Negro, Gonzalo Nicolas

Migración de tecnologías y automatización de procesos para sistema IntegraBPM

2021

Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática

PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

Migración de tecnologías y automatización de procesos para sistema IntegraBPM

Gonzalo Nicolas Negro

Florencio Varela, diciembre de 2021

Estudiante
Gonzalo Nicolás Negro
DNI: 39.161.353
Nº de Legajo: 13.950
gonzabj16@gmail.com
Cantidad de materias aprobadas al comienzo de la PPS: 45

Práctica Profesional Supervisada (PPS) enmarcada en artículo 7°B de la Resolución (CS) 103/16.

ORGANIZACIÓN DONDE SE REALIZA LA PPS
Consultoría en Tecnología de Información S.A.
Maipú 942, C1006 ACN, Buenos Aires
011 5252-8100
Sector: Consultoría en Tecnología de Información

Tutor por la organización
Ing. Gisela Sullivan
gisela.sullivan@softtek.com

Docente supervisor por UNAJ
Ing. Jésica Guzmán
jguzman@unaj.edu.ar

Docente tutor por el Taller de Apoyo para la Producción de Textos Académicos (UNAJ)
Prof. Lavigna Lía
llavigna@unaj.edu.ar

Coordinador de la carrera de Ingeniería en Informática
Dr. Ing. Martín Morales
martin.morales@unaj.edu.ar

Resumen en español

La Práctica Profesional Supervisada (PPS) se realizará dentro de la empresa de tecnología Softtek para el cliente OSDE e involucra a la aplicación IntegraBPM que se encarga de la gestión de procesos. Una de las problemáticas que presenta es que se trata de una aplicación desarrollada sobre tecnologías obsoletas en comparación con las que se utilizan en el mercado tecnológico. Otro de los inconvenientes que tiene el software es la cantidad de errores en su funcionamiento, que provoca altos volúmenes de memoria y procesamiento, generando la inestabilidad del sistema.

El proyecto consiste en la migración de distintas tecnologías que utiliza la aplicación web, realizando un proceso completo de análisis sobre mejoras y compatibilidades entre las herramientas. El objetivo del proyecto es mejorar el rendimiento y performance de la aplicación, utilizando mejores técnicas de trabajo y mejores medidas de seguridad. También, se requiere aplicar al proyecto la metodología DevOps, para reducir los tiempos de desarrollo y aumentar la productividad. Las tareas del plan de migración, consisten en la creación de todos los ambientes y sus instancias, además, la configuración para la ejecución de todas las herramientas. Las tecnologías que se van a utilizar en la PPS son: JBoss EAP, Maven, Java, Spring,Hibernate, Apache Solr, etc.

Resumen en inglés

The Supervised Professional Practice (PPS) will be carried out within the technology company Softtek for the client OSDE and involves the IntegraBPM application that is responsible for process management. One of the problems it presents is that it is an application developed on obsolete technologies compared to those used in the technology market. Another drawback of the software is the number of errors in its operation, which causes high volumes of memory and processing, generating system instability.

The project consists of the migration of different technologies used by the web application, carrying out a complete process of analysis on improvements and compatibilities between the tools. The objective of the project is to improve the performance of the application, using better working techniques and better security measures. Also, it is required to apply the DevOps methodology to the project, in order to reduce development times and increase productivity. The tasks of the migration plan consist of the creation of all the environments and their instances, as well as the configuration for the execution of all the tools. The technologies to be used in the PPS are: JBoss EAP, Maven, Java, Spring, Hibernate, Apache Solr, etc.

Dedicatorias y agradecimientos:

Agradezco a mis tutoras involucradas en el proyecto PPS, a la profesora Lía Lavigna por el apoyo y enseñanza durante todo este proceso en la redacción de mi informe. También, a Gisela Sullivan y Jésica Guzmán por la disposición y ayuda durante todo el proceso.

Índice

Resumen en español 3
Resumen en inglés 4
1. Introducción 8
1.1 Objetivos 9
1.2 Herramientas y Tecnologías 10
1.2.1 Java 10
1.2.2 Maven 11
1.2.3 Spring 11
1.2.4 JBoss EAP 11
1.2.5 Apache Solr 12
1.2.6 GitLab 12
1.2.7 Trello 12
1.3 Tareas a ejecutar 13
1.3.1 Planeación 13
1.3.2 Diseño 13
1.3.3 Desarrollo 14
1.3.4 Pruebas 15
1.3.5 Informes 15
1.4 Cronograma de trabajo 15
2. Desarrollo 16
2.1 Planeación 16
2.1.1 Análisis y requerimientos 16
2.1.1 Compatibilidad de herramientas e investigación 17
2.1 Diseño de ambientes 22
2.1 Migración de JBoss 25
2.2 Migración de Java y Maven 31
2.3 Migración de Frameworks 34
2.4 Migración de Apache CXF 35
2.5 Circuito de eventos 39
2.6 Integración continua 48
2.7 Apache Solr 57
2.8 Pruebas y monitoreo 68
3. Conclusiones 69
3.1 Reflexión sobre la práctica profesional supervisada como espacio de formación 71

Índice de Figuras

Figura 1 Arquitectura Java..... 10
Figura 2. Cronograma de trabajo. 16
Figura 3 JBoss EAP 7.2 Versiones soportadas. 18
Figura 4 Arquitectura Solr. 20
Figura 5 Handler de tipo Size. 29
Figura 6 Definición de Logger. 29
Figura 7 Properties dentro del Standalone 30
Figura 8 Ordenamiento de observaciones 33
Figura 9 Ejemplo de request del servicio Trámite. 36
Figura 10 Response con error de enconding. 36
Figura 11 Interceptor para arreglar el enconding 38
Figura 12 Archivo cxf.xml con la declaración de los Interceptors. 39
Figura 13 Circuito de eventos de Integra. 40
Figura 14 Configuración de conexión al MQ 42
Figura 15 Configuración para publicar eventos 42
Figura 16 Beans para la conexión al MQ y producción de eventos 43
Figura 17 Método para enviar eventos 45
Figura 18 Configuración para consumir eventos de la cola 46
Figura 19 Beans para el consumo de eventos. 46
Figura 20 Método de recepción de eventos 47
Figura 21 Modelo de un Job del Stage de producción 52
Figura 22 Pipelines del proyecto Integra. 52
Figura 23 Pipelines del proyecto Deploy 56
Figura 24 Lote 1 del DataImport de Solr 60
Figura 25 Creación de Collection. 61
Figura 26 Composición de la Collection IntegraCollection. 62
Figura 27 Recursos del Solr 64
Figura 28 Solución del formato fecha en trámites 66
Figura 29 Representación de un trámite en Solr con el formato fecha correcto 67
Figura 30 Gráficos proporcionados por Willy. 69

1. Introducción

El presente informe corresponde a la Práctica Profesional Supervisada (PPS) que se realizará en el marco de trabajo en Softtek, empresa dedicada a la tecnología de la información, que brinda sus servicios a la organización OSDE de atención médica. Dentro de las prestaciones se pueden mencionar las siguientes: el desarrollo de mejoras tecnológicas, soporte técnico y mantenimiento de las aplicaciones que son utilizadas por dicha empresa.

El proyecto en el que se centrará la PPS involucrará a la aplicación Web llamada **IntegraBPM,** encargada de la gestión de Workflows (flujo de trabajo de un proceso) en donde se lleva a cabo el flujo de vida de trámites. Los procesos que son utilizados en el software tienen como particularidad las definiciones que representan los tipos de datos que puede llegar a tener un trámite perteneciente al sistema y las acciones que se pueden ejecutar sobre este. Estas definiciones y estructuras de los distintos procesos son brindadas por otras aplicaciones externas que se encargan de su mantenimiento, estas suelen ser cargadas en la aplicación para poder interpretar su comportamiento. Los trámites en Integra pueden ser creados por un usuario a través de la interfaz gráfica, por medio de formularios, que se conectan a otras aplicaciones o a través de servicios expuestos por la misma aplicación. Una vez creados pueden ser tomados y trabajados por los usuarios que pertenezcan a la bandeja de trabajo de dicho proceso, pudiendo realizar acciones sobre los trámites y alterar su estado, derivarlos a otros grupos de trabajo hasta poder finalizar su ciclo de vida.

La aplicación detallada anteriormente se encuentra hoy en día desarrollada sobre tecnologías obsoletas con respecto al mercado tecnológico, por ese motivo se decidió realizar el siguiente proyecto de migración tecnológica sobre la aplicación. La propuesta consiste en mejorar su performance, su estabilidad, experiencia de usuario y el proceso de automatización de tareas para que en el sector de desarrollo se pueda optimizar mejor el tiempo para futuras

implementaciones a nivel productivo. Este proyecto será realizado mediante la aplicación de Scrum como metodología ágil, para su respectiva gestión y desarrollo. Aplicando sus principales características para la forma de trabajo, decisiones a tomar y la comunicación en cuanto a los avances.

Uno de los objetivos principales será el de automatizar la aplicación para los despliegues a través de la metodología DevOps. Esta solicitud surge a pedido del cliente OSDE hacia la empresa Softtek, ya que actualmente en distintos equipos se ha comenzado a implementar y genera diversos beneficios que son significativos para el cliente al momento de contar con una aplicación sólida, pudiendo optimizar el tiempo en el proceso de desarrollo. Uno de los motivos centrales para la utilización de la metodología es que en el caso de la aplicación Integra, esta contiene una totalidad de cuatro ambientes entre los que se diferencian un ambiente productivo y tres para pruebas. La mayoría de estos ambientes (incluyendo producción) se encuentran a cargo de un equipo externo, encargado de las tareas de despliegues de la aplicación y el control de archivos de configuración de las máquinas que la contienen. El principal beneficio de la metodología DevOps, es desligarse del equipo externo para que el proceso de implementación a nivel productivo sea automatizado y no necesite de la interferencia de terceros.

1.1 Objetivos

- Optimizar la aplicación IntegraBPM, con las tecnologías utilizadas en el mercado actual, que agregue valor y otorgue mejores herramientas que generen nuevos desarrollos para futuras mejoras.

- Mejorar la performance del sistema, aumentando la eficiencia en los procesos y reduciendo los tiempos de respuesta del usuario y aplicaciones externas por medio de la optimización de recursos.

- Generar estabilidad en el sistema, para reducir la cantidad de errores actuales y mejorar su funcionalidad.

- Aplicar al proyecto una Integración continua con la metodología DevOps, que permita automatizar las tareas de construcción, implementación y test de la aplicación en los distintos ambientes de trabajo.

1.2 Herramientas y Tecnologías

1.2.1 Java

Es el lenguaje de programación base que se utilizará en la mayor parte del código fuente de la aplicación. Este lenguaje es de tipo interpretado y facilita el modelo de programación orientada a objetos, por medio del uso de abstracción, polimorfismo, encapsulación y herencia.

Las aplicaciones hechas en Java dependen de un entorno de desarrollo de software denominado JDK, que tiene como finalidad permitir desarrollar programas en Java. Dentro de este entorno se encuentra JRE que contiene un conjunto de librerías, herramientas de interfaz y la Java Virtual Machine (JVM). En el caso de la JVM tiene como funcionalidad compilar el código desarrollado para transformarlo en uno de tipo bytecode, para ser interpretado por el procesador de la máquina. En la imagen siguiente se puede visualizar la arquitectura de la JDK:

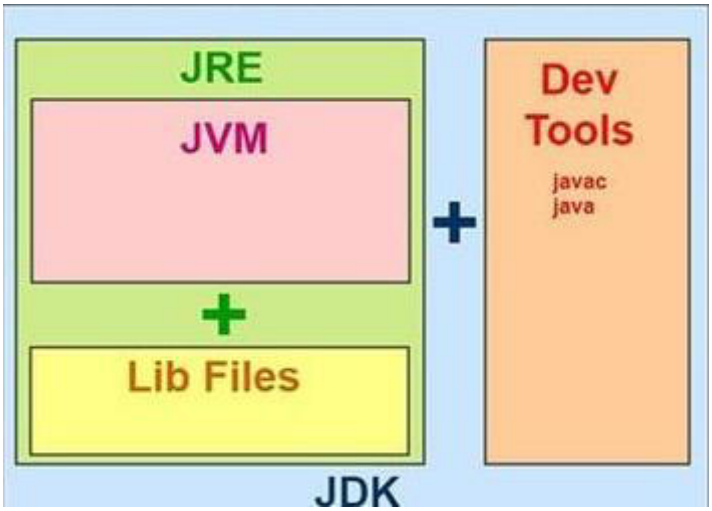


Figura 1 Arquitectura Java
Recuperado de: Recuperado de https://simplesnippets.tech/jdk-vs-jre-vs-jvm-in-java-whats-the-difference/

1.2.2 Maven

Es una herramienta de línea de comandos que se encarga de crear el formato de un proyecto. Tiene dentro de sus funcionalidades principales compilar el código, crear documentación, importar dependencias de otros proyectos y generar archivos de tipo Jar o War. En el caso del War, se trata del empaquetado de una aplicación web para su futura implementación en un servidor de aplicaciones.

El código fuente de la aplicación contiene una arquitectura basada en un proyecto Maven, el cual posee un archivo en donde se especifican las dependencias que se utilizarán en los distintos subproyectos de la aplicación y las versiones del mismo. Esta herramienta será de gran utilidad durante el desarrollo para poder indicar las versiones a utilizar en la aplicación.

1.2.3 Spring

Es un marco de trabajo de código abierto, mejor reconocido en el lenguaje técnico como un Framework para crear código de alto rendimiento, liviano y reutilizable. Este Framework trae diversas funcionalidades divididas en módulos donde se destacan las más importantes: Core container, patrón de diseño MVC (Modelo–vista–controlador), acceso a datos y programación orientada a objetos.

En el caso de IntegraBPM usa Spring como Framework central, utilizando sus características como inversión de control para el manejo del flujo de vida del proyecto. Además de la inyección de dependencias para la creación de objetos, utiliza Hibernate que realiza el mapeo de entidad-relación y almacena datos utilizando objetos de Java, en lugar del lenguaje de consulta utilizado por el motor de base de datos.

1.2.4 JBoss EAP

JBoss es un servidor suministrado por la organización Red Hat para aplicaciones de tipo Java EE (edición empresarial), el cual facilita una cierta variedad de librerías para ser utilizadas en, por ejemplo: archivos de configuración para la conexión a la base de datos, un contenedor de archivos de

logs y un espacio para poder almacenar los wars de las aplicaciones o servicios de tipo Java para ser consumidos por los usuarios.

1.2.5 Apache Solr

La siguiente herramienta es un servidor de búsqueda que tiene como función principal integrar motores de búsqueda verticales para facilitar la búsqueda de datos de forma más rápida con una indexación acelerada. Solr es utilizada para liberar carga de trabajo a la base de datos una aplicación, pudiendo satisfacer las consultas de los usuarios de una manera más rápida sobre los datos que son requeridos con mayor frecuencia y brindar una mejor performance.

1.2.6 GitLab

En el caso de GitLab, este es un servicio web de control de versiones para el desarrollo de forma colaborativa basándose en la herramienta Git. El uso de esta tecnología tendrá dos propósitos para el proyecto, en primer lugar, será utilizado para el almacenamiento del código de la aplicación y la utilización de su funcionalidad de gestión de ramas para avanzar en distintos temas y a la vez sin pisar el código que se encuentra actualmente productivo. En segundo lugar, tendrá como objetivo la funcionalidad de integración continua y despliegue continuo (CI/CD) de la aplicación, que facilita la creación, prueba y despliegue de mejoras de código de manera iterativa por medio de un conjunto de tareas automatizadas.

La implementación de Gitlab tiene como fin reducir los errores humanos y tiempos de implementación para nuevas versiones de Integra, tanto en los ambientes de prueba como el de producción, ya que son administrados por un equipo externo y los despliegues son realizados de forma manual.

1.2.7 Trello

Trello es un software utilizado para la administración de proyectos a través de una interfaz gráfica. Este permite la creación de tarjetas que pueden contener una descripción detallada de requerimientos de un proyecto o un plan de trabajo, además, estas pueden ser desplazadas dentro de una tabla donde las columnas

suelen ser el ciclo de vida de las tareas (en espera, desarrollo, prueba o finalizada). Las tarjetas de Trello pueden ser asignadas a uno o varios usuarios pertenecientes al proyecto, también se pueden agregar comentarios dentro de la mismas para poder informando del respectivo avance de la tarea.

De esta forma facilita la gestión del proyecto, pudiendo tener un seguimiento de las tareas para ver en qué etapa se encuentran y el avance de la misma por medio de recordatorios o comentarios ingresados por los responsables.

1.3 Tareas a ejecutar

El plan de migración constará de un total de cinco etapas en las cuales se identifican:

1.3.1 Planeación

En principio la tarea inicial de la primera etapa consistirá en un análisis profundo de las tecnologías y herramientas, que son utilizadas por la aplicación y afectan directamente al rendimiento de la misma. La selección de las versiones a migrar será consensuada tanto con el cliente como con el equipo de arquitectura, para utilizar las versiones homologadas que utiliza OSDE con fines de mantener la mayor seguridad y velocidad en la aplicación.

Esta etapa también contendrá una fase de investigación, en que se reunirá información sólida de las ventajas que pueden traer las versiones de las tecnologías de elección y su compatibilidad entre las mismas para ser implementadas de la manera más eficiente posible.

1.3.2 Diseño

La etapa del diseño es importante para que los cambios tecnológicos realizados durante el desarrollo de la migración puedan generar un impacto de mejora para el uso productivo del cliente. Esta fase constará con la elección de

los recursos de hardware como la memoria RAM y los núcleos del procesador para la creación de las nuevas máquinas en los distintos ambientes.

En el caso de las máquinas para los ambientes de prueba, estos deberán poseer recursos similares al nivel de producción, para que las pruebas puedan ser lo más realistas posibles en cuanto a tiempos de respuestas. Además, y para el caso del ambiente de producción, se pueda contar con una menor cantidad de máquinas, pero con mayores recursos, mejorando la productividad y performance de la aplicación.

