UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MEJORA DEL PROCESO DE DESARROLLO DE UNA EMPRESA DE SERVICIOS DE INFORMACIÓN MEDIANTE LA INCORPORACIÓN DE DEVOPS

TESIS PARA OPTAR AL GRADO DE MAGISTER EN
TECNOLOGÍAS DE LA INFORMACIÓN

EDUARDO ALBERTO DÍAZ CORTÉS

PROFESOR GUÍA:
DANIEL PEROVICH GEROSA

MIEMBROS DE LA COMISIÓN:
ALEXANDRE BERGEL
SERGIO OCHOA DELORENZI
ALCIDES QUISPE SANCA

SANTIAGO DE CHILE
2019

RESUMEN

Previred S.A, empresa de servicios de información, cuenta con una división de negocios denominada Apoyo al Giro que presta servicios para la industria previsional y de seguridad social nacional. Esta división cuenta con un equipo de desarrollo de software dedicado que construye la mayor parte de los sistemas de información que apoyan las labores de apoyo al giro. El nivel de demanda y exigencia por estos servicios ha aumentado, al punto que gran parte de ellos son considerados críticos por los clientes de Previred. Esto compromete a la organización a cumplir altos niveles de servicios y a garantizar la continuidad operativa de los mismos.

Actualmente el proceso de implantación de nuevas versiones de los sistemas se realiza mediante procesos manuales, con una tasa de fallos considerada insatisfactoria por los clientes internos y externos. Por otro lado, se ha establecido una métrica sobre la tasa de fallos críticos, cuyo valor se espera disminuir. Un fallo crítico corresponde a una indisponibilidad del servicio productivo por varias horas o incluso días. Una parte de estos fallos se debe a errores en el proceso de implantación en producción, por mala ejecución de las instrucciones, falta de prolijidad en la instalación, o en la elaboración de los documentos que describen los pasos a producción.

El objetivo general de este trabajo de tesis es ajustar el proceso de desarrollo mediante la implantación de los procesos automatizados de integración y entrega continua, incorporando procesos y herramientas de DevOps dentro de la organización, con el fin de reducir la tasa de fallos críticos debidos al proceso actual. Esto es aplicado en un piloto de un nuevo servicio productivo de Previred.

Para alcanzar el objetivo planteado primero se revisa el actual proceso de desarrollo de Previred, y luego se determinan los principales problemas y “dolores” que experimenta la organización con este proceso, mediante entrevistas a personas claves de la organización. En base a los antecedentes recogidos se proponen modificaciones al proceso de desarrollo, junto con una plataforma tecnológica que apoya estos cambios. Para plasmar esta plataforma, se propone una arquitectura de referencia y para construirla se analizan las herramientas disponibles y se seleccionan las adecuadas para la cultura y realidad de Previred.

Para verificar la factibilidad de la arquitectura se realizó primero una prueba de concepto, y luego se validó la plataforma mediante un piloto aplicado a un proyecto de desarrollo de un nuevo servicio. Además, se realizó una evaluación cualitativa de la solución a través de una encuesta a un grupo de personas clave en la organización y a los participantes en el piloto. Por último, se relevaron las mejoras obtenidas en el proceso mediante la obtención de algunas métricas del proceso.

TABLA DE CONTENIDO

1. Introducción

Servicios de Administración Previsional S.A., conocida como Previred, es una empresa fundada en el año 2000 con el fin de apoyar en el proceso de recaudación de cotizaciones previsionales y de salud, usando como medio de contacto la Web. Con el tiempo, Previred empezó a ofrecer más servicios a las AFP, Isapres, compañías de seguro, Fonasa, IPS y otras instituciones del sector de la Seguridad Social.

En 2008, producto de la Reforma Previsional (ley 20.255), Previred debió ampliar su cartera de productos y servicios para cubrir las necesidades que surgieron de esta reforma. Para este efecto creó el Área de Apoyo al Giro (AG) que presta servicios de información al sector de la seguridad social del país. Esta unidad se formó con su propio equipo de tecnología, incluyendo un área propia de desarrollo de software, operando de manera autónoma con respecto al ya existente en Previred. En ese entonces, se consideró incluso la posibilidad de que AG llegase a operar como empresa independiente, incubada en Previred.

Con el tiempo, se descartó la idea de constituir otra empresa y AG fue incorporado totalmente a la estructura organizacional de Previred. Por razones de eficiencia, la Unidad de Tecnología de AG delegó sus labores de administración y operación de la infraestructura a la Subgerencia de Sistemas de Previred, a partir de noviembre de 2011.

Es por esta razón que Previred cuenta hoy en día con dos áreas de desarrollo de software, la *Subgerencia de Desarrollo* que construye soluciones de software para los servicios prestados por el Área de Recaudación, y la *Subgerencia de Tecnología* que desarrolla sistemas para el Área de Apoyo al Giro. La Figura 1 ilustra la organización actual de las áreas de tecnología en Previred.

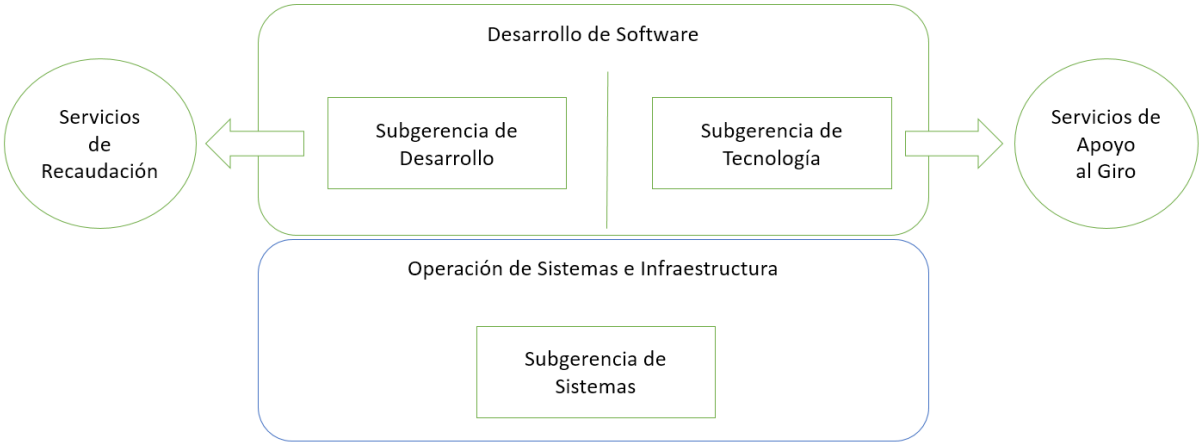

Figura 1. Organización actual de las áreas de tecnología en Previred.

