks, 2016, ISBN: 0783442119

[46] Tonse Sudhir, 2015, Scalable Microservices at Netflix. Challenges and Tools of the Trade, disponible en [https://www.infoq.com/presentations/netflix-ipc?utm\\_source=infoq&utm\\_medium=slideshare&utm\\_campaign=slidesharesf](https://www.infoq.com/presentations/netflix-ipc?utm_source=infoq&utm_medium=slideshare&utm_campaign=slidesharesf) [Recuperado 8 de Mayo 2017]

[47] Falé Flávia, 2017, SyntheticMonitoring, disponible en <https://martinfowler.com/bliki/SyntheticMonitoring.html> [Recuperado 8 de Mayo 2017]

[48] Bryant Daniel, 2015, Monitoring Microservices and Containers: A Challenge by Adrian Cockcroft, disponible en <https://www.infoq.com/news/2015/06/monitoring-microservices> [Recuperado 8 de Mayo 2017]

[49] Gajda Kim , 2016, The Circuit Breaker pattern, disponible en [https://www.ibm.com/devops/method/content/manage/practice\\_circuit\\_breaker\\_pattern/](https://www.ibm.com/devops/method/content/manage/practice_circuit_breaker_pattern/) [Recuperado 8 de Mayo 2017]

[50] Loukides Mike, What is DevOps?, Primera Edición, O'Reilly Media,,2012, ISBN 1-4493-3910-7