1.3.3 Desarrollo

La fase de desarrollo consistirá en la implementación y migración de todas las tecnologías descriptas anteriormente, comenzando con la instalación del servidor JBoss en donde correrá IntegraBPM. Una vez hechas las configuraciones requeridas por el servidor, se llevará a cabo el cambio a las nuevas versiones tanto de la JDK de Java como de la herramienta Maven, para que centre las bases del software y puedan ser acopladas las demás tecnologías de una forma secuencial y sin errores.

Luego de los primeros cambios se agregará la nueva versión del Framework de Osde, que consiste en una cierta cantidad de librerías homologadas por la empresa para mantener un requerimiento de seguridad. Posterior a esta tarea se proseguirá con la migración de los entornos de trabajo Spring, Hibernate y Cxf para poner en funcionamiento las principales funcionalidades. En simultaneidad a las tareas previas, se tendrá que migrar el circuito de eventos actual a uno nuevo, que cuente con una configuración compatible con las versiones anteriores para la producción y consumo de eventos.

En la última parte la etapa de desarrollo consistirá en la creación de las nuevas instancias de Apache Solr y su integración con el programa, para poner a prueba las funcionalidades de búsquedas indexadas. Por último, se implementará el circuito de integración continua para ser aplicado en los diversos ambientes hasta llegar a la etapa de producción una vez convalidado las mejoras realizadas.

1.3.4 Pruebas

En esta etapa del proyecto se realizarán las pruebas necesarias para la búsqueda de errores con el equipo QA a cargo, con el propósito de comprobar los cambios de versiones e implementación de nuevas tecnologías. Estas tendrán como objetivo verificar todas las funcionalidades y la integración con aplicaciones. Además, se agregará un monitoreo para visualizar el rendimiento de todas las instancias y ver los valores que arrojan los recursos utilizados para realizar una retroalimentación y sacar conclusiones de los cambios implementados.

1.3.5 Informes

Como última etapa se proseguirá a la finalización de los informes, que consiste en avanzar la redacción del segundo informe que contará con la información sobre el desarrollo del proyecto de manera detallada. Por último, el Informe Final, que contará con los resultados y conclusión sobre la finalización de la práctica supervisada.

1.4 Cronograma de trabajo

En esta sección se visualizarán las tareas a realizar a lo largo del proyecto PPS mediante la utilización del diagrama de Gantt, diferenciando las subtareas de cada etapa por un color diferente.

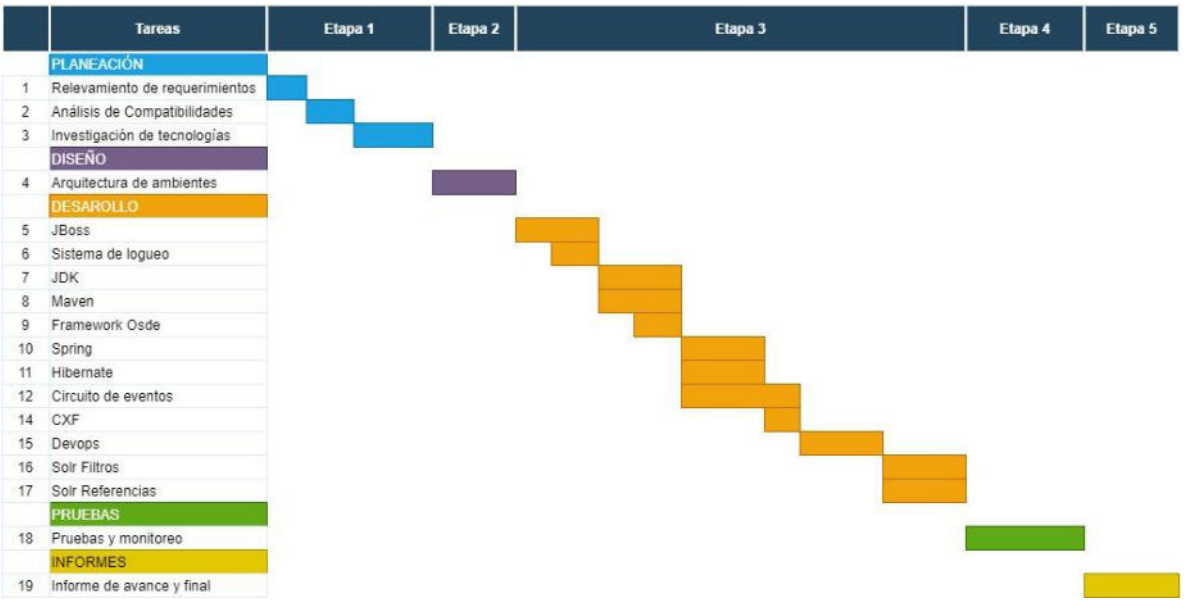


Figura 2. Cronograma de trabajo. Fuente: Elaboración propia (basada en la práctica)

2. Desarrollo

2.1 Planeación

2.1.1 Análisis y requerimientos

En primer lugar, durante la etapa de desarrollo se inició con la definición de requerimientos para la aplicación IntegraBPM y un análisis de su estructura actual. Estos requerimientos de tecnologías son esenciales para el proyecto, ya que se necesita un buen respaldo de información para conocer su uso, las mejoras que estos conllevan y las compatibilidades con otras herramientas para su futura aplicación.

Actualmente IntegraBPM se encuentra alojada en un servidor JBoss EAP de versión 4.3 dentro de una máquina Linux. Este servidor cuenta con un archivo de la aplicación web (War) que fue creado por la versión 1.6 IBM de la JDK para lo que hace referencia a la tecnología Java y el uso de una versión 2 de Maven. Por su parte, dentro del código fuente utilizan librerías como Spring 2.5.6, Hibernate 3.3.2 y Apache CXF 2.1, además, se emplea un framework procedente de OSDE que posee una cierta cantidad de librerías con las versiones ya establecidas.

En cuanto a la generación de archivos de logs para visualizar el funcionamiento del software, se utiliza la librería log4J por medio de archivos de extensión XML. Por otra parte, Integra cuenta con un archivo llamado wg.properties que es utilizado para la definición de variables, donde pueden tener diferentes valores dependiendo el ambiente en donde se esté utilizando.

Otro de los mecanismos, que es de preferencia refactorizar en el proyecto, es el circuito de eventos. Para este caso, el funcionamiento se basa en un componente denominado EDA que se encarga de la producción y consumo de eventos generados por el usuario, para enviar los datos creados o actualizados hacia la herramienta Solr en donde serán indexados.

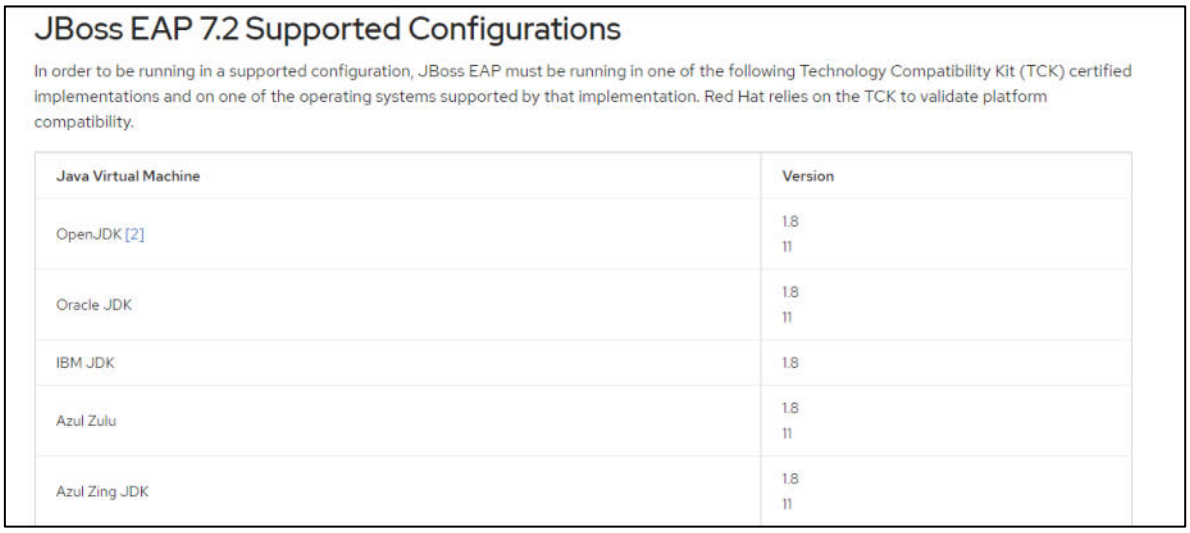
Por medio de reuniones con el cliente de OSDE y en conjunto con el equipo de arquitectura (quienes llevan a cabo la definición de versiones homologadas para la empresa), se definieron las versiones de las tecnologías que se requieren migrar durante el proyecto. Estas tecnologías son seleccionadas con fines de seguridad para respaldar información de la empresa y mejorar la performance de la aplicación. De esta manera, se consensuó con el cliente que se alcancen las siguientes versiones para las tecnologías mencionadas anteriormente:

- JBoss EAP 7.2
- JDK 1.8
- Maven 3.2.1 o superior
- Framework OSDE 2.3.1
- Apache Solr 8 o superior
- Spring 3.2.14
- Hibernate 3.6.9
- Apache CXF 2.5.11

2.1.1 Compatibilidad de herramientas e investigación

Mediante un análisis sobre las documentaciones oficiales de las tecnologías requeridas para el proyecto, se obtuvieron como resultado que el servidor JBoss EAP 7.2 requiere una versión superior a la JDK 1.8 de Java para ser utilizada. A diferencia de la versión anterior de JBoss, la utilización de esta nueva herramienta trae como novedad un archivo de configuración llamado

“Standalone”, en donde se pueden definir propiedades de sistema para ser utilizadas posteriormente en Integra. Estas propiedades del sistema son variables exteriorizadas que dependen propiamente del ambiente en donde se ejecuta la aplicación, en algunos casos dependiendo de su uso pueden ser modificadas en tiempo de ejecución. Otra característica importante que trae el uso de Standalone es la configuración de archivos de logueo para la aplicación, que permiten describir el comportamiento de los procesos que se ejecutan. Además, JBoss permite la configuración de distintas conexiones a bases de datos para ser utilizadas por aplicaciones desplegadas, en definitiva, todas estas ventajas que trae la nueva versión del servidor vienen a reemplazar a varios archivos de configuración aislados por uno de estructura centralizada.



JBoss EAP 7.2 Supported Configurations

In order to be running in a supported configuration, JBoss EAP must be running in one of the following Technology Compatibility Kit (TCK) certified implementations and on one of the operating systems supported by that implementation. Red Hat relies on the TCK to validate platform compatibility.

Java Virtual Machine	Version
OpenJDK [2]	1.8 11
Oracle JDK	1.8 11
IBM JDK	1.8
Azul Zulu	1.8 11
Azul Zing JDK	1.8 11

Figura 3 JBoss EAP 7.2 Versiones soportadas. Recuperado de https://access.redhat.com/articles/2026253#EAP_72

La versión 1.8 de la JDK de Java trae nuevas características útiles con respecto a las anteriores, entre las más destacadas se encuentran:

- **Java.time**: como una Api para trabajar con fechas, tiempos, instantes y duraciones.
- **Java.util.Stream:** permite utilizar operaciones funcionales sobre Streams.
- Agregar las expresiones **Lambda**: son funciones anónimas que no necesitan clases.
- Permite agregar comportamiento por defecto en funciones de una Interfaz.

- Nuevos métodos en clases comunes como es el caso de **Join** en String para unir dos cadenas de texto o **compareUnsigned** en Integer para comparar números.

El mismo requisito de la versión 1.8 de Java aplica para Apache Solr 8.9, que puede utilizarse hasta la JDK 13. En este caso la aplicación es utilizada para dos propósitos distintos, por un lado, se la utiliza para que el usuario pueda crear un filtro y ejecutar la cantidad de veces que sea requerido sin generar consultas a la base de datos. La otra funcionalidad que se le da a Solr, es para la búsqueda de trámites por medio de datos característicos, como puede ser una observación agregada o un valor de un campo del proceso. Ambos servicios están separados en máquinas diversas que contienen la misma aplicación dentro de una arquitectura de clusters, el primero denominado Solr de Filtros y el segundo como Solr de Referencias, en donde las instancias de un mismo clúster se encuentran comunicadas para réplica de datos y la alta disponibilidad ante una gran cantidad de solicitudes.

La arquitectura en que trabajan los Solrs es de modelo distribuido (Clúster), en donde cada instancia de una aplicación Solr se comporta como si fuera un nodo aislado. Para una sincronización y comunicación entre nodos es requerido el uso del servicio ZooKepeer, que permite mantener de forma centralizada los archivos de configuración e índices de datos de manera que se distribuyan automáticamente entre los integrantes del clúster. La manera en que trabajan los nodos en Solr, es creando un índice de búsqueda (Collection) donde se almacenan los datos, estos pueden fragmentarse en un subconjunto lógico llamado Shard, que a su vez contienen nodos de Solr. Esta arquitectura posibilita que dentro de un Shard se designe a un nodo como líder para coordinar la indexación del fragmento, también es posible agregarle más nodos para obtener redundancia de datos por medio de réplica.

En la siguiente captura se puede representar cómo es la arquitectura de un clúster de nodos en Solr.

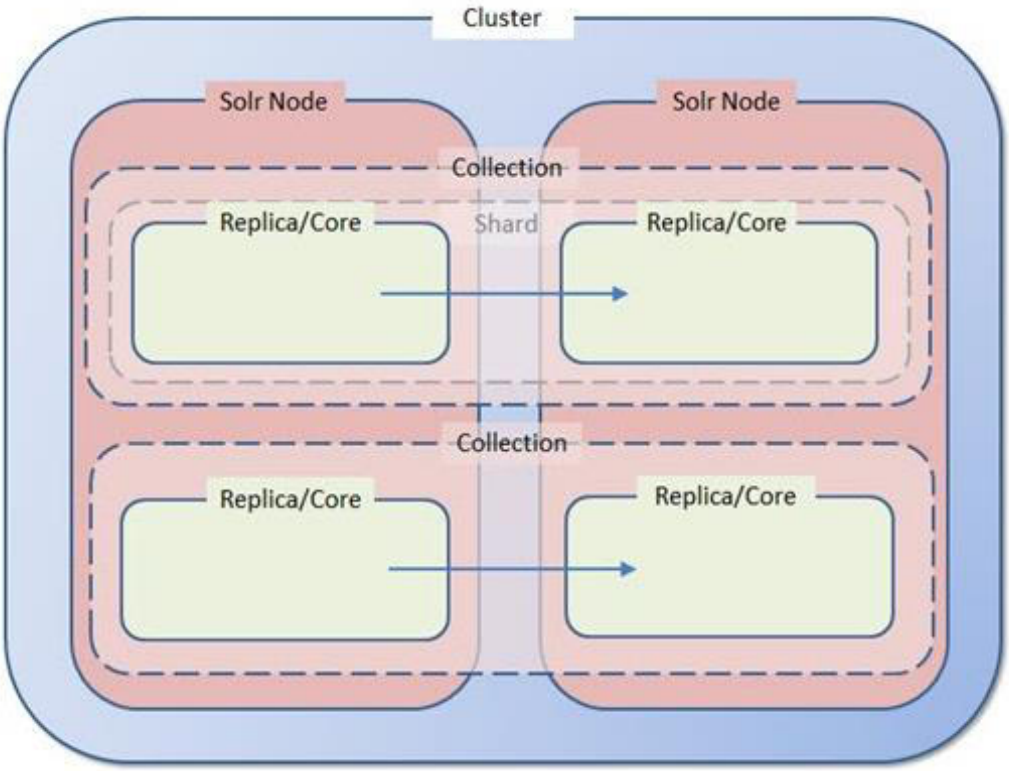


Figura 4 Arquitectura Solr.
Recuperado de: https://www.searchstax.com/docs/hc/solr-collection-core-shard-replica/

Por el lado de las ventajas que trae Solr con la versión 8 contiene una gran cantidad de cambios, teniendo en cuenta que se superan varias versiones de por medio. Dentro de los cambios y nuevas características que estas versiones traen, se mencionan a continuación las más relevantes:

1) Se encuentran muchas configuraciones por defecto que Solr toma por cuenta propia si no son modificadas en el archivo solrConfig.xml.
2) El archivo schema.xml donde se definen los tipos de datos que se van a indexar, se modifican los identificadores de una cierta cantidad de campos con similares comportamientos.
3) Modificaciones en las clases que emplean SolrJ -la librería que se utiliza para establecer la comunicación con los nodos de Solr-.
4) Cambio en la metodología de asignación de líderes en un Shard -se agregan nuevos tipos de réplicas al modo NTP como TLOG y PULL-.
5) Los nodos de Solr admiten solicitudes HTTP / 2.

6) Cambio en la configuración de Java Management Extensions (JMX) - es una tecnología Java que permite administrar y/o supervisar aplicaciones, objetos instanciados y dispositivos del servicio-.
7) Permite la replicación entre centros de datos -el clúster tiene la posibilidad de replicar en otro centro de datos y monitorear con la ayuda de una nueva interfaz de programación de aplicaciones (API)-.

Hibernate 3.6.9 puede trabajar a partir de la versión 1.5 de Java, además, contiene las siguientes librerías entre sus dependencias: hibernate-commons-annotations 3.2.0. Final, hibernate-jpa-2.0-api 1.0.1. Final y javax.transaction 1.1 entre otras.

La nueva versión de Spring Core 3.2.14 trae la compatibilidad para trabajar con la JDK 1.8 y la versión 1.1.3 de commons-logging para los registros basados en Java. También permite la opción de traer otras dependencias útiles como es Log4j 1.2.17, net.sf.jopt-simple 3.3 y org.aspectj 1.7.4. Es importante aclarar que para el caso del framework de Spring también es imprescindible nivelar todas las versiones usadas de sus subproyectos.