En la actualidad, el 80% del desarrollo de software se ejecuta con equipos internos. El 20% restante se gestiona a través de proyectos dirigidos por las respectivas unidades con el apoyo de proveedores externos. Los sistemas que se desarrollan en Previred corresponden a servicios de información que apoyan a diversos procesos de negocio de

la industria de la seguridad social. Los principales corresponden a recaudación de cotizaciones previsionales, cobranza de moras en el pago de cotizaciones previsionales, servicios de custodia y almacenamiento de registros históricos de cuentas previsionales, determinación de beneficios estatales, procesos de traspaso de afiliados, servicios de información previsional, etc.

Hoy en día Previred ofrece una cartera de casi sesenta servicios de información para el sector de la seguridad social (privada y estatal) y las necesidades del mercado siguen en aumento. Es por esto por lo que se les pide a las áreas de desarrollo una mayor agilidad en la entrega de sus productos.

Junto con esto, la criticidad de los servicios es cada vez mayor. Previred tiene comprometidos altos niveles de cumplimiento, requerimientos de operación cercanos a 24x7 en varios de sus servicios, además de un compromiso de asegurar la continuidad operativa y la seguridad de la información.

1.1 Problema

La implantación (*deployment*) de nuevas versiones de los sistemas existentes se realiza con un proceso manual, con una tasa de fallos considerada insatisfactoria por nuestros clientes internos y externos. De acuerdo a las mediciones realizadas en Previred, un 15% de los pasos a producción tienen un error o fallo considerado crítico.

Para la organización, un fallo crítico en producción corresponde a una falla en el software que impide su utilización por varias horas (o incluso días), o que afecta la imagen del servicio y pone en riesgo su continuidad. Estos errores pueden afectar el acceso, la integridad o la disponibilidad de la información. De este 15%, un 9% se atribuye a mala gestión de la calidad en el proceso de desarrollo de software, normalmente por mala especificación de requisitos, o mala calidad en el análisis. El otro 6% corresponde a errores en el proceso de implantación en producción de las piezas de software entregadas, como la instalación de piezas incorrectas, error en el orden de ejecución de los pasos de instalación, mala interpretación de las instrucciones, o falta de prolijidad en el proceso de implantación. A su vez, el diagnosticar el origen de una falla no tiene la velocidad requerida para permitir una reacción a tiempo.

Un análisis de las causas de estos errores permitió identificar las siguientes falencias en el proceso de desarrollo y de implantación de sistemas:

1. El nivel de cobertura del testing unitario es insuficiente, con apenas un 25% para algunos proyectos. Además, en muchos casos no se cuenta con tests unitarios.
2. El nivel de cobertura del testing regresivo es mínimo, contando sólo un 20% de los proyectos con planes de pruebas o tests automatizados de regresión.
3. La retroalimentación de las pruebas funcionales hacia el equipo de desarrollo es tardía.
4. Los ambientes de Desarrollo, QA y Producción difieren en las características de configuración y en la plataforma tecnológica que proveen (por ejemplo, en un proyecto se contaba con versiones distintas del motor base de datos entre los ambientes de QA y Producción, en otro caso un archivo de parámetros estaba

instalado en directorios distintos). Esta falta de homologación entre ambientes dificulta el diagnóstico de los errores y también ha sido una causa de fallas.

5. Dado que la implantación se realiza manualmente, siguiendo las instrucciones descritas en un “documento de instalación”, se producen errores frecuentemente debido a la mala transcripción, lectura o interpretación de las instrucciones.

En resumen, Previred se enfrenta al problema de que la implantación de nuevas versiones de sus sistemas de información se realiza sin testing suficiente y mediante un proceso manual que requiere esfuerzo, lo que introduce errores por su mala ejecución. Si bien la tasa de fallas no es alta (15%), se ha observado que hay una mayor demanda por desarrollo de nuevos servicios, y en los últimos dos años se ha observado un deterioro en la tasa de fallas críticas. Como consecuencia, Previred necesita resolver de una mejor manera su forma de poner sus sistemas en producción, de modo de reducir esta tasa de fallos críticos que pasan a producción, y así disminuir los incidentes y mantener un nivel de continuidad operativa adecuado para los requerimientos de sus clientes, que son cada vez más exigentes.

1.2 Solución

Esta tesis expone la adopción de procesos y herramientas que permiten incorporar parte importante de las prácticas DevOps en la organización. Tal como se expone en la Sección 2, DevOps comprende un conjunto de técnicas y herramientas que apoyan a la mejora del proceso de desarrollo de software, que se apoya en la automatización y monitoreo de los procesos de integración e implantación de software.

Esta tesis no considera la transformación total de la organización para la adopción de la cultura DevOps puesto que ésta es una decisión estratégica mayor que requiere un cambio organizacional y cultural fuera del ámbito de este trabajo. Específicamente, en DevOps hay aspectos de comunicación entre áreas y de gestión de personas que no son abordados por esta tesis.

La solución construida es principalmente de índole técnico, y propone cambios en el proceso de desarrollo y la adopción de herramientas tecnológicas. Para orientar el trabajo, se hizo primero un análisis de los principales dolores de la organización en el proceso de desarrollo y entrega del software. En función de los resultados arrojados por dicho estudio, se decidió que para desarrollar esta solución se adoptarían las siguientes prácticas de DevOps:

1. Integración Continua para automatizar el proceso de construcción (building) de los artefactos que serán desplegados en los ambientes de QA y Producción.
2. Despliegue automático de piezas de software en ambientes de QA.
3. Automatización parcial de pruebas funcionales.
4. Automatización de pruebas de estrés y de servicios web.
5. Inspección automática y estática de código.

6. Adopción de tecnología de contenedores, de modo que la configuración de la infraestructura sea gestionada de la misma forma en que se gestiona el código fuente de las aplicaciones (Infrastructure as Code).

Para adoptar estas prácticas, se necesita implantar una infraestructura que dé soporte a los procesos de entrega e implantación continua, tal como se propone en (Mouser, 2015).

Un relevamiento preliminar de técnicas y herramientas de DevOps, realizado en conjunto con proveedores tecnológicos de Previred, más la ejecución de una prueba de concepto, permitió identificar las herramientas para la arquitectura definida.

Para la plataforma como servicio se implantó el producto la OpenShift on premise de RedHat. La razón principal de la elección de este producto es el alto nivel de compatibilidad con las plataformas usadas por las aplicaciones desarrolladas en Previred. En particular, OpenShift corresponde a una implementación de una PaaS (Platform as a Service) que incorpora en forma nativa el Sistema Operativo RHEL y JBoss, dos componentes fundamentales de la arquitectura de sistemas usadas en Previred. Toda esta elección de herramientas y plataformas fue respaldada por un estudio de necesidades y criterios de aceptación.

1.3 Objetivos

El objetivo general de este trabajo es implementar un proceso de integración y entrega continua, incorporando procesos y herramientas de DevOps dentro de la organización, para un servicio productivo de Previred.

Para lograr este objetivo, se definen los siguientes objetivos específicos:

- Definir, documentar e implantar el proceso de integración continua para el desarrollo de software. Este proceso comprende el *building* de las aplicaciones, la ejecución de tests unitarios y el despliegue a los ambientes de Desarrollo y QA.
- Implantar la creación y despliegue de los ambientes mediante contenedores orquestados y el despliegue de las aplicaciones en estos ambientes.
- Agregar pruebas automatizadas y de regresión como una etapa del proceso de integración continua.
- Aplicar todo lo anterior en la ejecución de un piloto sobre un proyecto actual de Previred.

1.4 Metodología

Las actividades realizadas para construir y validar la solución, y así alcanzar los objetivos propuestos, fueron las siguientes:

1. Relevamiento de necesidades que podrían verse beneficiadas por la aplicación de técnicas y herramientas de DevOps.
2. Relevamiento de herramientas, técnicas y procesos de DevOps que abordan los problemas detectados y que además se adapten a la cultura de Previred.

3. Implantación de un Repositorio de Artefactos de software en Previred.
4. Implantación y configuración de un Orquestador que habilite la adopción de Integración Continua.
5. Definición de los procesos de construcción automática de software integrados con el repositorio de artefactos.
6. Adquisición, instalación y configuración la plataforma en el Data Center, utilizando la técnica de Plataforma como Servicio (PaaS).
7. Ejecución de una prueba de concepto de la integración de la PaaS con el Orquestador para habilitar la adopción de Entrega e Implantación Continua.
8. Validación del proceso y la arquitectura mediante la ejecución de un piloto aplicado a un proyecto de desarrollo de un nuevo servicio en Previred.
9. Validación cualitativa del proceso y la arquitectura mediante entrevistas semiestructuradas a principales interesados en el proceso de desarrollo de software dentro de la organización.

1.5 Estructura del Informe

Este informe de tesis está estructurado de la siguiente forma. En el Capítulo 2 se presenta el marco teórico que expone los principales conceptos y elementos de la cultura DevOps que son relevantes a este trabajo. En el Capítulo 3 se revisa el actual proceso de desarrollo de Previred, luego se realiza un estudio de los principales problemas y "dolores" que experimenta la organización con este proceso. En el Capítulo 4 propone ajustes al proceso junto con el apoyo tecnológico necesario para sostener estos ajustes. Luego se analizan las herramientas disponibles y se seleccionan las adecuadas a la cultura y realidad de Previred, basado en criterios detallados en dicho capítulo. En el Capítulo 5 se presenta la validación de la solución. Primero se analiza la factibilidad de la plataforma construida y documentada en el Capítulo 4. Segundo, se expone los resultados de la prueba de concepto. Tercero se expone la ejecución del piloto y los resultados obtenidos y, por último, se analiza una evaluación cualitativa de la solución realizada a un grupo de personas claves en la organización. El Capítulo 6 presenta las conclusiones e indica posibles trabajos futuros.

2. Marco teórico

2.1 Las fases del ciclo de desarrollo de software tradicional

El desarrollo de software tradicional considera diversas fases conocidas y comunes. Primero se realiza una etapa de especificación y análisis de requisitos, seguida de una fase de diseño y posteriormente de una fase de construcción del software. Este proceso genera como salida una serie de artefactos o piezas de software, que pasan por una etapa de validación a cargo de un equipo de control de calidad. Luego los usuarios aprueban estos artefactos, validando que las funcionalidades implementadas satisfacen los criterios de aceptación, prestablecidos en la fase de especificación de requisitos. Con la aprobación de los usuarios, se entrega el software al área de operaciones para que lo instale en el ambiente de producción.