La versión 2.5.11 de la herramienta Apache CXF incluye una considerable variedad de mejoras y correcciones de errores con respecto a la versión anterior, utilizada hasta el momento por Integra (versión 2.1). Para empezar a utilizar este framework se piden como requisitos tener una versión igual o superior a la JDK 1.5 de Java y 2.2.1 de Apache Maven. En cuanto a las nuevas funciones que trae CXF, se pueden encontrar las siguientes:

- WS-SecurityPolicy (Seguridad en servicios web)
- WS-SecureConversation (Especificación de servicios web)
- Soporte para JAX-RS, incluyendo una Api para acceder a servicios Rest
- Admite la clase de excepción Marshall con la etiqueta @XmlAccessorOrder en JAXBEncoderDecoder
- Nuevas anotaciones de los primeros casos de uso de Java:
    - @WSDLAnotación de documentación para agregar nodos de documentación al WSDL generado

- @SchemaValidation anotación para activar la validación del esquema
- @DataBinding para establecer el enlace de datos utilizado (si no es JAXB)
- @GZIP para activar la compresión GZIP
- @FastInfoset para activar la compatibilidad con FastInfoset
- @Logging para encender y controlar varias funciones de registro
- @EndpointProperty para configurar las propiedades del punto final
- @Policy para asociar documentos WS-Policy con el servicio

2.1 Diseño de ambientes

Para la siguiente etapa del proyecto se desea establecer los recursos con los que contarán los distintos ambientes (desarrollo, testing, preproducción y productivo) en donde se desplegará la aplicación IntegraBPM. Se requiere que estos ambientes dispongan recursos de hardware similares a las máquinas que se encuentran actualmente, además, que cuenten con las mismas aplicaciones que utiliza Integra en producción para disponer de un circuito completo. Las aplicaciones con las que deberán contar los entornos dentro de un JBoss 7 son IntegraForms y SelfIntegration, ambas tienen como función principal albergar los modelos que definen la estructura y comportamiento de un proceso en Integra. Por último, se encuentra la aplicación Index Service, que se encarga de levantar los eventos dentro de una cola para indexar sus datos en el clúster del Solr de Referencias. Estos datos pueden contener, por ejemplo, la creación de un trámite o modificación de alguno existente.

Cabe destacar que en los ambientes que contengan más de una instancia de la aplicación serán accedidos por balanceador (una dirección centralizada) y no se conectarán directamente al nombre de dominio. Esta modalidad de trabajo permite que no se sobrecarguen las máquinas con el flujo de trabajo y se pueda distribuir de forma equitativa en las demás instancias, del mismo modo, permite dar de baja alguna que se encuentre fallando en algún momento determinado.

A continuación, se detallarán los recursos que contendrán los nuevos ambientes:

1) Ambiente de Desarrollo:

En la actualidad se encuentran a disponibilidad dos máquinas que cuentan con la aplicación Integra y solo una de ellas posee la integración con la aplicación Solr. Por esta razón, el diseño del entorno de desarrollo se planificó para que se utilice una sola máquina dedicada que contenga todos los softwares, con una distribución Linux Red Hat versión 8, 4GB de memoria Ram y una CPU que contenga 2 Cores. A su vez el ambiente contará con las siguientes aplicaciones:

- Un servidor JBoss 7.2 que aloje IntegraBPM, IntegraForms, SelfIntegration e IndexService.
- 2 instancias para Solr de Referencias en un servidor Jetty para cada una y un Zookeeper.
- 2 instancias para Solr de Referencias en un servidor Jetty para cada una y un Zookeeper.

2) Ambientes de Testing:

En el caso del marco de Testing se diferencian dos ambientes, uno para pruebas internas del equipo similar al de desarrollo, pero con más recursos y otro en donde poseen acceso equipos externos para pruebas de integración con sus aplicaciones. Para poder diferenciarlos se les asigna un nombre identificatorio, para el caso del ambiente de pruebas de integración se lo denomina Testing Genérico y para el otro Testing Evolutivo.

El diseño de Testing Evolutivo contará con dos máquinas que contengan una distribución de Linux Red Hat versión 7, 4096 MB para la memoria Ram y una CPU con dos Cores. El primer hosting incluirá un JBoss 7.2 con IntegraBPM, Index Service y dos instancias de Solr Referencias con un Zookeeper, por otro lado, la segunda máquina tendrá un JBoss 7.2 con las aplicaciones IntegraBPM, IntegraForms y SelfIntegration. Además, esta última hará uso del servicio de Solr Filtros con la integración de un solo Zookeeper.

El ambiente de Testing Genérico tiene como objetivo principal simular el comportamiento de interacciones entre aplicaciones que se realizan en

producción, por eso el armado de este entorno debe contener recursos similares al ambiente productivo. Por ese motivo, tendrá una cantidad de cinco máquinas disponibles, donde dos de ellas serán exclusivas para contener el software IntegraBPM con IntegraForms, una memoria máxima de 4096 MB y dos cores en su CPU. Otra de las máquinas incluirá un JBoss 7.2 donde almacenará por separado la aplicación SelfIntegration e Index Service, contando con una cantidad de 4GB en la memoria. Por último, las dos máquinas restantes serán destinadas a albergar los clusters de Solr, donde cada una dispondrá una cantidad de 9GB de memoria Ram para su uso. Para el caso del Solr de Filtros la composición de instancias será de dos para sus nodos y un servicio Zookeeper, en cuanto para el Solr de Referencias consistirá de cuatro instancias de Solr y tres para los Zookeepers.

3) Ambiente de Preproducción:

Antes de llegar al entorno de producción, se dispone el último ambiente denominado preproducción donde la finalidad de este mismo es trabajar con un volumen de datos parecidos al uso diario productivo. Para representar este contexto se trabaja con una copia de la base de datos productiva, para que el impacto de las pruebas en este ambiente sea lo más realista posible. En cuanto al diseño del entorno se utilizarán en principio dos máquinas con 4GB cada una y la misma distribución de aplicaciones que se lleva en Testing Evolutivo.

4) Ambiente de Producción:

El actual ambiente productivo se encuentra en funcionamiento con seis máquinas que alojan la aplicación Integra con IntegraForms dentro de un servidor JBoss 4.3, donde cada una posee una cantidad de 4GB de memoria Ram para su uso. Durante la migración de tecnologías se llevará a una versión 7.2 de JBoss para contener las mismas aplicaciones, pero con diferentes versiones entre sus herramientas. A causa de este cambio el uso de instancias será minimizado, ya que la performance de la aplicación aumentará de manera que se utilicen una cantidad de cuatro máquinas para alojar el software de IntegraBPM. Por otro lado, ambos servicios de Solr seguirán contando con la misma cantidad de recursos para su funcionamiento, para el servicio de Filtros cuatro instancias de Solr (dos para replicas) y tres instancias de Zookeeper para

la distribución de datos y nivelación. En cuanto al servicio de Referencias también tendrá cuatro instancias de Solr y tres para Zookeeper al igual que el Solr de Filtros.

2.1 Migración de JBoss

Para realizar la migración del servidor JBoss a la versión 7.2 se tuvo que descargar una versión proveniente de la guía de ambientes de Arquitectura de OSDE, esta versión llegó con algunas configuraciones de seguridad y módulos que contienen otras librerías para ser utilizadas. Los módulos en el caso de querer ser utilizarlos o no, se declaran a través de un archivo de configuraciones llamado “jboss-deployment-structure”, que sirve para gestionar la carga de clases en la implementación de la aplicación. A su vez Integra excluyó el subsistema WebServices ya que no sería utilizado y se agregaron módulos tales como Javax, Xerces y Jmx, entre otras.

Uno de los módulos utilizados en Integra es el de Xerces, que se usa para la validación, análisis sintáctico y manipulación de documentos XML. Con la versión que utiliza JBoss originalmente se obtuvieron errores a la hora de cargar las definiciones de procesos, ya que usaba una validación extra que filtraba algunas definiciones y no podían ser enviadas a la aplicación destino SelfIntegration. En este caso se descargó una versión menor a la actual para evitar ese tipo de validaciones y no afectar las definiciones de los demás equipos.

Para la inicialización del servidor se creó un script (secuencia de comandos) de arranque, donde se declara la ubicación de la JDK que va a utilizar, configuraciones del Garbage Collector, la ubicación y modelo del Standalone que se implementará. El script también facilita el inicio de JBoss y permite la declaración de variables y configuraciones propias que va a tener el servidor.

Dentro del directorio Standalone del JBoss existe el siguiente sistema de directorios:

- Deployments: Información de servicios o aplicaciones implementadas.
- Configuration: Archivos de configuración para el servidor autónomo.
- Lib: Bibliotecas externas.

- Tmp: Datos temporales

Una de las configuraciones importantes es la sincronización entre las instancias de Integra, para generar la comunicación fue necesario la configuración de los archivos “integra-jgroups-config” e “integra-ehcache”. Para el primero de los archivos se declaró el protocolo de transporte TCP y la elección de un puerto para la comunicación, además, se fija el tiempo de espera de respuesta, número de integrantes y el host inicial. Para la configuración de integra-ehcache se definen las instancias que componen al clúster con su puerto de comunicación, también se definen las caches que van a ser utilizadas con la ruta de clase y sus propiedades. Las propiedades en la definición de una cache, hacen referencia a la acción que van a contemplar para replicar la información en las demás instancias, por ejemplo: replicar cuando se agrega un nuevo dato, se elimina, actualiza o copia.

Sistema de logueo:

Para la creación de archivos de logueo Integra utilizaba la librería de logs llamada Log4j, esta misma es referenciada en el proyecto raíz como una dependencia externa al proyecto. La finalidad de la librería es escribir mensajes de registro de la aplicación durante su flujo de vida, es decir, que se pueden imprimir los errores generados y mensajes que detallen el comportamiento de datos en una clase.

Log4j trabaja con un archivo de configuraciones dentro del código donde se declaran las salidas de destino, denominados Appenders. Los Appenders pueden configurarse tanto en archivos de tipo XML como de Java, en ellos se declaran la ubicación de salida de los logs, el nivel de logueo a filtrar y detalles del archivo que se creará.

Por recomendaciones del equipo de Arquitectura y problemas de compatibilidad con las nuevas versiones de JBoss, se tomó la decisión de migrar la librería Log4J por Slf4j que trae nuevas mejoras en su implementación. Una de las mejoras es la utilización de una única configuración de registros, esto quiere decir, que es independiente de cualquier implementación de registro específica y no es necesario administrar múltiples configuraciones. Otra ventaja que trae Slf4j es que proporciona registro de registros basados en marcadores

de posición para mejorar la legibilidad de los logs, eliminando las comprobaciones como isDebugEnabled(), isInfoEnabled(), etc. Para el uso de Slf4j se tuvo que importar la librería en cada una de las clases que contenga código de logs, además, en la instanciación se reemplaza la clase Logger de Log4j por la de LoggerFactory de Slf4j, utilizando el método getLogger con un parámetro que recibe la clase origen como argumento.

La librería tiene a disposición una lista para categorizar los niveles de prioridad de logueo, entre ellos se distinguen los siguientes:

- TRACE: El nivel más bajo que muestra todos los logs.
- DEBUG: Contiene los mensajes de depuración, por ejemplo, mensajes para seguir el flujo de un método.
- INFO: Detalla información esencial de depuración.
- WARN: Advertencia de un posible fallo que no altere el flujo del sistema.
- ERROR: Errores no bloqueantes del sistema, cuando no son controlados por excepciones.
- FATAL: Sirve para errores críticos, de tipo bloqueante que no permiten seguir con el flujo del sistema.

JBoss EAP utiliza su propio subsistema de registros llamado “JBoss LogManager” que centraliza toda la configuración de registros, sirviendo de apoyo a la librería de Slf4j que envía todos los registros al subsistema. Esta herramienta utiliza Handlers para reemplazar la misma funcionalidad que los Appenders de Log4j, para establecer el comportamiento y estructura de los archivos de logs. En este caso la configuración correspondiente de los Handlers se lleva a cabo dentro del Standalone.xml.

JBoss posee ocho tipos de Handlers para controlar los mensajes de registros capturados, entre ellos se encuentran:

- Console: los mensajes son expuestos en el flujo de salida del sistema operativo que inicia el servidor. Para este tipo de Handler los mensajes no son almacenados en un archivo externo.
- File: escribe los mensajes en un archivo específico.

- Size: los mensajes se almacenan en un archivo limitado por un tamaño de memoria, cuando se alcanza dicho tamaño se le agrega un sufijo numérico y se vuelve a crear uno nuevo con el nombre original.
- Periodic: escribe los mensajes en un archivo determinado por un periodo de tiempo específico, una vez finalizado el periodo le agrega al nombre una extensión del tiempo y crea un nuevo archivo para seguir logueando.
- Periodic Size: es una mezcla entra los Handlers Periodic y Size, en tal caso el límite para crear un nuevo archivo está definido tanto por el periodo de tiempo como el tamaño máximo especificado.
- Syslog: permite enviar los registros a un servidor remoto centralizado.
- Custom: se puede incluir en un módulo, implementado como una clase Java que extiende de "java.util.logging.Handler". También se puede agregar el Appender de Log4j.
- Async: controlar el comportamiento asincrónico para los casos que haya alta latencia o problemas de rendimiento.

En la configuración del Standalone para la aplicación IntegraBPM, se decidió por usar los Handlers de tipo Size y asincrónicos. La elección de los tipos de handlers fue en primer lugar para mantener el mismo comportamiento que se utilizaba con Log4j, por otra parte, el motivo es que la creación de archivos con un límite de almacenamiento que facilita la lectura de los mismos.

En la siguiente figura se puede observar cómo se define un Handler Size, identificado con el nombre de INDEX_SERVICE. Dentro de su configuración se define el nivel mínimo de logueo (INFO), el tipo de encoding que va a utilizar, el nombre del archivo y la capacidad máxima en MB para crear uno nuevo. Por último, se declara la cantidad máxima de archivos que se pueden crear y la metodología Append, para saber si los nuevos mensajes serán almacenados en un nuevo archivo o en uno ya existente.

```
<size-rotating-file-handler name="INDEX_SERVICE" autoflush="true">
    <level name="INFO"/>
    <encoding value="utf-8"/>
    <formatter>
        <pattern-formatter pattern="[%t]|%d{dd/MM/yy HH:mm:ss}|[%c{1}].%M()-%m%n"/>
    </formatter>
    <file relative-to="jboss.server.log.dir" path="index-service.log"/>
    <rotate-size value="5m"/>
    <max-backup-index value="5"/>
    <append value="true"/>
</size-rotating-file-handler>
```

Figura 5 Handler de tipo Size. Fuente: Elaboración propia (basada en la práctica)

En la captura también se puede identificar un tipo de formato, este sirve para agregar información relevante a las lineas de logs. En este ejemplo se imprimen la fecha y hora de cuando se genera el mensaje, como también la clase y método de donde proviene el log.

Además de la declaración de los Handlers, se deben definir los Loggers en el archivo Standalone. Un logger describe la ubicación de la clase que utilizará el Handler, el nivel mínimo de logueo y el nombre del Handler al que pertenece. A continuación, se puede observar un ejemplo de un Logger asociado al Handler de la captura anterior.

```
<logger category="ar.com.osde.wf.eda.events.consumer" use-parent-handlers="false">
    <level name="INFO"/>
    <handlers>
        <handler name="INDEX_SERVICE"/>
    </handlers>
</logger>
```

Figura 6 Definición de Logger. Fuente: Elaboración propia (basada en la práctica)

Propiedades:

En cuanto a la implementación de propiedades (también llamado properties en inglés) utilizadas por la aplicación Integra, actualmente se utiliza un archivo llamado “wg.properties” que se encuentra dentro del código fuente del software. La funcionalidad de este archivo surge de utilizar valores estáticos o dinámicos que levanta la aplicación durante su inicio, para manejar la lógica de los servicios utilizados. Uno de los usos del wg.properties puede ser para declarar la capacidad máxima de archivos adjuntos, que permite la aplicación o la activación de la búsqueda por Solr.

Las nuevas propiedades que utiliza JBoss 7.2 traen como ventaja, cambiar sus valores mientras la aplicación se encuentra corriendo, esto significa, que un usuario tiene la posibilidad de modificar el valor de una property mediante una acción en una vista. Existen dos tipos de formas para poder cambiar el valor a una propiedad, una de ellas es mediante el código dentro de la aplicación y la otra es por medio de una conexión al servidor. La última alternativa mencionada permite por medio de la ejecución del ejecutable "jboss.cli" conectarse a una interfaz para configurar el Standalone, para este caso sería agregar, modificar o eliminar una propiedad en el sistema.

Para la migración de todas las propiedades del wg.propierties, se creó un archivo de extensión Cli para conectarse al JBoss, donde contiene la totalidad de las variables a crear. La sintaxis de este archivo tiene una estructura en donde se ingresa el nombre de la property y su valor correspondiente, a continuación, se representa un ejemplo genérico de su estructura:

/system-property=nombre:add (value=valor)

La ejecución del archivo mencionado anteriormente creó una configuración en el Standalone.xml, como se muestra en la Figura #7, se puede apreciar la forma en que se almacenan las propiedades dentro del archivo Standalone.xml. Se puede identificar una conformación compuesta por el nombre de la propiedad (parámetro name), el cual debe ser de tipo único para poder ser diferenciando de las otras propiedades, en último lugar se encuentra el valor que contiene dicha propiedad.

```
<property name="solr.backend.enabled" value="true"/>
<property name="solr.maxResult.count" value="1"/>
<property name="solr.query.rows.limit" value="50"/>
```

Figura 7 Properties dentro del Standalone. Fuente: Elaboración propia (basada en la práctica)

Configuración de la base de datos:

Previamente la configuración para la base de datos que usa Integra y las demás aplicaciones alojadas en el servidor JBoss 4.3, debían contener un

archivo Xml ubicado en el directorio “deployments”. Con el cambio a la nueva versión de JBoss se migró esa misma configuración al subsistema “datasources" del Standalone, este permite la gestión de distintas bases de datos y la posibilidad de indicarle las condiciones de conexión. Dentro del subsistema mencionado se agregaron los siguientes puntos relevantes:

- Driver del motor de base de datos (Se utiliza el de DB2).
- Nombre JNDI y puerto de conexión.
- Schema a utilizar.
- Credenciales de conexión a la base.
- Validaciones para la conexión.
- Tipo de transacciones (Se usa transaction_read_committed).