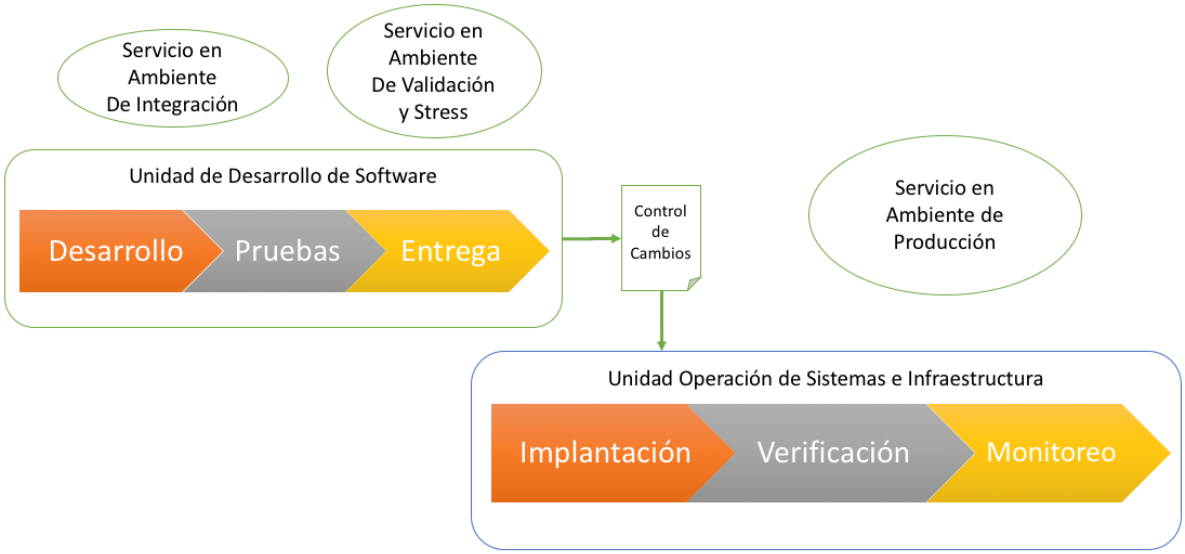

Figura 2. Proceso de desarrollo y puesta en producción.

En muchas organizaciones este proceso ocurre de acuerdo a lo ilustrado en la Figura 2. La fase de Desarrollo comprende el análisis, diseño y construcción. La fase de Pruebas comprende la ejecución de test y la corrección de errores. La Entrega a operaciones se realiza formalmente mediante un documento de Control de Cambios. La implantación del software a los distintos ambientes es ejecutada manualmente siguiendo las instrucciones descritas en documentos de Control de Cambios. Tal como se aprecia en Figura 2, el desarrollo y la implantación son realizadas por unidades separadas que se comunican a través de documentos y reuniones formales.

2.2 Habilitación de DevOps

DevOps es un proceso de desarrollo y entrega de software que se apoya en la automatización y monitoreo de los procesos de integración de software, testing,

despliegue y cambios en la infraestructura. Esto se logra estableciendo un conjunto de prácticas y hábitos, junto con un entorno donde la construcción, testeo y liberación del software ocurren de manera continua, rápida, frecuente y confiable (Mouser, 2015) (Kim, 2013).

De acuerdo a la literatura, la habilitación de DevOps requiere una serie de etapas [1] (Gene Kim, 2016):

1. *Implementar Integración y Testing continuo* (Continuous Integration, o CI), mediante la automatización de los procesos de compilación (building) y la automatización de las pruebas (testing automatizado).
2. *Implementar Entrega y luego Implantación Continua* (Continuous Delivery o CD y Continuous Deployment) mediante la automatización del proceso de instalación de las aplicaciones en los distintos ambientes (QA, Stress y en el caso de la Implantación Continua también en Producción). La Entrega Continua permite asegurar que las aplicaciones operarán adecuadamente en un ambiente productivo mediante un proceso automatizado de pruebas. La Implantación Continua es el paso que sigue, que permite automatizar totalmente la entrega en producción [8]. No todas las organizaciones pueden instituir la Implantación Continua, e incluso en algunos casos puede no ser posible (por ejemplo, debido a marcos regulatorios).
3. *Gestionar la configuración de los distintos ambientes*, lo cual es necesario para lograr la Implantación Continua, mediante la implementación de tecnología que permita configurar y desplegar infraestructura de manera controlada (conocido también como Infrastructure as Code).
4. *Monitorear* y obtener retroalimentación a lo largo de toda la cadena de entrega (Continuous Monitoring), mediante herramientas de monitoreo en los ambientes productivos (alertas, análisis dinámico de logs, monitoreo de la plataforma).

La Figura 3 ilustra las etapas de la habilitación de las técnicas de DevOps, que muestra los niveles de automatización que se pueden alcanzar usando procesos y herramientas de DevOps.

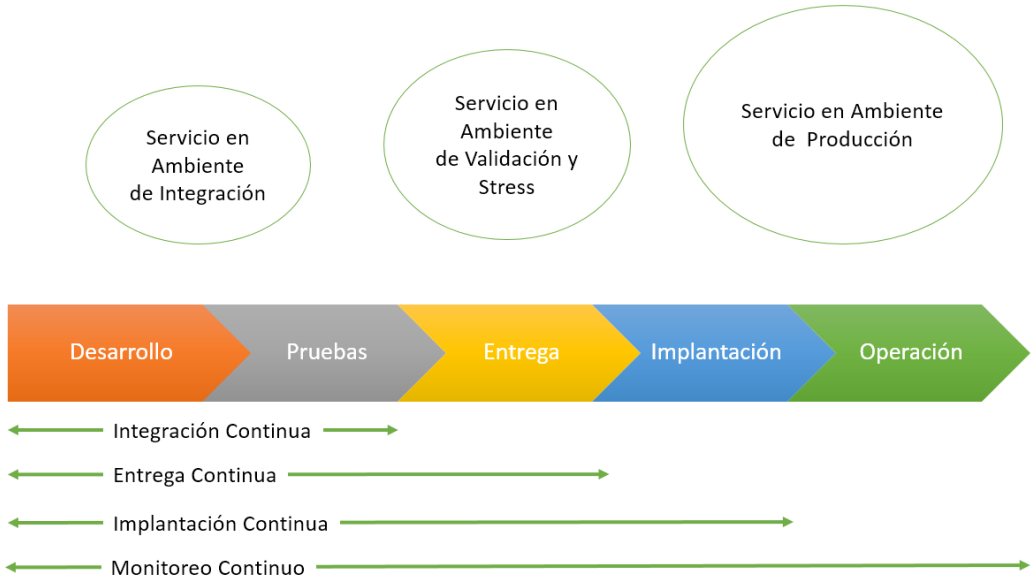

Figura 3. Niveles de automatización en DevOps.

El proceso de Integración Continua y su extensión, la Entrega Continua (Fowler), requieren la adopción de una cierta infraestructura que los soporte. Gruver y Mouser proponen en (Mouser, 2015) una arquitectura para dicha infraestructura de soporte, tal como lo ilustra la Figura 4.

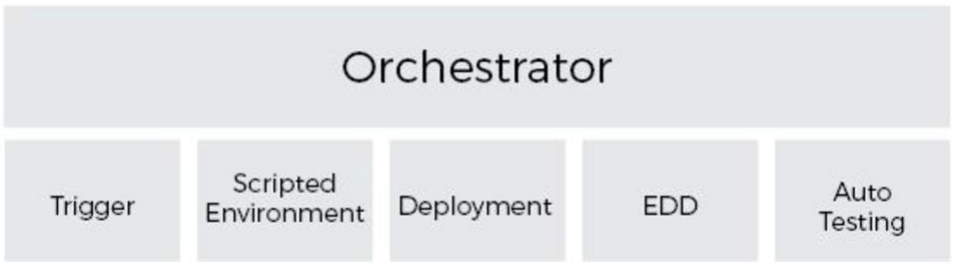

Figura 4. Arquitectura para soportar Entrega Continua, de acuerdo a Gruver y Mouser [1].

En el modelo propuesto por Gruver y Mouser, existe un orquestador (Orchestrator) que corresponde a un sistema que se encarga de controlar todo el proceso automatizado. Todo se gatilla desde un agente externo (Trigger), el cual corresponde normalmente a un proceso de entrega de código fuente a través de un gestor de versiones conectado al orquestador.

Una vez gatillado el evento de entrega, se requiere que el orquestador decida si es necesario crear un nuevo ambiente. La creación de los ambientes está preconfigurada mediante un guion (script) que permite definir cómo instalar y configurar un ambiente donde se ejecutará el software. Una forma de adoptar este mecanismo es utilizar tecnología de contenedores (*containers*), por ejemplo mediante la tecnología Docker ("What is Docker"). Esto se explica en más detalle en la Sección 2.3. Lo importante es que mediante estos scripts es posible habilitar servidores configurados con todo lo necesario para soportar las aplicaciones a implantar (Farcic, 2016).

Una vez obtenido el ambiente, se produce el despliegue de la aplicación, lo que implica instalar automáticamente las piezas y configurarlas para que empiecen a operar. A veces es necesario hacer cambios en las bases de datos (Mouser, 2015). Esto se administra mediante mecanismos de gestión de la evolución de la base de datos (Sadalage, 2006), los que corresponden a herramientas que calculan las diferencias que se producen en el modelo de datos y generan scripts con instrucciones para hacer cambio a nivel de estructura de tablas, y scripts que ejecutan la migración de datos correspondientes. Por último, la aplicación es testeada de forma automática, entregando retroalimentación al equipo técnico de operaciones sobre el estado de la misma.

Para que esta arquitectura funcione, un concepto esencial es el de “Infraestructura como Código” (Infrastructure as Code) [1], que se ilustra en la Figura 5. En la actualidad es posible generar ambientes virtuales que contienen todas las características necesarias para que opere el software. Lo relevante es que se puede describir en un script común todo lo necesario para operarlo, lo que permite simplemente aplicar parámetros de operación según el tipo de ambiente en que se quiere realizar la implantación. De este modo, los distintos ambientes de operación son idénticos (inmutables), lo que permite descartar errores debido a la mala configuración de los ambientes.

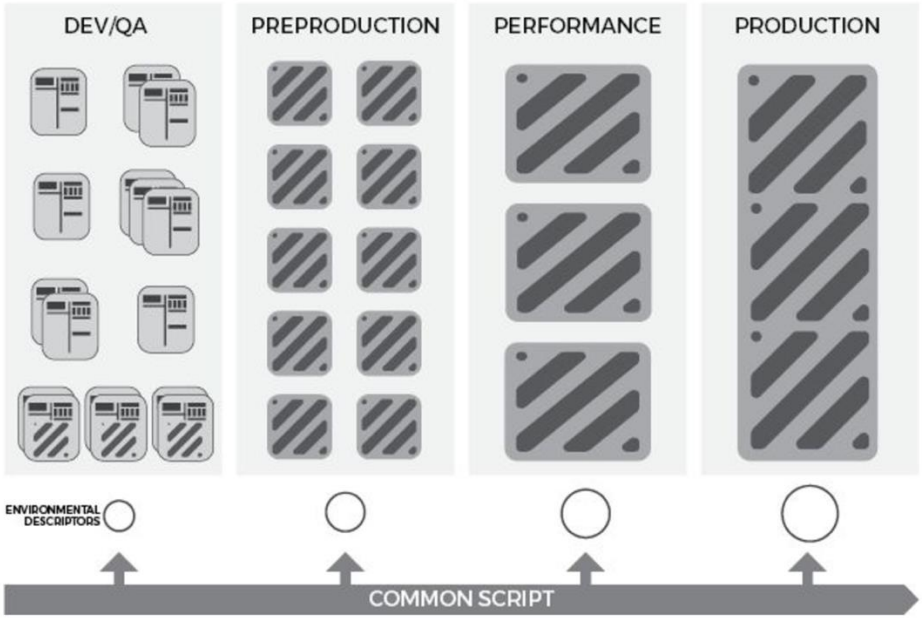

Figura 5. Infraestructura como código, de acuerdo a Gruver y Mouser [1].

2.3 Tecnología de contenedores y su importancia para habilitar DevOps

Antes del surgimiento de la tecnología de contenedores, trabajar con la implantación (*deployment*) de aplicaciones era un proceso complicado de automatizar. Los principales problemas se daban en la homologación de los ambientes. Para lograr una homologación de ambientes se requiere disciplina y control, de modo de garantizar que éstos contengan las mismas configuraciones básicas necesarias. Sin embargo, a pesar de todos los esfuerzos y controles, pareciera ser una meta difícil de lograr [2] [4].

La metáfora de contenedores viene del mundo de la logística y del transporte. Los contenedores son un medio que facilita el transporte de mercaderías a través de camiones, barcos y aviones. Tienen una forma estándar que facilita y ordena el despliegue de la carga en cada uno de estos medios de transporte. Son fáciles de disponer y apilar, es decir, se pueden colocar unos sobre otros, sin que se dañe el contenido. Para los encargados del transporte, lo que hay adentro del contenedor es incluso desconocido. Lo único relevante para el transporte es saber dónde recogerlos y dónde dejarlos. Los transportistas saben cómo manejar los contenedores por fuera, pero el contenido sólo interesa a quien empaquetó la carga y a su destinatario.