Ubicación de despliegues:

En la versión anterior de JBoss los despliegues de las aplicaciones se realizaban ubicando el archivo de extensión War en un directorio de nombre “deployments”, con la versión actualizada del servidor se mantiene la opción de utilizar la misma metodología, pero agregando una opción mejorada. El nuevo camino utilizado para disponibilizar los despliegues en el servidor es por medio de la consola de administración o el CLI, esto quiere decir, que se crea una conexión al servidor y este genera una ubicación por medio de un código hash en donde se almacena el War de la aplicación. Esta metodología de despliegues permite que una aplicación pueda tener un estado disponible o no según sea requerido, sin necesidad de eliminar el paquete War.

2.2 Migración de Java y Maven

Para la migración de la JDK de Java a la versión 8 y Maven 3.2.1 fue necesario configurar inicialmente el entorno donde se llevará a cabo la generación del War de Integra, por esta razón, se generaron modificaciones en el entorno de desarrollo integrado (IDE). Es importante destacar que, para la etapa actual del proyecto, la compilación (crear el archivo War) de la aplicación

la lleva a cabo el IDE, ya que posteriormente estas acciones serán propias de la metodología DevOps con la herramienta GitLab. Para configurar el IDE, primero fue necesario descargar la versión de JDK 1.8.0_251 y un Maven 3.2.1 o superior. En el caso de Maven fue necesario agregar en el archivo setttings.xml la ubicación local destino en donde se descargarían las dependencias del proyecto, además la url del repositorio de OSDE para descargar las dependencias solicitadas.

Anteriormente para compilar el proyecto Integra en el IDE había que cambiar manualmente en cada uno de los subproyectos la versión de Java, esto era un proceso que requería bastante tiempo y por eso se decidió automatizarlo. Para automatizar este proceso se agregó el plugin “maven-compiler-plugin” al archivo pom.xml del proyecto raíz de Integra, esta herramienta compila las clases fuentes del proyecto con la versión de Java ingresada. Con la utilización del plugin de Maven ya no es necesario indicar por separado la versión de Java que se va a utilizar en el proyecto, sino que se define una sola vez de forma centralizada en el proyecto.

Una vez generado el cambio de la JDK en el proyecto se visualizaron errores en el ordenamiento de datos en diferentes lugares de uso en Integra. Este hecho se genera por consecuencia de que cambia la sintaxis para ordenar colecciones de datos con la versión 8 de Java con respecto a la anterior. Integra utilizaba el método Sort de la clase Collections que recibía como parámetro una lista para ordenar, en cambio en la versión 8 de Java se reemplazó el mismo método, pero recibiendo un objeto Comparator. El comparador que se ingresa utiliza un método comparing con el objeto y referencia por el cual se desea ordenar. A continuación, se agrega un ejemplo de los cambios en la sintaxis:

Collections.sort(lista)→lista.sort(Comparator.comparing(objeto::metodo))

Otra alternativa que se utilizó para el ordenamiento de colecciones fue el uso de expresiones Lambda, una herramienta nueva que se suma a la migración de JDK. Las expresiones Lambda son conocidas como funciones anónimas que no necesitan ser declaradas para ser utilizarlas. En el siguiente ejemplo se ingresa una colección de adjuntos llamada attachments y el uso de Lambda como

parámetros, que lo que hace es comparar entre los objetos adjuntos de la lista y los ordena por el campo fecha por medio del método getFecha. En este ejemplo de fragmento de código, se utiliza un ordenamiento de manera descendente en los adjuntos de un trámite.

Collections.sort(attachments, (SimpleAttachment a, SimpleAttachment b) -> b.getFecha().compareTo(a.getFecha()));

El error de ordenamiento afectó, por ejemplo, a la lista de observaciones o adjuntos de un trámite, con el cambio de la nueva sintaxis en el código se pudo corregir dicho bug (error de software). En la siguiente captura (#8) se puede observar el correcto ordenamiento de observaciones en un trámite de ejemplo, luego de haber modificado el código.

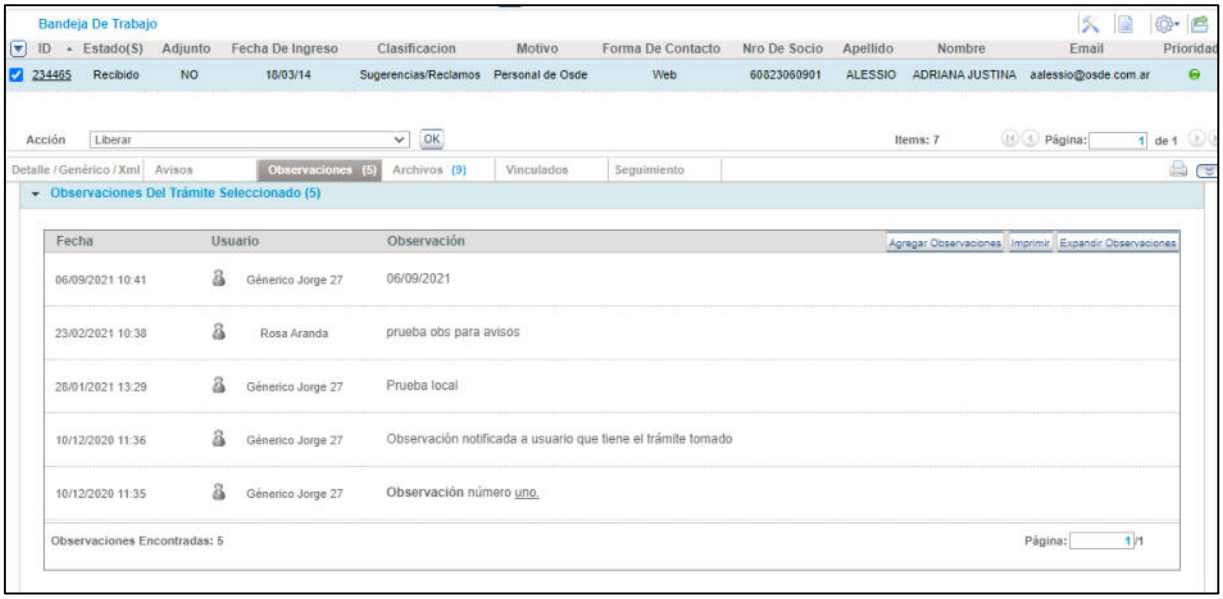


Figura 8 Ordenamiento de observaciones. Fuente: Elaboración propia (basada en la práctica)

Otro de los errores encontrados al utilizar la versión 8 de Java, ocurrió en la inicialización de las clases que almacenan datos numéricos. Las clases como Long, que sirve para almacenar datos con un tamaño de 64 bits o Int para los de 32 bits, se encontraban inicializados para que reciban de parámetro un tipo de dato “Int”. En la nueva versión de Java este tipo de constructor se encuentra deprecado y la sugerencia de la documentación oficial recomienda usar el

método estático valueOf(long), ya que trae una mejora en el rendimiento de espacio y tiempo.

2.3 Migración de Frameworks

Integra como otras aplicaciones del cliente OSDE utiliza una dependencia llamada “framework-osde” que trae consigo la declaración de diversas tecnologías y sus versiones para el uso en la aplicación. Dentro de este framewrok se encuentran tecnologías como los componentes de Spring, Hibernate, Slf4j, entre otras con la exposición de sus versiones. Con el cambio de versión en la dependencia de framework-osde por la 2.3.1, que fue el objetivo en la planificación, en consecuencia, se obtienen las versiones requeridas de los componentes de Spring e Hibernate.

Se realiza una depuración de las dependencias repetidas, que se encontraban tanto para el proyecto raíz de Integra como para sus subproyectos. Este cambio beneficia a Integra en cuanto a dar un orden en el uso de sus dependencias y que, además, no se interponen las distintas versiones de una misma tecnología.

En el caso de la migración de Hibernate la lógica en el código resultó siendo la misma sin ningún cambio en su sintaxis durante la implementación, por otra parte, se añadieron nuevas características sobre funcionalidades extras. También se generó una mejora de rendimiento y corrección de algunos bugs que se generaban con la antigua versión.

Para Spring 3.2.14 el cambio tuvo un impacto orientado hacia algunas clases que fueron deprecadas durante el cambio. Una de las clases deprecadas en las configuraciones de los Beans en Integra fue AttributesJmxAtributeSource, encargada de monitorizar y administrar la aplicación IntegraBPM, como también sus objetos instanciados. La funcionalidad de la clase deprecada de Spring fue reemplazada por AnnotationJmxAttributeSource, posibilitando el monitoreo el comportamiento de la JDK utilizada por Integra.

2.4 Migración de Apache CXF

Para la migración del Framework Apache CXF encargado de la creación de servicios web en Integra, se utiliza el archivo de gestión de dependencias pom.xml del proyecto para declarar una variable con la versión a utilizar de CXF. Una vez declarada la versión 2.5.11 en el pom raíz del proyecto, los artefactos (identificadores de la dependencia) utilizados del subproyecto integra-core como, por ejemplo: cxf-rt-frontend-jaws o cxf-rt-core utilizan la variable para hacer referencia a la misma versión.

La nueva versión de Apache trajo consigo una capa de seguridad para proteger a la aplicación de posibles ataques D.O.S (denegación de servicio), en consecuencia, Integra empezó arrojar un error en su compilación de “Cannot créate a secure XMLInputFactory”. Una posible solución no recomendada era que se declare como insegura a la aplicación, modificando la propiedad del sistema “org.apache.cxf.stax.allowInsecureParser” en true (verdadero). Siguiendo con el análisis se encontró una solución más óptima que consistía en adjuntar las dependencias “stax2-api” and “woodstox-core-asl”, ambas compatibles con la herramienta CXF. En primer lugar, las dependencias de Woodstox siguen protegiendo a la aplicación de un ataque D.O.S y se encargan de realizar un control sobre el tamaño del mensaje de tipo XML entrante.

Además de la capa de seguridad, que provee CXF con respecto a la denegación de servicio, se agregan una serie de configuraciones de seguridad:

Configuración	Defecto	Descripción
org.apache.cxf.stax.maxChildElements	50000	Máximo de elementos secundarios para un elemento principal determinado.
org.apache.cxf.stax.maxElementDepth	100	Cantidad máxima de un elemento.
org.apache.cxf.stax.maxAttributeCount	500	Máximo atributo de elementos.
org.apache.cxf.stax.maxAttributeSize	64K	Tamaño máximo de un solo atributo
org.apache.cxf.stax.maxTextLength	El 128M	Tamaño máximo del valor de texto de un elemento
org.apache.cxf.stax.maxElementCount	Long.MAX_VALUE	Máximo de elementos en el documento XML

org.apache.cxf.stax.maxXMLCharacters	Long.MAX_VALUE	Máximo de caracteres analizados por el analizador

Tabla 1 Configuraciones de seguridad para servicios de CXF. Fuente: Elaboración propia (basada en la práctica)

Otro de los cambios que se generaron al subir la versión de CXF, fue la de no aceptar mensajes con caracteres especiales en los request de los servicios. Este bug encontrado en Integra afectaba a cualquier request o response que tuviera en algunos de sus campos un carácter especial, por ejemplo, alguna palabra con acento. Al enviar un request con las mencionadas características, el mensaje nunca llegaba a la clase encargada de gestionar los request de los servicios, de lo contrario devolvía una respuesta de error.

En la figura #9 se puede observar un ejemplo del método para agregar una observación del servicio Trámite, en el mensaje de la observación se agrega la palabra “observación” que contiene un acento.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/sc
   <soapenv:Header />
   <soapenv:Body>
      <wfs:addObservacion>
         <!--Optional:-->
         <idTramite>4041242323</idTramite>
         <!--Optional:-->
         <user>WG00000127</user>
         <!--Optional:-->
         <observacion>confirmación de mensaje</observacion>
      </wfs:addObservacion>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 9 Ejemplo de request del servicio Trámite. Fuente: Elaboración propia (basada en la práctica)

En el caso de la respuesta del servicio se obtiene un response con el siguiente mensaje “Error Reading XMLStreamingReader”, se adjunta una captura (#10) del response del ejemplo anterior.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Client</faultcode>
         <faultstring>Error reading XMLStreamReader.</faultstring>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Figura 10 Response con error de enconding. Fuente: Elaboración propia (basada en la práctica)

Mediante un análisis del problema se llevó a la conclusión, que estos errores se producían por diferencias en la decodificación de caracteres (encoding), es decir, Integra trabaja con un encoding distinto al que se envía en los mensajes. Por el lado de Integra se utiliza un encoding de tipo ISO-8859-1 y los request de los servicios se envían en UTF-8.

Para la resolución del problema de los caracteres especiales, se decidió utilizar un Interceptor tanto para la entrada de los servicios como para la salida. CXF trae la herramienta de interceptors para poder leer, transformar y validar los mensajes, además, permite procesar los encabezados. Para interceptar los mensajes, los interceptors utilizan fases que son equivalentes al momento en que se quiere detener el proceso, para los casos de entrada y salida existen las siguientes fases importantes:

Fases de envío	**Funciones**
RECEIVE	Nivel de proceso de transporte
READ	Lectura del encabezado
PROTOCOL	Procesamiento de protocolos
UNMARSHAL	Desagrupamiento de la solicitud
PRE_INVOKE	Acciones previas a la invocación
INVOKE	Invocación del servicio

Tabla 2 Fases para la entrada del request en el servicio. Fuente: Elaboración propia (basada en la práctica)

Fases de respuesta	**Funciones**
SETUP	Cualquier configuración para las siguientes fases.
PREPARE_SEND	Apertura de la conexión.
WRITE	Escritura de protocolos del mensaje.
STREAM	Procesamiento a nivel bytes del mensaje.

SEND	

Tabla 3 Fases del response en el servicio. Fuente: Elaboración propia (basada en la práctica)

Para la solución se crea una clase llamada HeaderInInterceptor que hereda los métodos de AbstractSoapInterceptor, la finalidad de esta clase será interceptar los mensajes de entrada al servicio para transformar el encoding del encabezado SOAP. En el constructor del interceptor se ingresa como parámetro la fase Recive que va a detener el mensaje del servicio en el nivel de transporte. Para realizar el cambio de la decodificación de caracteres se utiliza el método handleMessage, que recibe el mensaje como parámetro, luego con el método "put" se modifica el campo Encoding del encabezado en el request y se ingresa el valor "ISO-8859-1". Como se expresó anteriormente el error de caracteres especiales sucedía tanto para mensajes entrantes como salientes, por lo tanto, también se creó un interceptor para la salida de mensajes para los servicios, de ahí que se utiliza el interceptor llamado HeaderOutInterceptor para la fase Setup como se muestra en la captura #11.

```
public class HeaderOutInterceptor extends AbstractSoapInterceptor {

    public HeaderOutInterceptor() {
        super(Phase.SETUP);
    }

    @Override
    public void handleMessage(SoapMessage message) throws Fault {
        message.put("org.apache.cxf.message.Message.ENCODING", "UTF-8");

    }
```

Figura 11 Interceptor para arreglar el enconding. Fuente: Elaboración propia (basada en la práctica)

Una vez declarados los interceptors que se desean utilizar, se deben volcar en un archivo de configuración donde se declaran los servicios de tipo SOAP. Dentro de cxf.xml existen todos los servicios que expone Integra a sus clientes, en la declaración de cada servicio es donde se debe ingresar el tipo de interceptor (entrada o salida) que se quiere adjuntar y la ruta de su clase. Para la captura #12 se observa los interceptors nombrados previamente que contiene el servicio Trámite.

```
<jaxws:endpoint id="tramiteEndpoint"
                implementor="#tramite"
                address="/tramite" >
              <jaxws:inInterceptors>
                    <bean class="ar.com.osde.wf.interceptor.HeaderInInterceptor" />
              </jaxws:inInterceptors>
              <jaxws:outInterceptors>
                    <bean class="ar.com.osde.wf.interceptor.HeaderOutInterceptor" />
              </jaxws:outInterceptors>
              <jaxws:outFaultInterceptors>
                  <bean class="ar.com.osde.wf.interceptor.CustomSoapFaultOutInterceptor" />
              </jaxws:outFaultInterceptors>
</jaxws:endpoint>
```

Figura 12 Archivo cxf.xml con la declaración de los Interceptors. Fuente: Elaboración propia (basada en la práctica)

2.5 Circuito de eventos

En esta etapa del proyecto se requiere refactorizar todo el circuito de eventos, que utiliza Integra para la comunicación con el clúster del Solr de Referencias. Este circuito de eventos es utilizado en el momento que se genera una acción como, por ejemplo, la creación o escritura sobre un trámite. Integra, además, de ingresar la información en la base de datos, genera a la par un evento para ser consumido por el Index Service. El circuito posee una prioridad alta, ya que la información que se encuentre en la base de datos debe quedar replicada dentro de la herramienta Solr, en el caso de una consulta.

Funcionamiento del circuito:

Para describir el funcionamiento del circuito de eventos en Integra, es necesario aclarar que existe una totalidad de seis clases de eventos que se producen en su lógica. Las acciones producidas por un usuario sobre la aplicación pueden desencadenar alguno de estos eventos, pudiendo ser generados tanto por medio de la interfaz gráfica como por el uso de algún servicio. Además, Integra tiene un rol de productor de eventos, encargándose de encapsular los mensajes para enviarlos hacia un gestor de colas MQ. Para conectarse al MQ es necesario contar con unas credenciales, puerto y un canal que permita establecer la conexión, donde el canal establece un tope máximo de conexiones concurrentes y un protocolo de transporte. El gestor de colas posee

unos contenedores denominados tópicos, los cuales son diferenciados por un nombre clave y se los utiliza con el fin de enviar eventos de un mismo tipo.

Un tópico se encarga de publicar mensajes o eventos a una lista de suscriptores que se encuentren interesados de este modo, el tópico replica los mensajes sólo a los suscriptores que cumplen con ciertas especificaciones del mensaje. El suscriptor configura un selector para saber qué mensajes debe interceptar y, por consiguiente, los deriva a una cola específica. Los selectores que utiliza el suscriptor de Integra son el “namespace” y el “source”, el primero sirve como identificador del evento producido y en el caso del source hace referencia al ambiente y aplicación que lo genera. Una vez que los eventos llegan a la cola son consumidos por la aplicación Index Service, la cual desencapsula los mensajes del evento y los envía al clúster del Solr para que sean indexados. En las circunstancias que la aplicación Index Service no pueda consumir dicho evento, los mensajes serán enviados a una cola de backout. La cola de backuot es donde se almacenan todos los mensajes rechazados, que contengan algún tipo de error, con el fin de analizarlos posteriormente.