Los contendores de software implementan una idea similar. Son imágenes binarias, aisladas, autosuficientes e inmutables del software base, que proveen una funcionalidad diseñada accesible sólo a través de una API publicada. Son una solución para lograr que nuestro software corra en cualquier ambiente (el laptop del desarrollador, un servidor de testing, en servidores de producción, en data centers y en la nube), pero de modo tal que el resultado que obtengamos sea el mismo, salvo diferencias en capacidad.

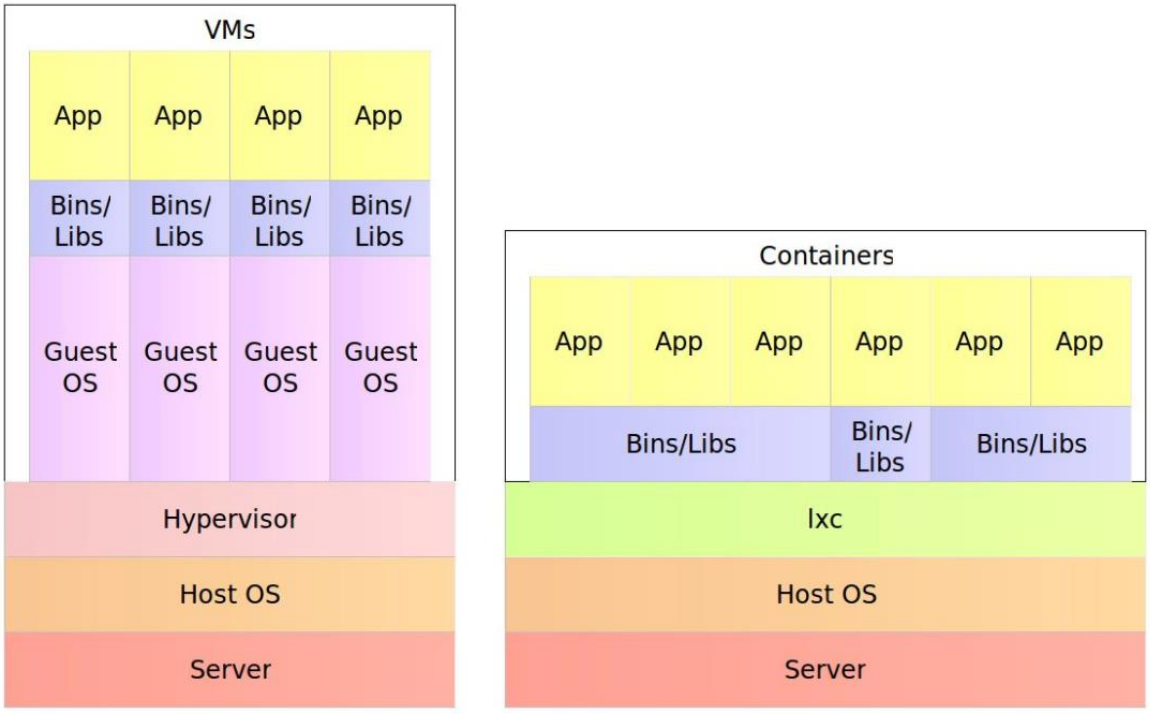

Figura 6. Máquinas virtuales vs contenedores, comparación en el uso de recurso, de acuerdo a Farcic [2].

Para lograr esto, los contenedores tienen dos propiedades importantes: *autosuficiencia e inmutabilidad*. En un ambiente tradicional, al colocar un artefacto se espera que todo esté en su lugar: el servidor de aplicaciones, los archivos de configuración, las dependencias, etcétera. En un ambiente basado en contenedores, éstos son los que portan en su interior todo lo necesario para las necesidades del software.

Aunque la tecnología de contenedores es similar a la tecnología de máquinas virtuales, es importante identificar sus diferencias. En un ambiente de virtualización, si tenemos cinco máquinas virtuales corriendo, tendremos cinco sistemas operativos corriendo junto al hypervisor (pieza de software a cargo de la virtualización de la plataforma del host).

Por otro lado, con cinco contenedores, tendremos un escenario en que éstos comparten el sistema operativo, el servidor físico y según sea apropiado, otras dependencias (como binarios y bibliotecas). La Figura 6 ilustra estas diferencias. De esta forma los contenedores presentan las siguientes ventajas sobre las máquinas virtuales:

- Su despliegue es más rápido, puesto que corresponden a procesos y no requieren un proceso de “boot” muy largo.
- Hacen uso eficiente de la infraestructura existente, puesto que tienen un menor tamaño (footprint).
- Comparten recursos del sistema operativo host.

3. Desarrollo de aplicaciones en Previred

El desarrollo, operación y mantención de las aplicaciones en Previred están a cargo de las unidades TI de la organización. Desde el primer semestre de 2017, el desarrollo y evolución de las aplicaciones se realiza siguiendo un proceso de desarrollo ágil, lo cual permitió cambiar la dinámica de construcción, validación y puesta en producción de las aplicaciones, centrándose en la entrega de valor a los usuarios. El ambiente de producción de las aplicaciones está contratado a un proveedor externo, que se encarga de la implantación de las aplicaciones y de mantenerlas disponibles y operativas en el tiempo. El monitoreo de las aplicaciones se realiza utilizando un servicio externo.

En este capítulo se describe y analiza el proceso de desarrollo y operación utilizado en Previred. En la Sección 3.1 se describe el proceso de desarrollo, QA y operación. En la Sección 3.2 se detallan los ambientes y tecnologías utilizadas para apoyar al proceso. La Sección 3.3 documenta el estudio cualitativo que se realizó en Previred para identificar los principales problemas que presenta el proceso y que representan un obstáculo para la organización. La Sección 3.4 concluye el capítulo con un análisis de los problemas identificados.

3.1 Proceso de desarrollo de software actual

El proceso utilizado en Previred para el desarrollo, validación y operación de las aplicaciones está basado en Scrum (Learn About SCRUM, n.d.) y consiste de tres fases principales:

1. **Desarrollo:** Fase de elaboración del software y que se ejecuta en paralelo y con retroalimentación permanente de la fase que sigue. Esta etapa parte con la incepción de un proyecto, producto o necesidad de negocios.
2. **QA** (Quality Assurance, Aseguramiento de la Calidad en español): Fase en la que se prueba y valida el cumplimiento de los requisitos, y la calidad de las piezas de software construidas. Esta fase se ejecuta en paralelo con la fase de Desarrollo y empalma con la fase de Operación tras la aprobación de las áreas de negocio y del usuario.
3. **Operación:** Fase en que el software se pasa a producción y se deja disponible para ser usado por los usuarios finales En esta fase se entregan los artefactos construidos los ingenieros de sistemas quienes se encargan de implantar el software en producción y monitorear su ejecución. En esta fase se genera información que sirve de retroalimentación para la fase de desarrollo, además se informan los *bugs* o fallos en producción.

La Figura 7 muestra el proceso de desarrollo de software de una forma esquemática de alto nivel.

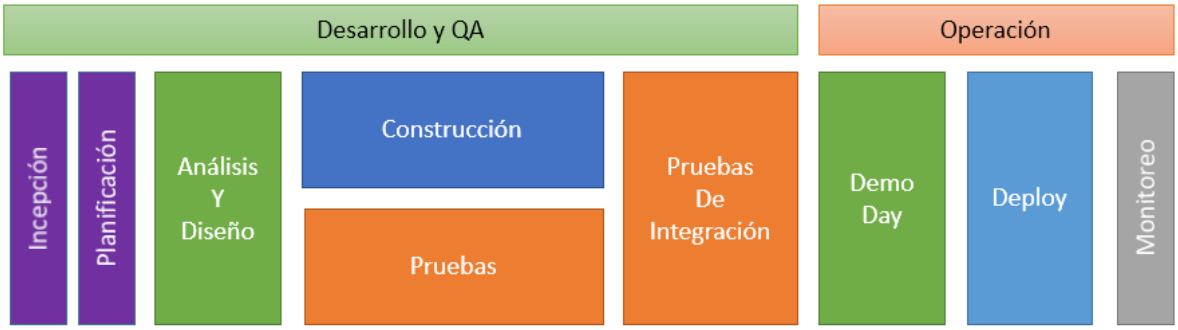

Figura 7. Proceso de desarrollo de software.

A lo largo de estas tres fases se ejecutan las siguientes actividades generales:

- **Incepción**: corresponde a la etapa en que se concibe y especifica, mediante una lista de historias de usuario, el producto o servicio que se construirá. Este producto se construirá parcialmente, para lo cual se desarrollarán "incrementos", que corresponden a los entregables que se pasan a producción al final de un sprint.
- **Planificación**: es un periodo de tiempo en que participan el Product Owner, el Scrum Master, los Stakeholders y los Desarrolladores, donde se define qué historias de usuario se abordarán en el próximo sprint.
- **Análisis/Diseño**: corresponde a la etapa inicial del sprint donde se refinan los requerimientos levantados en la etapa de Incepción. Además, se completan las historias de usuario con sus respectivos criterios de aceptación. Las historias de usuario usualmente se deben subdividir en tareas específicas que se ejecutarán durante el sprint.
- **Construcción**: es la actividad en la que se codifica el software. Durante este periodo se realizan diseñan y codifican pruebas unitarias y se compila el código. Esta actividad se extiende por unos quince días, y durante este periodo se ejecuta una reunión diaria (Daily Meeting), de 15 minutos, en la que cada desarrollador responde tres preguntas: ¿en qué trabajó el día anterior?, ¿qué problema o impedimento tiene para continuar?, y en ¿qué trabajará el día de hoy? Con esto el Scrum Master puede detectar impedimentos y debe tratar de gestionar la solución a estos, para no trabar el avance de los desarrolladores.
- **Pruebas**: donde se ejecutan pruebas sobre el código construido durante el día, se entrega retroalimentación diaria a los desarrolladores de lo construido.
- **Pruebas de Integración:** cuando se finaliza la codificación se integra el software producido por todos los programadores y se realizan pruebas en un ambiente de QA.
- **Demo Day**: es una reunión de 1 a 4 horas donde se demuestra el sistema al Product Owner y los Stakeholders después de las pruebas de integración.
- **Deploy**: corresponde a la entrega de las piezas de software al equipo de sistemas para que las implanten en el ambiente productivo y así la aplicación entregada esté disponible para los usuarios. Mientras se realiza esta fase el equipo se reúne en una retrospectiva, que corresponde a una reunión de una a dos horas donde el equipo de desarrollo evalúa su desempeño y propone mejoras a su labor.

- **Monitoreo:** el software en ejecución es monitoreado permanentemente. En esta etapa se recogen errores y se informan al equipo de desarrollo para que sean corregidos.

Todas estas actividades, salvo Incepción y Planificación, se ejecutan dentro de un sprint, que corresponde a un periodo de un mes. Durante este periodo de tiempo se trabaja en la generación de un producto usable por parte del usuario, ejecutando las actividades descritas.
Los roles que participan en este proceso son:

- **Product Owner:** representa al negocio. Tiene la decisión final sobre las prioridades. Es el que acepta el incremento.
- **Scrum Maste**r: es el facilitador del trabajo del equipo de desarrollo. Gestiona la labor de los desarrolladores. Detecta y gestiona los impedimentos.
- **Stakeholders**: conocen el negocio, saben las necesidades, validan las historias de usuario y los criterios de aceptación. Usan y prueban el sistema.
- **Desarrolladores**: elaboran las historias de usuario y los criterios de aceptación. Construyen el software y los tests automatizados. Previred distingue entre dos tipos de desarrolladores: **programadores** y **testers**, que trabajan en paralelo.
- **Integrador**: es un rol temporal asumido por uno de los programadores, que se encarga de realizar la integración de las distintas ramas git al final de la actividad de construcción.
- **Administrador de la configuración:** es parte del equipo de QA y su rol es mantener los ambientes homologados, configurar las herramientas de soporte del proceso y es el administrador del gestor de código fuente y del orquestador de integración continua.
- **Ingeniero de Sistemas:** es el encargado de administrar y operar los sistemas en ambientes productivos. Coordina y ejecuta las labores de implantación y monitoreo de los sistemas en producción.
- **Ingeniero de redes:** es el responsable de la infraestructura de comunicaciones y quien define las rutas a los servicios a través de las redes a las que está conectado Previred (Internet y WAN de AFP).