Para una mejor comprensión del circuito de eventos descripto, se adjunta el siguiente diagrama de flujo.

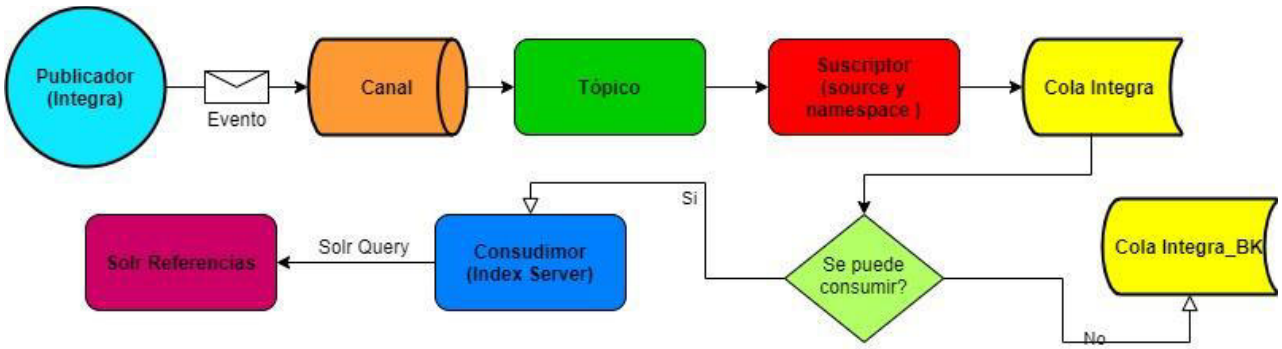


Figura 13 Circuito de eventos de Integra. Fuente: Elaboración propia (basada en la práctica)

A continuación, se detallarán los eventos mencionados previamente:

- Trámite agregado o modificado
- Trámite eliminado
- Observación agregada
- Observación modificada

- Trámite archivado
- Trámite desarchivado

Configuración:

Inicialmente para la configuración del circuito de eventos en los diferentes ambientes, se necesitó definir los componentes del circuito como el suscriptor, el tópico, el canal y la cola que se va a consumir. Para llevar a cabo estas tareas, se utilizó la herramienta MQ Explorer de IBM donde se creó el tópico a utilizar llamado Eventos. Para la creación del suscriptor, se definió el tópico al cual pertenece la cola que transportará los eventos y las propiedades del mensaje (source y namespace) que definirá los eventos que interceptará el suscriptor. También, fue necesario definir el canal por el cual se va a comunicar Integra con el tópico y la cola que va a consumir el Index Service al final del circuito.

Para la configuración de la conexión entre Integra y el gestor de colas MQ, se utilizó el archivo de configuración Standalone.xml como base central. JBoss dispone de un subsistema llamado “resource-adapters” para establecer dicha conexión, donde se configuran todos los datos necesarios para crearla. Dentro de la configuración para establecer la conexión, se destacan principalmente el nombre del host donde se encuentra el MQ, el canal y puerto disponible para crear la comunicación. Además, se establece un usuario, la contraseña y el nombre del gestor al cual se desea acceder.

En la siguiente captura se encuentra la configuración del subsistema que usa JBoss para la conexión hacia el MQ.

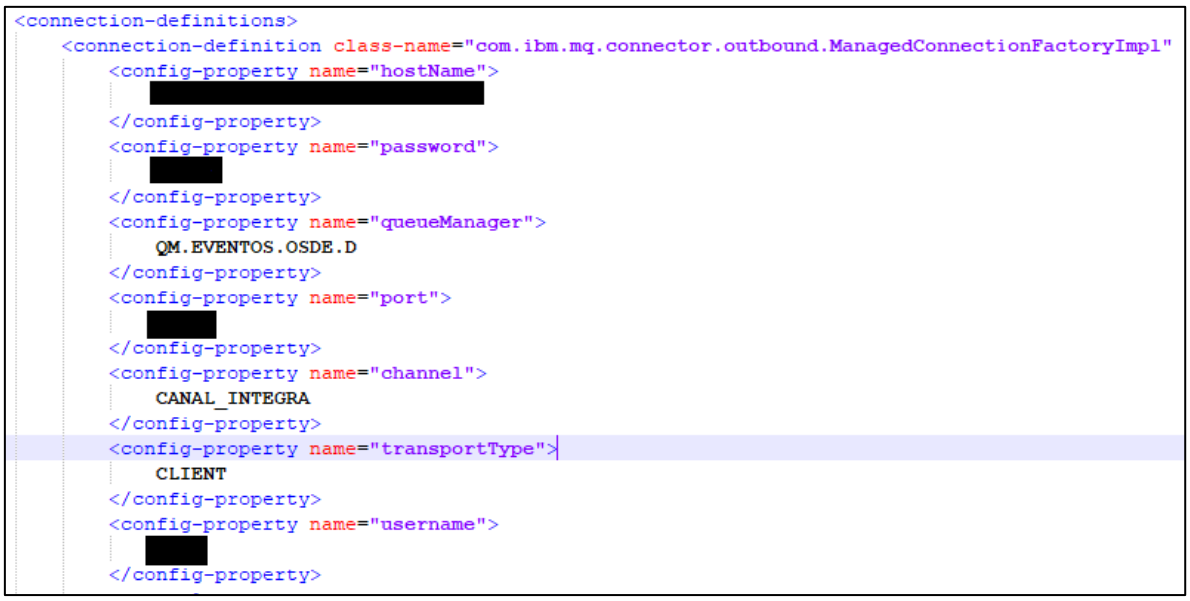


Figura 14 Configuración de conexión al MQ. Fuente: Elaboración propia (basada en la práctica)

Dentro del mismo subsistema del Standalone, existe una sección donde se declara el tópico que utilizará Integra para enviarle los eventos. Enunciando el nombre del tópico y el gestor de colas que lo contiene. En la siguiente imagen se observa la estructura de dicha configuración:

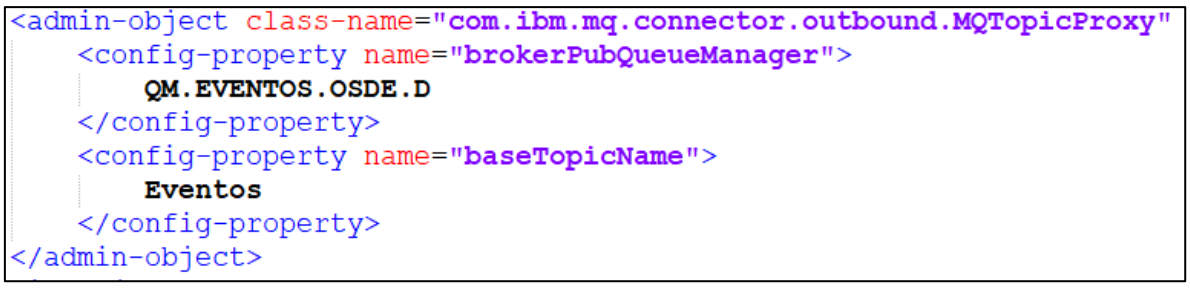


Figura 15 Configuración para publicar eventos. Fuente: Elaboración propia (basada en la práctica)

Cambios dentro de Integra:

Con lo que respecta a los cambios dentro del código fuente de Integra, se tuvo que migrar toda la lógica relacionada al uso del framework del componente EDA. Este componente quedó obsoleto por la nueva configuración provista por Spring Framework.

Para dicha configuración se necesita una clase XML, donde se declaró por medio de utilización de beans toda la configuración de conexión para el uso de MQ. Inicialmente, se crea un bean llamado “connectionFactory” con el propósito

de establecer los parámetros de conexión y otro “jmsTopicDestination” en el que se declara el tópico de destino. Ambos beans se encuentran relacionados con el parámetro jndi-name declarado en el Standalone, donde obtienen todos los valores declarados en el mismo. Por otro lado, son parte de un bean padre que funciona como un template (plantilla) de la configuración. El template declarado en el bean jmsTopicTemplate es utilizado por todas las clases de la aplicación que desean hacer uso del envío de eventos.

```
    <bean id="connectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean" lazy-init="false">
        <property name="jndiName" value="java:/jboss/jms/wmq/connectionFactoryWMQDESA" />
    </bean>

    <bean id="jmsTopicTemplate" class="org.springframework.jms.core.JmsTemplate">
        <constructor-arg ref="connectionFactory" />
        <property name="pubSubDomain" value="true"/>
        <property name="defaultDestination" ref="jmsTopicDestination"/>
    </bean>

    <bean id="jmsTopicDestination" class="org.springframework.jndi.JndiObjectFactoryBean" lazy-init="true">
        <property name="jndiName" value="java:/jboss/jms/wmq/topic/TopicEvento" />
    </bean>

</beans>
```

Figura 16 Beans para la conexión al MQ y producción de eventos. Fuente: Elaboración propia (basada en la práctica)

Para los eventos relacionados a las observaciones (creación de un trámite, eliminación o alteración de uno), se utiliza la clase IndexServiceImpl. Esta clase inicializa el objeto JmsTemplate por inyección de dependencias, para hacer el envío de eventos que es utilizado posteriormente por el método sendEvent de la clase. Este método es de tipo genérico, ya que es llamado por los demás métodos relacionados a los eventos mencionados. En cuanto a su consumo, utiliza como parámetros un objeto de tipo FrameworkEntity, que es la entidad que utiliza el destinatario para interpretar la información y el segundo parámetro es el nombre clave del evento a utilizar. Dentro de la implementación de SendEvent, se inicializa un objeto Event donde se ingresarán mediante métodos de tipo Set toda la información necesaria para enviar de forma correcta el evento. Dentro de la información, que se adjunta en el evento, se encuentran las siguientes:

- El namespace, que se obtiene de una constante de la aplicación.
- El nombre del evento.
- El source, que es extraído de las propiedades del entorno

- Una constante de tiempo, que hace referencia al tiempo de vida del evento.
- El nombre de la clase a enviar.
- La entidad relacionada al mensaje en un formato de datos adecuado para su transmisión

Cabe destacar que para hacer uso del objeto Event, para crear el formato del mensaje, es necesario importar la dependencia osde-framework-event dentro del proyecto Integra.

Una vez que se agrega toda la información necesaria en el evento creado, se realiza un proceso de transformación de datos en el evento para su envío. La función marshal, que es utilizada tanto para transformar el evento como a la entidad de uso, utiliza dentro su flujo la librería JAXB para realizar la transformación.

Una vez generado el evento, el método sendEvent llama a la función Send del objeto jmsTemplate referenciado en la clase. Este tiene como función crear el mensaje del evento y después enviarlo al tópico que se definió en el template. Además, el método sendEvent usa la sesión de conexión y crea un objeto de tipo TextMessage, que utiliza el objeto evento transformado previamente. También, al objeto TextMessage se le adjuntan tres propiedades donde se vuelven a destacar el namespace del evento, el source y, por último, el nombre que identifica al evento.

En la siguiente captura se puede observar la lógica del método sendEvent detallada.

```
private void sendEvent(FrameworkEntity eventObject, final String eventName) {
    if (eventObject == null){
        return;
    }

    if (LOGGER_INDEX_SOLR_MANAGER.isDebugEnabled()){
        LOGGER_INDEX_SOLR_MANAGER.debug("Enviando un evento de actualizacion del indice Solr: "+EDAEventConstants.SOLR_EVENT_NAMESPACE+"."+eventName);
    }

    String sourceName = PropertyPlaceholderConfigurerContainer
            .getCustomPropertyPlaceholderConfigurer().getPropertyValue("sourceName");

    Event event = new Event();

    event.setNameSpace(EDAEventConstants.SOLR_EVENT_NAMESPACE);
    event.setEventName(eventName);
    event.setSource(sourceName);
    event.setTtl(EDAEventConstants.EVENT_TTL);
    event.setDuration(EDAEventConstants.EVENT_DURATION);
    event.setClassname(eventObject.getClass().getName());

    try {

        event.setPayload(marshal(eventObject));

        String eventoXml = marshal(event);

        this.jmsTemplate.send(new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                TextMessage createTextMessage = session.createTextMessage(eventoXml);
                createTextMessage.setStringProperty("namespace", event.getNameSpace());
                createTextMessage.setStringProperty("eventname", event.getEventName());
                createTextMessage.setStringProperty("source", event.getSource());
                return createTextMessage;
            }
        });
```

Figura 17 Método para enviar eventos. Fuente: Elaboración propia (basada en la práctica)

El mismo caso de uso de eventos de la clase IndexServiceImpl, es utilizado para las acciones de archivar y desarchivar trámites en la clase SimpleArchivingManager. Donde también, se inyecta el template de conexión jmsTopicTemplate. Dentro de los métodos de archivar y desarchivar se utiliza el mismo método, pero con la salvedad que en su llamado se ingresa el nombre propio de la función que realizan (archivar y desarchivar).

Consumo del evento:

Para el consumo de los eventos en la cola de Integra, se utiliza la aplicación externa Index Service que se encarga de obtener los mensajes, transformarlos y enviarlos en formato SolrQuery al Solr de Referencias. El Index Service también necesita de una configuración dentro de su Standalone, con los mismos parámetros de conexión del MQ que figuran en la captura #14. Además, necesita la declaración de un objeto administrador, que contiene en principio el nombre de la cola que va a consumir y el nombre de referencia del MQ, y que contiene dicha cola. A continuación, se representa la configuración descripta que contiene el Standalone para el Index Service.

```
<admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
    <config-property name="baseQueueName">
        DEST_INTEGRA_MIGRACION
    </config-property>
    <config-property name="baseQueueManagerName">
        QM.EVENTOS.OSDE.D
    </config-property>
</admin-object>
```

Figura 18 Configuración para consumir eventos de la cola. Fuente: Elaboración propia (basada en la práctica)

Para hacer referencia a la configuración proveniente del Standalone, se establecieron beans derivados del framework Spring utilizando la misma estructura que se usó en Integra para la conexión al gestor MQ. También, se declara un bean llamando “jmsQueueDestination”, que representa un valor de referencia hacia el nombre de la cola a consumir. Por último, otro bean que apunta a la clase ConsumerSpring, que será la encargada de trabajar los eventos consumidos. Los tres beans comentados anteriormente pertenecen al contenedor listenerContainer, que se ocupa de recolectar todos los mensajes pertenecientes a la cola y entregarlos a la clase ConsumerSpring para poder trabajarlos. En la siguiente captura se pueden identificar los beans utilizados en el Index Service.

```
<bean id="listenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer">
                <property name="concurrentConsumers" value="1"/>
                <property name="connectionFactory" ref="connectionFactory" />
                <property name="destination" ref="jmsQueueDestination" />
                <property name="messageListener" ref="messageListener" />
</bean>

<!--  parte MDB -->
<!-- this is the Message Driven POJO (MDP) -->
<bean id="messageListener" class="ar.com.osde.wf.eda.events.consumer.ConsumerSpring" >
    <property name="tramiteAgregadoOModificado" ref="tramiteAgregadoOModificado"/>
    <property name="tramiteEliminado" ref="tramiteEliminado"/>
    <property name="observacionAgregada" ref="observacionAgregada"/>
    <property name="observacionModificada" ref="observacionModificada"/>
    <property name="tramitesDesarchivados" ref="tramitesDesarchivados"/>
    <property name="tramitesArchivados" ref="tramitesArchivados"/>
</bean>

<bean id="connectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean" lazy-init="false">
    <property name="jndiName" value="java:/jboss/jms/wmq/connectionFactoryWMQDESA" />
</bean>

<!-- queue declara en ds en el jboss -->
<bean id="jmsQueueDestination" class="org.springframework.jndi.JndiObjectFactoryBean" lazy-init="false">
    <property name="jndiName" value="java:/jms/wmqQueueDEST_INTEGRA_MIGRACION" />
</bean>
```

Figura 19 Beans para el consumo de eventos. Fuente: Elaboración propia (basada en la práctica)

Dentro de la clase ConsumerSpring que implementa la interfaz MessageListener de Javax, se declara el método onMessage para la recepción de mensajes. Este método, se encarga de interceptar todos los mensajes de la cola en formato TextMessage, luego los transforma con ayuda de la librería JAXB y el método unmarshalTextMessage a un formato de tipo Event. Con el evento generado, se extrae el nombre que lo identifica y la clase entidad del mensaje (Payload) que es almacenada en un objeto. Para identificar a qué clase se va a derivar el mensaje recibido, se utiliza un switch que clasifica al evento según su nombre. Dentro de cada caso llama al método processEvent con un objeto de tipo trámite u observación. Existen distintas clases que trabajan los mensajes interceptados, estas se encargan de extraer los datos del objeto (trámite u observación) formado y los agrega a una query que se envía al Solr, con el fin de indexar los datos.

```
@Override
public void onMessage(Message message) {

    if (message instanceof TextMessage) {
        TextMessage textMessage = (TextMessage) message;

        try {

            Event textMessageEvent = unmarshalTextMessage(textMessage);
            String textMessageEventName = textMessageEvent.getEventName();
            String textMessageEventPayload = textMessageEvent.getPayload();

            //Logueo el consumo del evento
            log.info("***************EVENTO CONSUMIDO*******************");
            log.info("Nombre del evento: " + textMessageEventName);
            log.info("Fecha del evento:" + new Date());
            log.info("Datos del evento: " + textMessageEventPayload.toString());
            log.info("***************EVENTO CONSUMIDO*******************");

            Object objectEvent = unmarshalEvent(Class.forName(textMessageEvent.getClassname()), textMessageEventPayload);

            Tramite tramite;
            switch(textMessageEventName)
            {
                //Observacion Agregada
                case EDAEventConstants.SOLR_OBSERVACION_AGREGADA_EVENTNAME :
                    ObservacionTramiteWrapper observacion = (ObservacionTramiteWrapper) objectEvent;
                    this.observacionAgregada.processEvent(observacion);
                    break;
                //Tramite Agregado O Modificado
                case EDAEventConstants.SOLR_TRAMITE_AGREGADO_O_MODIFICADO_EVENTNAME :
                    tramite = (Tramite) objectEvent;
                    this.tramiteAgregadoOModificado.processEvent(tramite);
                    break;
```

Figura 20 Método de recepción de eventos. Fuente: Elaboración propia (basada en la práctica)

Para el método onMessage, se agregó un nuevo Handler llamado “Index_Service” con el objetivo de registrar todos los eventos consumidos. Como se observa en la captura anterior, se loguea la información que contiene el evento como ser su nombre clave, la fecha en formato completo (día/mes/año) del momento en que se consume y los datos que transporta en un formato XML.

2.6 Integración continua

En esta sección, se describe el desarrollo para el cumplimiento del objetivo enfocado en la aplicación de Integración continua al proyecto IntegraBPM, mediante la aplicación de la metodología DevOps. La aplicación de DevOps, es posible con el uso del módulo CI/CD de la herramienta GitLab para la aplicación de metodologías continuas al proyecto.

Conceptos

- **Pipeline**: es un grupo de pasos relacionados, en este caso agrupa un conjunto de Jobs.
- **Job**: son trabajos que pueden estar regidos mediante condiciones, contienen un script para ejecutar alguna tarea y son ilimitados
- **Enviroments**: es el entorno en el cual se va a desplegar la aplicación, es decir, los ambientes.
- **CI/CD variables**: Permite almacenar valores que se desean reutilizar, para no definirlos en el archivo gitlab-ci.yml.
- **Runners**: Es el proceso encargado de leer el gitlab-ci.yml y correr los scripts que pertenecen al Job.

Dentro del proyecto Integra se trabaja con dos tipos de versiones, una es la versión Release y la otra es Snapshot. La diferencia entre ambas versiones, es que la Snapshot es una versión inestable que se encuentra en proceso de desarrollo, en cambio, la Release es una versión estable próxima a implementarse. También, es necesario aclarar el uso y propósito de las ramas que se utilizan dentro del proyecto Integra:

- **Tags**: es una rama que tiene un estado inalterable, se utilizan en los casos en donde no se requiere agregar más modificaciones o versiones desplegadas, es decir, una rama estable de tipo Release.

- **Master**: es la rama principal que contiene el siguiente número de versión a desplegar y los últimos cambios que se encuentran en el ambiente productivo. De esta rama se crea Develop.
- **Develop**: es la rama que se encuentra en un estado de desarrollo, posee el contenido próximo a desplegar y es una versión de tipo Snapshot.

Para lo que es el circuito DevOps, el desarrollo se divide en dos proyectos relacionados ubicados en la herramienta de Gitlab, uno de ellos se encuentra en el repositorio de Integra y, por otro lado, existe el proyecto Deploy de Integra. El proyecto Deploy cumple varias funciones, como establecer en qué ambiente se debe realizar el despliegue, la conexión hacia el servidor JBoss y las pruebas para determinar que se realizó bien la implementación.

Proyecto Integra

Para la configuración de DevOps dentro del proyecto Integra, es necesario una serie de scripts, la declaración de las properties a utilizar para cada ambiente y el archivo gitlab-ci.yml. En primer lugar, se crearon tres scripts dentro del proyecto en un nuevo directorio llamado pipeline que serán utilizados posteriormente, entre ellos se destacan:

- **Build**: este script contiene el comando “package” de Maven, que genera el paquete War del proyecto.
- **Deploy_nexus**: el script contiene el comando Deploy de Maven, para subir el proyecto al repositorio Nexus en donde se van a encontrar todas las versiones generadas por Integra.
- **Request_deploy**: por último, se encuentra este último script que genera un Curl (comando para la transferencia de archivos) para enviar al proyecto Deploy las variables necesarias para el despliegue. El Curl está constituido con la URL del proyecto Deploy, las variables para el ambiente a desplegar, versión, fase de despliegue y el artefacto del proyecto (Integra).

Dentro de la estructura del proyecto Integra existe el directorio llamado config, que cuenta con un listado de carpetas con el nombre del ambiente de despliegue (desarrollo, testing, producción, etc) al que pertenecen. Cada subdirectorio posee un archivo con todas las properties de Integra con sus respectivos valores propias del ambiente. Este archivo es denominado como "configuracion.properties", el documento está compuesto por una línea con el nombre de la property del lado izquierdo y su valor del lado derecho separado por el carácter igual.

Como se mencionó antes, en Integra se utiliza el archivo gitlab-ci.yml donde su funcionalidad se centra en configurar el comportamiento de integración continua del proyecto en Gitlab. La configuración sirve para describir la estructura y orden de los Pipelines (conjunto de Jobs o tareas), además, se definen las acciones que van a ejecutar los runners y las diferentes decisiones a tomar antes una variedad de estados. Inicialmente el archivo utiliza una imagen de Docker para lo que es la compilación y creación del War de Integra, esta imagen contiene una versión de JDK y Maven similares a las establecidas en los objetivos del proyecto PPS. También, se declaran las fases por las que el proyecto de Integra puede atravesar en Gitlab, también referenciados como Stages (fases en inglés) y, por dentro, se encuentran los Jobs de despliegue de Integra para los distintos ambientes. Además, los Stages se hallan de manera ordenada mediante el criterio de ejecución, a continuación, se encuentra una breve descripción de los mismos:

- **build**: es la etapa en donde se lleva a cabo la compilación del proyecto Integra con el uso de las tecnologías declaradas en la imagen Docker del archivo gitlab-ci.yml. El Stage tiene una expiración de un día en Gitlab y se ejecuta automáticamente en todas las ramas.
- **deploy_nexus**: la etapa en la que se despliega la versión Snapshot, en el repositorio Nexus, es ejecutada tanto por la rama Master y Develop. El repositorio puede contener varias versiones del War de una misma versión Snapshot.
- **deploy_to_desa**: utiliza el script que se conecta por medio de un Curl al proyecto Deploy de Gitlab. Sus Jobs declaran el ambiente a deployar

(desa), versión, nombre del proyecto a desplegar y la fase que en este caso siempre es igual a 1. Las ramas que pueden ejecutar los Jobs son Develop y los Tags.

- **deploy_to_test**: Existen dos Jobs que referencian al Stage de testing evolutivo, porque este ambiente contiene dos instancias. El Job utiliza las mismas variables que el Stage de desarrollo, pero referenciando a testing y a cada Job haciendo referencia a una fase distinta.
- **deploy_to_test_gen**: Contiene dos Jobs (uno por instancia) para el ambiente de testing genérico y la misma lógica que los anteriores despliegues.
- **deploy_to_prepro**: Abarca dos Jobs (uno por instancia) para el ambiente de preproducción y una lógica idéntica a los anteriores despliegues.
- **deploy_to_prod**: Los Jobs tienen la misma lógica que los anteriores en cuanto a los scripts que usa, pero con la diferencia que utiliza cuatro Jobs para satisfacer a cada instancia y apunta al ambiente de producción. Sin embargo, cabe aclarar que solo las ramas de tipo Tag pueden ejecutar los Jobs del Stage de producción.
- **promotion_to_release**: Es utilizado solo por la rama Develop, su función es mergear (fusionar) con la rama Master para crear el Tag de la versión a desplegar y publicarlo en el repositorio Gitlab. Por último, sube un número en la versión y lo referencia como Snapshot, preparándolo para el siguiente desarrollo, además, elimina la rama Develop al final.
- **deploy_tag_nexus**: Es ejecutada solamente por las ramas de tipo Tag, realiza el despliegue de la versión Release al repositorio de Nexus. Es la versión que se va encontrar desplegada en producción.

Para los Jobs que se encuentran relacionados a los Stages de despliegues, tiene un modo de ejecución de tipo manual a diferencia de los Stages, que se encargan de compilar el proyecto o subirlo al repositorio Nexus (estos son automáticos). En la captura siguiente se puede observar el primer Job del Stage perteneciente al environment (ambiente) de producción, además, en este Job se pueden identificar las variables que utiliza. En el caso de las variables Versión y Artefacto son obtenidas por un comando Maven, también se puede identificar la fase a la cual pertenece y el ambiente que referencia a producción. Al final se

utiliza una etiqueta denominada script, que contiene la ubicación del ejecutable request_deploy, adjuntando en el llamado las variables declaradas para armar el Curl con destino al proyecto Deploy.

```
deploy_to_prod_fase1:
  stage: deploy_to_prod
  environment: prod
  only:
     - tags
  when: manual
  before_script:
    - export "VERSION=$(mvn help:evaluate -Dexpression=project.version -q -DforceStdout)"
    - export "ARTEFACTO=$(mvn help:evaluate -Dexpression=project.artifactId -q -DforceStdout)"
    - export "AMBIENTE=prod"
    - export "TRIGGER_TOKEN=$CI_JOB_TOKEN"
    - export "FASE=fase1"

  script:
    - sh ./pipeline/request_deploy.sh $VERSION $ARTEFACTO $AMBIENTE $TRIGGER_TOKEN $FASE
```

Figura 21 Modelo de un Job del Stage de producción. Fuente: Elaboración propia (basada en la práctica)

En la siguiente captura, se observa la creación de un pipeline con un número clave que lo diferencia de los otros pipelines, este se genera luego de la ejecución del Job Promotion-To-Release proveniente de una rama Develop. El pipeline ubicado en la parte superior pertenece a una rama de tipo Tag, también contiene en el centro unos checks circulares de las fases ejecutadas satisfactoriamente. El primer check en verde pertenece al Stage Build para compilar y generar el War de la rama, por otro lado, los últimos checks ejecutados correctamente son para los despliegues para cada instancia de Integra en producción. También existen las fases en gris que todavía no han sido ejecutadas, que referencian a los ambientes de desarrollo y a los dos de testing para nivelar con la misma versión de Integra que se implementó en el ambiente productivo.

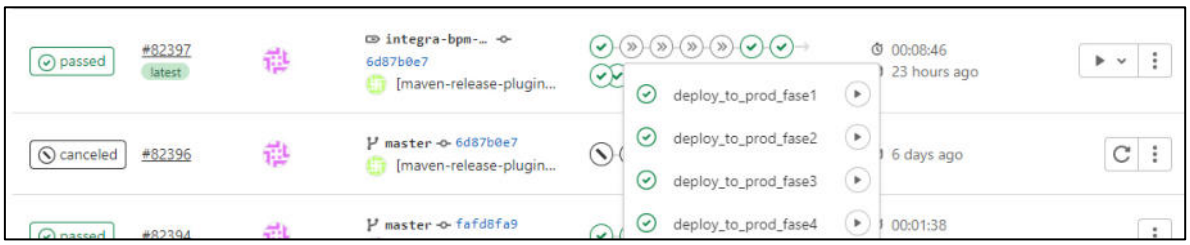


Figura 22 Pipelines del proyecto Integra. Fuente: Elaboración propia (basada en la práctica)

Proyecto Deploy

El proyecto Deploy para la integración continua también utiliza el archivo gitlab-ci.yml, para este caso se usa una imagen de Docker de versión 7.2 y un solo Stage llamado Deploy en la declaración de todos los Jobs. Existe un Job para cada ambiente, que utiliza una etiqueta de tipo regla y que debe coincidir con el nombre del ambiente, también tienen un modo de ejecución manual y una expiración de un día. Dentro de cada Job se hace referencia al script deploy.sh, en conjunto con las variables recibidas por el Curl de Integra (ambiente, versión y artefacto) y otras de tipo CI/CD, las cuales son declaradas dentro del proyecto para cada environment. Las variables de tipo CI/CD son las más sensibles en cuanto a seguridad, son las credenciales de acceso y ubicación de una bóveda para traer otros datos críticos.

Además, del archivo gitlab-ci.yml comentado previamente existen otros scripts importantes para el despliegue de Integra que serán mencionados y descritos a continuación en función al orden que son ejecutados.

Deploy: este script es empleado por todos los Jobs que contiene el archivo gitlab-ci, recibiendo todas las variables expuestas en el proyecto Integra (artefacto, ambiente, versión, etc). Mediante el armado de un nuevo Curl, hace una petición al repositorio Nexus para poder extraer el War de Integra y un comprimido con todas las properties perteneciente al ambiente a deployar. Además, existen dentro de la lógica dos comandos de linux que sirven para eliminar los espacios en blanco y líneas comentadas del archivo que contiene todas las properties. Estos comandos sirven para evitar cualquier tipo de error a futuro o datos innecesarios en el Standalone. También, durante el flujo de trabajo el archivo deploy.sh se encarga de ejecutar los scripts obtener_jboss_secrets, replace_secrets, generar_cli y deploy_multi_fase.

Ambientes: es un listado con los diferentes ambientes y sus respectivas instancias, declarando su dirección ip o Hostname y el puerto que usa la interfaz cli para conectarse al JBoss.

Obtener_JBoss_Secrets: es un script de python que usa las variables de su llamado, como el ambiente a desplegar, dirección y credenciales para acceder al Vault (bóveda de datos sensibles). La secuencia de comandos tiene como propósito recuperar las credenciales de acceso para el JBoss de los ambientes a desplegar y almacenarlas mediante variables en el servidor. Para realizar dicho propósito, se extrae mediante una función la dirección de los nodos pertenecientes a la variable ambiente, una vez obtenidos los utiliza para obtener las credenciales del JBoss por medio del script Vault_Utils.

Vault_Utils: utiliza el usuario y la contraseña de sus variables, para obtener mediante un request un token de acceso a la bóveda. Luego, con el token obtenido en el paso anterior y con la dirección del Vault trae las secrets (credenciales de acceso al JBoss). Para el almacenamiento de estos datos de acceso, Integra hace uso de la herramienta Vault de Hashicorp para almacenamiento de forma segura de datos secretos y sensibles para entornos dinámicos en la nube.

Replace_Secrets: este script se ejecuta posteriormente a la ejecución de Obtener_JBoss_Secrets, con las credenciales del Vault y el archivo descomprimido de las properties de Integra. Se encarga de recorrer todas las properties extraídas del repositorio Nexus, buscando identificar si alguna de ellas requiere obtener su valor proveniente del Vault. En el caso de existir propiedades que cumplan con dichas condiciones, se reemplazan sus valores críticos y se adjuntan a un nuevo archivo de properties separadas de las demás. Las properties con datos sensibles son extraídas siempre del mismo Vault, donde se obtienen las credenciales de acceso al JBoss. Estas medidas son parte de un protocolo de seguridad, para evitar definir datos sensibles sobre texto plano en el archivo “configuration.properties” de Integra.

Generar_Cli: su función es generar el archivo de extensión Cli que se va a ejecutar en el servidor JBoss del ambiente, para insertar las properties declaradas en el archivo config de Integra y también desplegar la aplicación en el servidor. Primero recorre el archivo extraído del Nexus con todas las properties

que se requieren introducir al ambiente, para eso se extrae la clave y el valor por separado de cada property. Una vez extraída la clave se genera una sentencia de tipo If para consultar si ya se encuentra la property en el Standalone, en caso afirmativo se elimina dicha property del Standalone. Para el siguiente paso, se genera otra sentencia con el uso de clave y valor de la property dentro del Standalone. En segundo lugar, se crea otro archivo cli denominado deploy.cli, que posee la sintaxis necesaria para deployar en el JBoss la aplicación, y para eso es necesario el uso de la variable del Artefacto.

Deploy_Multi_Fase: en este script se recorre el archivo ambientes.properties y se obtienen todos los nodos pertenecientes al ambiente a deployar. Además, se utiliza una función principal para la ejecución de archivos de tipo cli llamada “ejecutar_cli”. La función utiliza como parámetros el destino (host) como el entorno de ejecución y el nombre del archivo de extensión cli a ejecutar. Después, obtiene las variables de usuario y contraseña guardadas en el sistema, para utilizarlas en conjunto con el host y armar el comando que se ejecutará en la consola del JBoss. Este comando de tipo cli está compuesto por las credenciales de acceso al servidor, el host del ambiente a deployar y el nombre del archivo que se ingresa por parámetro.

Después de determinar el ambiente en el cual se va a implementar, la lógica se divide en tres componentes. Uno de los componentes es cuando el ambiente indicado es el de desarrollo, que se diferencia por tener una sola instancia a deployar. Utilizando la función ejecutar_cli con sus argumentos, el único host y el archivo deploy.cli para desplegar el War de Integra con sus properties correspondientes.

Para el segundo componente, se encuentran los demás ambientes bajos y estos utilizan dos fases de despliegue. Para el caso de la fase uno, se obtiene el primer nodo del listado de ambientes y realiza una implementación balanceada. Este deploy balanceado consiste en primero desbalancear la instancia con la utilización de Detener_Health.cli y después implementar el War de Integra, ambos con el uso de la función ejecutar_cli. Para la fase dos se balancea la instancia inicial con el Iniciar_Health.cli y luego se realiza un despliegue de forma balanceada como en la fase uno. Al finalizar la última fase todas las instancias deben quedar balanceadas para su utilización.

Por último, el tercer componente es a nivel producción y contiene un total de cuatro fases en su flujo de trabajo. En cada fase, se valida que no haya alguna instancia balanceada de una fase anterior y, en el caso de haberla, se realiza un balanceo. Una vez hecha la validación de la fase anterior se procede a desbalancear la instancia actual e implementar el War de integra con el uso de ejecutar_cli. Como particularidad, la última fase (número 4) después de desplegar la aplicación realiza el balanceo de la instancia, ya que no existe una fase siguiente que lo haga.

Detener_Health y Iniciar_Health: Son dos archivos de extensión Cli que sirven para modificar el estado de una property para cualquier instancia de Integra. Pueden modificar el valor de la property healthCheck a un estado falso/verdadero, para desbalancear o balancear la instancia. Estos archivos son totalmente necesarios para que, en el momento del despliegue de la aplicación, la instancia no se encuentre utilizada por algún usuario y no altere su flujo de trabajo.

En la figura #21 se encuentra una vista similar a los pipelines generados en el proyecto Integra, pero con la diferencia que existe un solo Stage. En relación con la imagen anterior, se puede afirmar que cuando se ejecuta un Job relacionado al ambiente de producción, se crea dentro del proyecto Deploy un pipeline como en el ejemplo de la captura. La ejecución de este Job iniciaría la implementación productiva de la aplicación Integra en conjunto con las properties del ambiente, la instancia de despliegue dependería de la fase ejecutada inicialmente. En el caso, que el segundo check se encuentre de color verde representaría que la implementación se realizó de manera satisfactoria, en caso contrario (una X de color rojo) podría haber ocurrido algún error en alguna de las etapas descriptas.

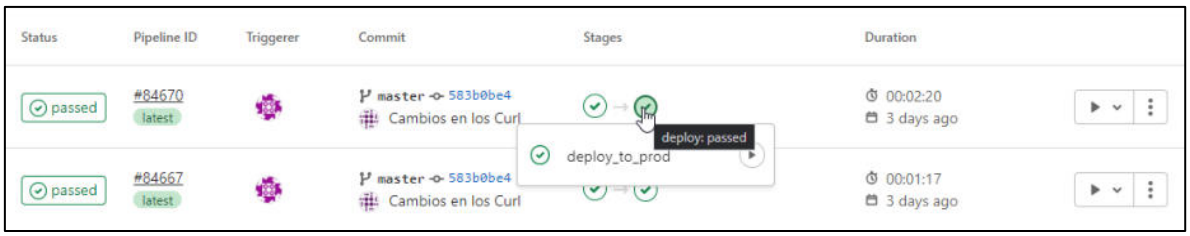


Figura 23 Pipelines del proyecto Deploy. Fuente: Elaboración propia (basada en la práctica)

2.7 Apache Solr

Para la migración de la herramienta Solr desde la versión 4.7.2 a la 8.9, tanto el Solr de Referencias como el de Filtros requieren realizar cambios similares dentro del código. Para el Solr de Filtros los cambios se realizan dentro del código fuente de Integra y en el caso del de Referencias se llevan a cabo en la aplicación Index Service. También, se debe descargar el proyecto de Solr que coincida con la versión requerida desde la página oficial de Apache, por este motivo este proyecto debe ser nivelado con las configuraciones propias de Integra en la versión anterior.

Configuraciones de las instancias de Solr

Para la parte de desarrollo, las nuevas instancias de Solr de los servicios de Filtros y Referencias, se apoyó en la página oficial de Apache. A través de esta página, se procedió a descargar los archivos fuente del Solr y del Zookeeper, que provienen con una configuración estándar y que debe ser modificada de acuerdo al servicio que se requiere brindar. Ambos servicios (Filtros y Referencia), conllevan una configuración similar en cuanto a la estructura de sus scripts de arranque o librerías de uso.

En principio, se configuró el Zookeeper que es el encargado de gestionar las configuraciones de todos los nodos que componen al clúster. Dentro del directorio raíz de cada instancia de Zookeeper, es necesario crear una carpeta con el número que identifica a la instancia. Esa carpeta creada trabaja como un id de registro para que pueda ser diferenciado de las demás instancias. Por otra parte, se configuraron tres archivos de configuración importantes antes de inicializar el Zookeeper, entre ellos se encuentran:

zoo.cfg: es el archivo principal de configuración, donde se declararon las diferentes variables definen el comportamiento de la instancia. Las utilizadas en esta ocasión fueron:

- **tickTime**: define el tiempo de espera de la sesión.

- **initLimit**: tiempo de espera para comunicarse con los demás nodos de Zookeeper.
- **syncLimit**: cantidad máxima de sincronización de los nodos.
- **dataDir**: ubicación del directorio que tiene el índice o id del Zookeeper.
- **clientPort**: se define el puerto cliente de conexión, que utiliza Integra y los nodos de Solr para conectarse.
- **admin.serverPort:** es el puerto de administración.
- **server.1**: se define la primera instancia del Zookeeper, y con la misma sintaxis más una modificación del número, se definen las instancias restantes del clúster.
- **4lw.commands.whitelist=mntr,conf,ruok**: permite habilitar una vista dentro de la interfaz de Solr llamada zkStatus.

zookeeper-env.sh: se utiliza para la definición de variables de entorno, para el momento de inicializar la instancia de Zookeeper. Las variables que se registraron fueron las de la memoria reservada del servidor, políticas de garbage collector, la ubicación de la JDK y de los archivos de logs a utilizar.

log4j.properties: es un archivo propio de la herramienta Log4j, para lo que es el registro de logs del Zookeeper. Dentro de este archivo, se registró el nombre del log principal de logueo, el nivel, capacidad máxima y cantidad de archivos de log. Además, permite la posibilidad de agregar más Appenders a la configuración en el caso de requerir loguear en otros archivos de log más registros.

La configuración requerida en las instancias de Solr, se centra en el uso de variables dentro del script de arranque “solr.in.sh”. Dentro del script, se utilizó la variable de entorno que referencia a la ubicación de la JDK, los volúmenes de memoria Ram de la instancia y las políticas de garbage collector. Otra de las variables importantes declaradas, fueron la del host del Zookeeper que gestiona las configuraciones y la del puerto por el cual se va acceder al Solr. Este script es utilizado para encender, reiniciar o apagar la instancia de Solr cuando sea requerida, también, existe la alternativa de configurar Solr como un servicio del sistema.

Para cada clúster de Solr (Referencias y Filtros), se dispone de una Collection diferente que representa a un índice de datos. Cada Collection utiliza un conjunto de configuración, que contiene varios archivos que sirven para describir su estructura y funciones. Por eso, este conjunto de archivos de configuración, es referenciado mediante un nombre específico por elección de su creador y almacenado de forma centralizada dentro del Zookeeper. Dentro del conjunto de configuraciones, se destacan el schema, solrconfig.xml y las consultas a la base de datos para la indexación inicial.

El archivo managed-schema contiene la estructura de datos del índice a utilizar en la Collection, dentro del mismo, se declaran todos los campos que son referenciados dentro de un documento en Solr. Cada documento tiene un formato de tipo JSON, que contiene los datos de un trámite en específico. Además, los campos pertenecen a un tipo de dato, que son especificados dentro del schema. Cada tipo de dato utiliza una política propia, que se encarga de describir la forma en que se van a cargar los datos dentro del índice. Las políticas son reconocidas como Tokenizers, que se utilizan para la carga por indexación (desde un fullImport) o por Query (carga desde integra).

Otro de los componentes del conjunto de configuraciones de una Collection es el solrconfig.xml, que es el segundo archivo más importante después del schema. Este componente, es utilizado para configurar los controladores de solicitudes en Solr, como las solicitudes para agregar nuevos documentos al índice o para devolver resultados de una consulta. También, contiene eventos de tipo listener para ejecutar código ante la aparición de algún evento, configurar la interfaz web de administración y el comportamiento de la replicación de datos.

Por último, se encuentran los archivos que contienen la sintaxis de consulta para extraer la información de una fuente de datos. Los archivos de consulta poseen un componente para establecer la conexión a la base de datos, declarando su ubicación, credenciales y tipo de motor de base de datos. Después, se declara la entidad que va a contener todos los documentos (datos de los trámites) con los campos que la componen, que son los mismos que se declararon dentro del schema. Cada campo referenciado en la consulta contiene su contraparte, que hace referencia al nombre de la columna al cual pertenece el dato dentro de la base. En conclusión, estos archivos de consulta trabajan de

manera similar a una Query, con el fin de extraer los datos de todos los trámites en Integra, para indexarlos dentro de la Collection.

En la siguiente captura, se puede observar uno de lotes de consulta para el Solr de Referencias, dentro de la sección del fullImport en la interfaz web. Además, se puede apreciar la definición de los campos de un documento asociado a un trámite en Solr. Por medio de esta interfaz, se realiza la indexación de datos en el clúster.

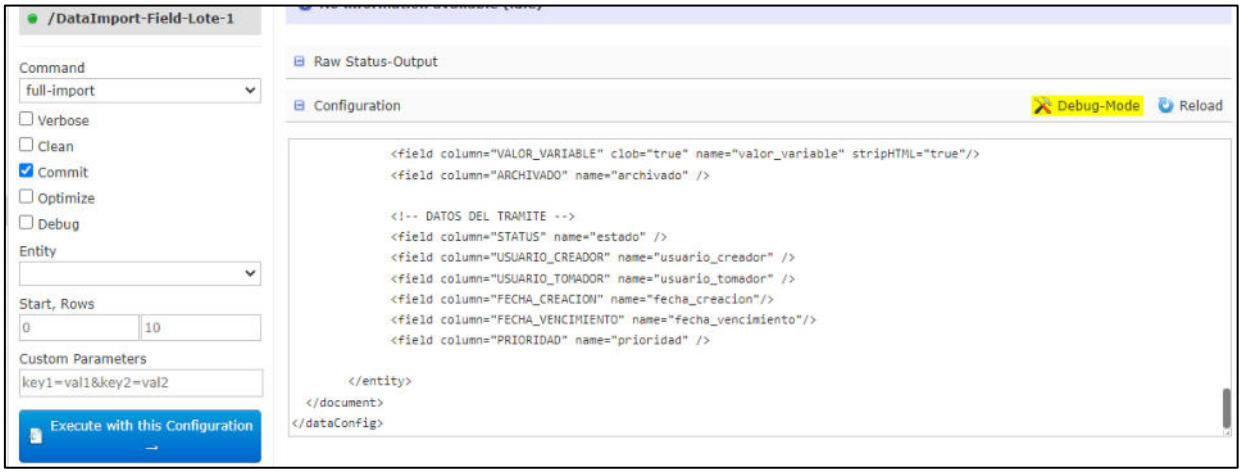


Figura 24 Lote 1 del DataImport de Solr. Fuente: Elaboración propia (basada en la práctica)

Comunicación con los Zookeeper

Solr contiene dentro de sus directorios un script llamado zkcli.sh, que se utiliza para establecer una comunicación entre la instancia de Solr y el Zookeeper. Esta conexión es necesaria, ya que el servicio de Zookeeper gestiona las Collections, todos los archivos de configuración como las queries o el schema de datos de todo el clúster. El script trabaja como una interfaz entre las dos herramientas, permitiendo crear, actualizar o eliminar el conjunto de configuraciones utilizadas por la Collection. También posibilita enlazar una nueva configuración a una Collection existente. Este script fue utilizado dentro de la etapa de migración, para cargar todas las configuraciones necesarias para los respectivos servicios de Solr.

Se presentan algunos ejemplos de comandos, que se utilizaron con el script zkcli.sh:

- Para crear o actualizar una configuración:

  ./zkcli.sh -**z** host_del_zookeeper:puerto -**cmd upconfig** -**confname** nombre_configuración -**confdir** ruta_de_archivos

- Eliminar una configuración del zookeeper:

  ./zkcli.sh -cmd clear -z host_del_zookeeper:puerto /**configs/myconf**

- Enlazar una configuración a una Collection:

  ./zkcli.sh -cmd linkconfig -collection nombre_collection **-confname** nombre_configuración **-z** host_del_zookeeper:puerto

Una vez cargado el conjunto de configuración dentro del Zookeeper, se procede a la creación de la Collection. Esta acción se puede realizar mediante la interfaz web de una instancia de Solr, donde se indica el nombre que va llevar el índice de datos. También, es necesario declarar el alias del conjunto de archivos que se va a utilizar, el número de Shards y la cantidad de replicas por cada uno. A continuación, se observa una captura del formulario de creación para la Collection del Solr de Referencias.

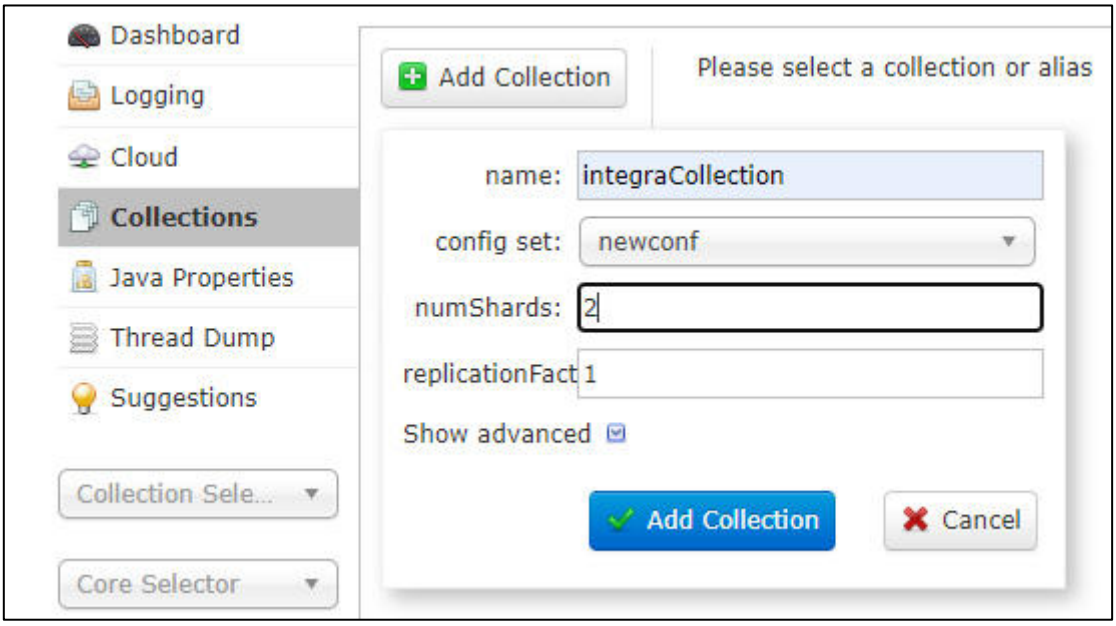


Figura 25 Creación de Collection. Fuente: Elaboración propia (basada en la práctica)

Luego de la creación de una Collection dentro de un Zookeeper, los nodos pertenecientes al clúster son asociados a cada Shard. En el siguiente ejemplo, se pueden identificar la composición de una Collection, con dos nodos líderes

(uno por cada Shard) que van a repartirse la información del índice en partes iguales.

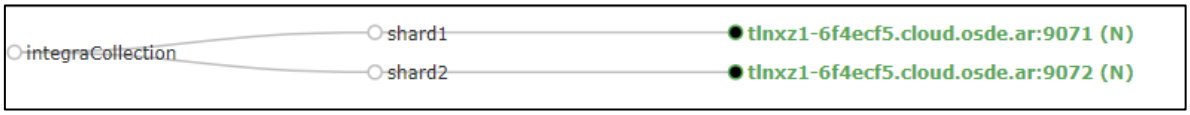


Figura 26 Composición de la Collection IntegraCollection. Fuente: Elaboración propia (basada en la práctica)

Autentificación

Para generar una autentificación y autorización para el inicio de sesión dentro de un clúster de Solr, es necesario emplear las clases BasicAuthPlugin y RuleBasedAuthorizationPlugin. A fin de crear la seguridad en la conexión, se requiere usar el archivo security.json, que define el componente “authentication”, donde se establece que solo las solicitudes autenticadas pueden pasar. También, se declara el usuario y contraseña requeridos para la autentificación, el otro componente utilizado hace referencia a las reglas de usuario. Para esta parte, se definen los roles y los permisos asignados, una vez establecido esos roles, son asignados a los usuarios declarados en la parte inicial del archivo JSON. Después de configurar los usuarios y sus roles, se lleva a cabo el registro de la configuración en la herramienta Zookeeper.

Cambios en Integra

Inicialmente es necesario editar la versión en la dependencia “solr-solrj” por la 8.9.0 dentro del pom de Integra, como también en la aplicación Index Service. Una vez actualizada las versiones en el pom, se requiere migrar a una nueva sintaxis en el armado de las consultas en Solr. Uno de los cambios requeridos dentro de la migración, es el método setSortField que es reemplazado por setSort con el uso de los mismos parámetros (uno para el valor a ordenar y el otro declarar si es ascendente o descendente).

Para el consumo del Solr de Filtros, Integra declaraba una conexión hacia el servicio de Solr utilizando la clase CloudSolrServer, esta conexión apuntaba hacia una sola instancia y no hacia el clúster. Al apuntar la conexión hacia una sola instancia, la herramienta quedaba expuesta a que en el caso de que falle el nodo, el servicio de Solr se interrumpa. Por el contrario, una conexión clusterizada brinda ventajas como la alta disponibilidad cuando ocurre un fallo en algún nodo, distribución de la carga de trabajo y escalabilidad para aumentar la productividad. Para generar una conexión de forma clusterizada como lo hace el Index Service con el Solr de Referencias, es necesario el uso de la clase CloudSolrClient. Además, por temas de seguridad de los datos se planteó agregar una autentificación para la conexión al clúster de Solr, por ese motivo se utilizó la clase provista CloudHttp2SolrClient. La clase CloudHttp2SolrClient permite a su vez registrar un cliente para la autenticación en la conexión, brindando seguridad en los datos para que el acceso sea de manera restringida para el consumo de datos en Solr.

Para llevar a cabo las configuraciones de una conexión con autenticación, fue necesario utilizar los componentes beans para instanciar una clase CloudHttp2SolrClient, dentro de las clases que se comunican con el Solr. En principio, se crea un cliente asincrónico con la clase Builder de Http2SolrClient, que sirve para gestionar las conexiones a diferentes nodos de manera eficiente. Para eso, se declara un bean individual para el Builder y otro que utilice el método withBasicAuthCredentials, para registrar las credenciales de acceso y generar una conexión segura. Para obtener el usuario y la contraseña de la conexión, se utilizan las propiedades de entorno del JBoss donde se registran todas las properties de Integra. Una vez generada la conexión, se crea otro bean para instanciar la clase CloudHttp2SolrClient, que sirve para comunicarse con el cloud de Solr usando HTTP2SolrClient. Junto con el constructor de este bean, se registra un listado con las rutas pertenecientes a los Zookeepers que posee el servicio de Filtros o el de Referencias, dependiendo para quien sea la configuración. Además, se introduce la conexión cliente mencionada anteriormente y se registra el nombre de la Collection (índice de búsqueda) perteneciente al servicio, que se obtiene por medio de las properties del servidor.

El bean generado al final de la configuración es llamado solrClient, el cual es utilizado en todas las clases que realizan Queries para registrar datos, consultas u otra interacción con el clúster de Solr.

Cambios con las nuevas versiones

Con las nuevas versiones de Solr se pueden apreciar los recursos consumidos por cada instancia del clúster, esto permite obtener un análisis completo ante una necesidad. Esta nueva herramienta, se puede utilizar en los casos de que algún nodo contenga errores o posea un bajo rendimiento. En la captura siguiente se puede identificar los valores de CPU, memoria y disco del clúster de Filtros.

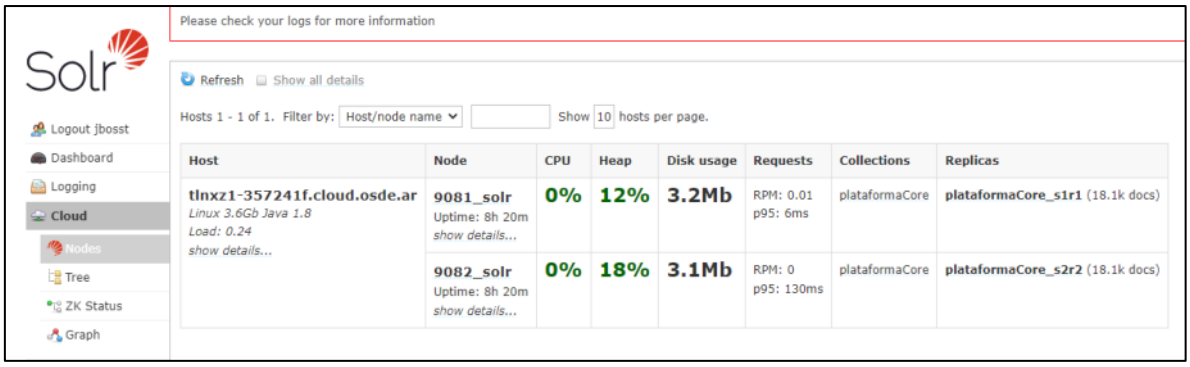


Figura 27 Recursos del Solr. Fuente: Elaboración propia (basada en la práctica)

Con la implementación de las nuevas versiones de la herramienta Solr, surgieron varias modificaciones en la configuración de los archivos schema.xml y solrconf.xml. Para el primer archivo (schema), se tuvieron que eliminar todos los parámetros que hacían referencia a enabledPositionIncrements, porque el mismo viene actualmente con un valor verdadero por defecto. Además, se migraron varias clases que utilizan los FieldType, y siendo necesario reemplazar cada una por su clase equivalente en la nueva versión, trayendo mejoras en su comportamiento. Las clases modificadas fueron las siguientes:

- TrieIntField → IntPointField (para el campo Int)
- TrieFloatField → FloatPointField (para el campo Float)
- TrieLongField → LongPointField (para el campo Long)
- TrieDoubleField → DoublePointField (para el campo Double)

- TrieDateField → DatePointField (para el campo Date)
- LatLonType → LatLonPointSpatialField (para el campo Location)

En el caso del archivo solrconfig.xml, también tuvo cambios considerables con la nueva versión de Solr. Uno de los cambios más destacables fue que se deprecó el RequestHandler que contenía la clase JsonUpdateRequestHandler, donde su funcionalidad pasó a ser gestionada por la clase UpdateRequestHandler. Otro de los cambios que se realizaron, fue el de la eliminación del Handler para la administración de Handlers, ya que el mismo fue deprecado en las versiones recientes por no ser requerido.

Resolución de errores

Durante la migración de la herramienta Solr en el proyecto, se encontraron dos errores que se producían en el uso de la tecnología. Uno de ellos se presentaba en la indexación de fechas cuando se creaba un trámite, la carga del dato se producía en otro tipo de formato diferente al utilizado en la indexación inicial de datos. La carga de fechas en el momento de creación de un nuevo trámite, se cargaba con el siguiente formato incorrecto "**Sun May 09 11:50:52 ART 2021**" dentro de los documentos del Solr. La deficiente carga del dato producía que los usuarios no pudieran buscar sus trámites a través del ingreso de fechas, el formato de fecha utilizado en esos casos era "**dd/mm/aaaa**". El otro error existente es el de la búsqueda de datos que contienen caracteres especiales, por ejemplo, el uso del carácter @ o el – no podían ser utilizados en el buscador superior. La solución de estos errores encontrados durante la ejecución de pruebas en Integra fueron importantes para el uso completo de la herramienta y no tener que restringir o acotar las búsquedas al cliente.

En el primer error encontrado (la carga incorrecta de fechas), se realizó un análisis exhaustivo del problema, para encontrar la fuente en donde se estaba produciendo el error. En este caso, el encargado de generar la Query para la indexación de datos en el Solr de Referencias es la aplicación Index Service, por eso el cambio para modificar el formato de las fechas se realizó en el código fuente de esta misma aplicación. Dicho esto, el cambio se introdujo en la clase

SolrDocumentAssembler, dentro de un método donde se ingresan como parámetros todos los datos pertenecientes al nuevo trámite a indexar. Para modificar el formato de fecha, se utilizó la clase SimpleDateFormat que provee Java, donde se inicializa con el formato de fecha que se requiere utilizar (se utilizó el formato **dd/mm/aaaa).** La clase SimpleDateFormat utiliza el método format, para transformar un objeto de tipo Date en un String con un formato ingresado por parámetro, esta lógica se introdujo en una función separada. La función declarada es invocada en el momento de carga de datos en el armado de la Query, los datos que se ingresan en la función de tipo Date y corresponden a las fechas del trámite (creación y vencimiento). En la captura # se puede observar la función ParseDate y su uso para la carga de fecha de creación y vencimiento.

```
-               docTramite.addField("fecha_creacion", tramite.getFechaIngreso());
-               docTramite.addField("fecha_vencimiento", tramite.getFechaVencimiento());
+               docTramite.addField("fecha_creacion", ParseDate(tramite.getFechaIngreso()));
+               docTramite.addField("fecha_vencimiento", ParseDate(tramite.getFechaVencimiento()));
                docTramite.addField("prioridad", Integer.valueOf(tramite.getPrioridad()));

                return docTramite;
@@ -331,4 +331,16 @@ public class SolrDocumentAssembler {
                }
                return doc;
        }
+       /**
+        * Parsear las fechas de tipo date en un String para el que el solr
+        * pueda almacenarlas en formato dd/MM/yyyy
+        * @param date
+        *
+        * @return un String
+        */
+       public static String ParseDate(Date date)
+   {
+               SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
+               return  formatter.format(date);
+   }
```

Figura 28 Solución del formato fecha en trámites Fuente: Elaboración propia (basada en la práctica)

En la próxima imagen se encuentra una captura de la interfaz de Apache Solr, con los datos fecha_creación y fecha_vencimiento utilizando el formato de fecha correcto para los trámites.

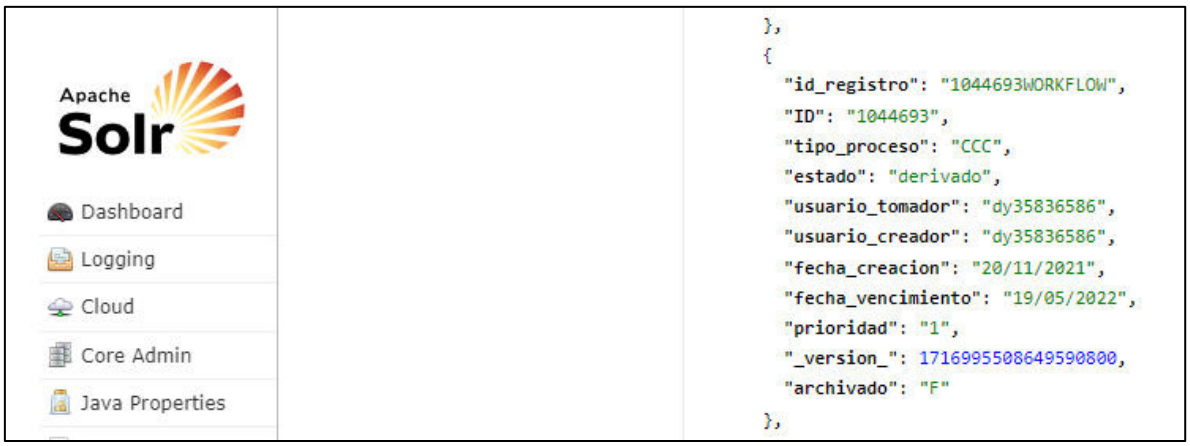


Figura 29 Representación de un trámite en Solr con el formato fecha correcto. Fuente: Elaboración propia (basada en la práctica)

El error para de la búsqueda de trámites por medio de datos con caracteres especiales, se producía en los casos en que los caracteres se encontraran dentro de datos del trámite como un mail o en una observación. Este error se encontraba en la definición del schema de Solr, porque tanto las observaciones como lo datos propios de un proceso pertenecen al campo valor_variable de tipo de dato Text. El tipo de dato Text, tiene dentro de su definición una sección para la indexación de los datos con el uso de un Tokenizer de tipo Standard. El Standard Tokenizer reemplaza algunos caracteres especiales como la arroba, los dos puntos o el guión medio por un espacio en blanco. Este tipo de indexación genera que cuando el usuario realice una búsqueda de algún trámite con estos caracteres no lo pueda encontrar, ya que fueron reemplazados por un espacio en blanco.

Para entender mejor el propósito de los Tokenizers en la herramienta Solr, se pueden definir como los responsables de dividir los datos de campo en unidades léxicas o tokens. Son utilizados tanto para la indexación de datos como para la búsqueda de estos.

Para la solución al error de caracteres especiales, se crea un nuevo tipo de dato denominado text_valor_variable que reemplaza al Text, con la única diferencia que este emplea el tipo de Tokenizer Standard por el White Space para la indexación de datos. A diferencia del Tokenizer anterior, el White Space divide el flujo de texto en espacios en blanco y devuelve secuencias de caracteres que no son espacios en blanco como tokens.

A continuación, se agregan ejemplos de la forma en que se indexan los datos en el Solr con los dos tipos de Tokenizers mencionados.

Standard Tokenizer

- **Entrada:** " hola, cómo estás?"
- **Salida:** " hola,", " cómo ", " estás?"

White Space Tokenizer

- **Entrada:** "federico diaz, email fede.diaz@gmail.com fecha 08-10, localidad: lp."
- **Salida:** " federico ", " diaz ", "email", "fede.diaz", "gmail.com", "fecha", "08", "10", "localidad", "lp"

2.8 Pruebas y monitoreo

Para la última etapa del proyecto PPS, se llevó a cabo una serie de pruebas relacionadas a la funcionalidad básica que debe brindar el software de Integra. Estas pruebas también fueron hechas durante el transcurso del proyecto, por el motivo de que los cambios realizados en las tecnologías puedan haber afectado el funcionamiento de la aplicación. Las pruebas que se realizaron en Integra fueron las siguientes:

- Creación de trámites: creación de distintos procesos en la aplicación, que involucraran la integración con aplicaciones externas, SelfIntegration e IntegraForms.
- Servicios: uso de los servicios expuestos por Integra.
- Observaciones: agregar observaciones en un trámite, en algunos casos agregando caracteres especiales para validar la carga correcta.
- Archivar y desarchivar trámites.
- Integración con los servicios de Solr.
- Indexación de datos sobre trámites creados o modificados en el índice de Solr.
- Agregar adjuntos.

- Validar carga correcta de datos de un trámite.

Para el monitoreo de la aplicación Integra en el ambiente productivo, se implementó una aplicación interna llamada Willy que se usa en otras aplicaciones de OSDE. Esta aplicación se encuentra conectada al script de arranque del JBoss, utilizando la información extraída genera gráficos con los recursos utilizados. Willy contiene varios gráficos importantes, entre los que se encuentra el de la memoria consumida por Integra, porcentaje de cpu, utilizado e hilos en uso. Otra de las gráficas representadas son cantidad de peticiones, ejecución de Garbage Collector y tiempos de respuesta de la base de datos. Todos los gráficos mencionados representan el comportamiento de cada instancia de Integra y con la ayuda de esta herramienta se puede medir la performance de la aplicación. Se adjunta una captura con los gráficos proporcionados por la aplicación Willy sobre IntegraBPM.

Figura 30 Gráficos proporcionados por Willy. Fuente: Elaboración propia (basada en la práctica)

3. Conclusiones

La finalización de la presente práctica profesional supervisada consistió en el desarrollo de los cuatro objetivos principales establecidos en el inicio sobre la aplicación IntegraBPM. Dichos objetivos se centraron en otorgar mejores herramientas al software y mejorar su rendimiento. El desarrollo de la práctica

profesional se llevó a cabo en tres etapas: la primera fue una fase de análisis de tecnologías y necesidades de la aplicación, la segunda se centró en la investigación de las tecnologías a aplicar y su desarrollo y la última etapa consistió en una serie de pruebas de funcionalidad, monitoreo de rendimiento y la obtención de resultados.

Se logró el primer objetivo del proyecto, que consistió en la aplicación de nuevas tecnologías y herramientas para la optimización del Software. Por una parte, esto cumplió con las necesidades del cliente sobre aumentar las versiones de sus tecnologías por las utilizadas en el mercado tecnológico, agregando un análisis completo de las compatibilidades para un mejor rendimiento. Además, se otorgó una mejor posición para futuras mejoras en los servicios brindados por la aplicación, ofreciendo soluciones de una mejor calidad. Por otra parte, se actualizaron las versiones de las herramientas que proporcionan soporte de parte de las licencias del software, ya que es uno de los motivos principales del proyecto. Uno de los inconvenientes resultantes, fue la convivencia entre las tecnologías durante la modificación de sus versiones, por eso fue muy importante gestionar el orden en el aumento de las versiones y de este modo evitar los errores de incompatibilidad.

El segundo y tercero objetivo establecido fueron realizados, estos hicieron referencia a mejorar la performance del sistema, eficiencia en procesos y disminución de errores existentes. La mejora en el rendimiento se cumplió con la implementación de nuevas versiones en las herramientas, consiguiendo técnicas más eficientes para el manejo de datos y que corrigen problemas de seguridad de versiones anteriores. En conjunto con el cambio de tecnologías se generó una optimización y refactorización del código en Integra, que tuvo como resultado final un código más mantenible a futuro. También, por medio de los logs proporcionados por JBoss, se pudo observar que se solucionaron varios errores que se registraron y afectaban la performance de Integra. Además, se produjo un nivel de productividad alto teniendo en cuenta que se dieron de baja dos instancias en producción. Con la implementación de Willy, se pudo observar los eficientes valores de uso de memoria, procesador y mejores niveles de respuesta.

El último de los objetivos cumplidos en la PPS, fue la aplicación de las prácticas de integración continua y de despliegue en Integra con el uso de la metodología DevOps. El impacto de esta técnica de trabajo se ve reflejado tanto en la productividad como en el desarrollo ágil, ya que con su implementación se logran mejores tiempos en el proceso de implementación. La reducción en los tiempos es consecuencia de la automatización de tareas repetitivas en los procesos, dejando una posibilidad de agregar otras tareas similares al circuito. También, con el uso de DevOps se minimiza significativamente el error humano durante los despliegues en el ambiente productivo, delegando la compilación y armado del War a la herramienta. Otras de las ventajas que trae su implementación, es el fácil manejo de la tecnología con el uso de una interfaz sencilla para una rápida adaptación. Para concluir, con los cambios realizados durante todo el proceso de migración se obtuvo una aplicación más estable y mantenible, acorde a las necesidades del mercado.

3.1 Reflexión sobre la práctica profesional supervisada como espacio de formación

En lo personal, siento que adquirí mucha experiencia durante la realización de mi proyecto PPS, en cuanto a la gestión de proyecto de forma individual y cumplir los objetivos fijados en tiempo y forma. Realizar una planificación, definición de objetivos acordes a los tiempos y características del proyecto, del mismo modo, la implementación de una metodología de trabajo como Scrum. También, uno de los desafíos más importantes fue el de la redacción del informe final, una tarea de gran volumen y que a su vez no contenía mucha experiencia con tareas similares, ya que en mi caso desarrollé un perfil más técnico y de gestión durante la carrera.

Otros factores importantes durante el proyecto final son los obstáculos o inconvenientes, estos aparecieron en diferentes facetas del proyecto de manera imprevista. En ese momento, fue cuando se puso en práctica el nivel de gestión, para responder de manera eficaz y rápida con el menor impacto posible.

En mi opinión, pienso que la práctica profesional supervisada tiene un valor muy importante en la finalización de la carrera del alumno y su formación. Es una

oportunidad para aplicar todos los conocimientos adquiridos a lo largo de la formación, las herramientas, técnicas y conceptos teóricos sobre cada etapa del desarrollo del proyecto.

4. Bibliografía

- Conoce qué es Spring Framework y por qué usarlo. Recuperado de: https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/ [Consulta: agosto 2021]
- Solr: el servidor de búsqueda de Apache. Recuperado de: https://www.ionos.es/digitalguide/servidores/configuracion/apache-solr/ [Consulta: agosto 2021]
- CI/CD concepts. Recuperado de: https://docs.gitlab.com/ee/ci/introduction/ [Consulta: agosto 2021]
- Introducción a JBoss EAP. Recuperado de: https://access.redhat.com/documentation/es-es/red_hat_jboss_enterprise_application_platform/7.1/html/introduction_to_jboss_eap/overview_of_general_concepts#java_ee_7 [Consulta: agosto 2021]
- Maven Repositorio. Recuperado de: https://mvnrepository.com/ [Consulta: septiembre 2021]
- Hibernate. Recuperado de: https://hibernate.org/ [Consulta: septiembre 2021]
- Spring Framework. Recuperado de: https://docs.spring.io/ [Consulta: septiembre 2021]
- Apache CXF. Recuperado de: http://cxf.apache.org/ [Consulta: octubre 2021]
- GitLab CI/CD. Recuperado de: https://docs.gitlab.com/ee/ci/ [Consulta: noviembre 2021]
- Guía de referencia de Apache Solr. Recuperado de: https://solr.apache.org/guide/8_9/ [Consulta: noviembre 2021]