Los artefactos que se producen durante el proceso son:

- **Visión del producto:** se almacena en una herramienta de gestión de conocimiento tipo wiki (Atlassian Confluence).
- **Product Backlog:** contiene todas las historias de usuario que se deben implementar en el proyecto. Esto se almacena en la herramienta Atlassian JIRA.
- **Sprint Backlog:** contiene las historias de usuario y tareas que se implementarán en un sprint o incremento.
- **Historias de Usuario:** es una descripción breve de una necesidad del negocio. Estas se almacenan en la herramienta Atlassian JIRA. Estas historias se pueden refinar en varias tareas que también se registran en JIRA.
- **Lista de Impedimentos:** es una lista de las tareas que presentan problemas y requieren gestión por parte del Scrum Master. Si no se gestionan no es posible avanzar con el sprint.

- **Incremento:** es el producto entregado al final del sprint, pudiendo ser este una pieza de software funcional y usable por el usuario.

3.1.1 Desarrollo

Durante la fase de Desarrollo, los programadores pueden trabajar en uno o varios ambientes de desarrollo. Los testers pueden acceder a estos ambientes durante la construcción con el fin de elaborar pruebas, y dar retroalimentación temprana a los programadores de los errores acerca de los errores encontrados.

La Figura 8 describe cómo en la fase de construcción trabajan en conjunto los programadores y los testers. Cada programador trabaja en una rama propia en el repositorio git (feature branch). Usamos como workflow de gestión de git la metodología Git Flow (Introducing GItFlow, s.f.). El proceso de compilación, o building, que convierte el código fuente en código objeto susceptible de ser probado, se ejecuta de manera automática usando la herramienta Maven.

Para esta etapa Previred tiene definida la regla de que por cada dos programadores hay un tester. Durante el día cada uno escribe código o planes de pruebas (de preferencia automatizados) y al final del día el código se despliega en ambientes de desarrollo. Estos ambientes corresponden a servidores Linux o contenedores Docker, que se despliegan en servidores a los que tienen acceso tanto testers como programadores. La implantación en estos ambientes se realiza de forma manual por parte de los programadores. En las últimas horas de la tarde se realizan pruebas y se entrega retroalimentación. Los problemas relevantes son informados al otro día en el Daily Meeting.

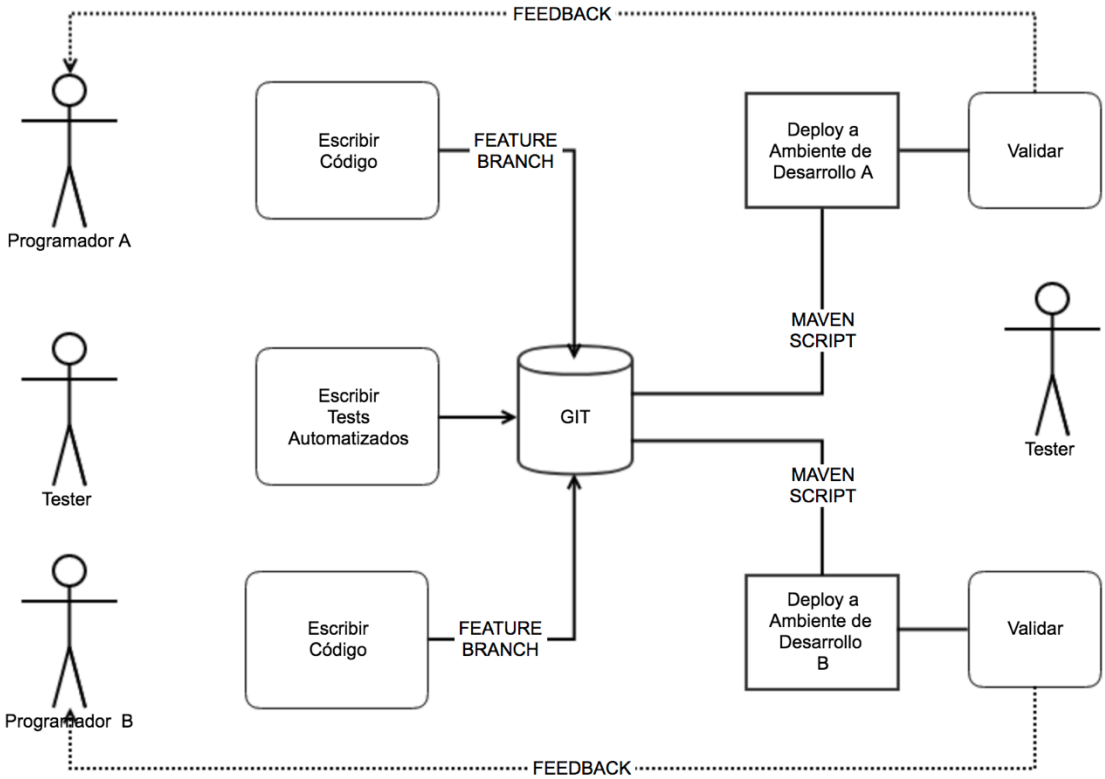

Figura 8. Pruebas durante la actividad de construcción.

3.1.2 QA

La Figura 9 muestra cómo se procede en las Pruebas de Integración, que es donde ocurre el trabajo más importante de la fase de QA.

Antes de realizar las pruebas de integración, se debe consolidar todo el código, elaborado en los distintos *feature-branches*, en una rama base que se llama la *release-branch*. El proceso de consolidación es semiautomático y es supervisado por uno de los programadores, que cumple en ese momento un rol temporal llamado **integrador**. Si hay conflictos, éstos deben ser resueltos en conjunto por el integrador y el programador autor del *feature-branch* que presenta problemas.

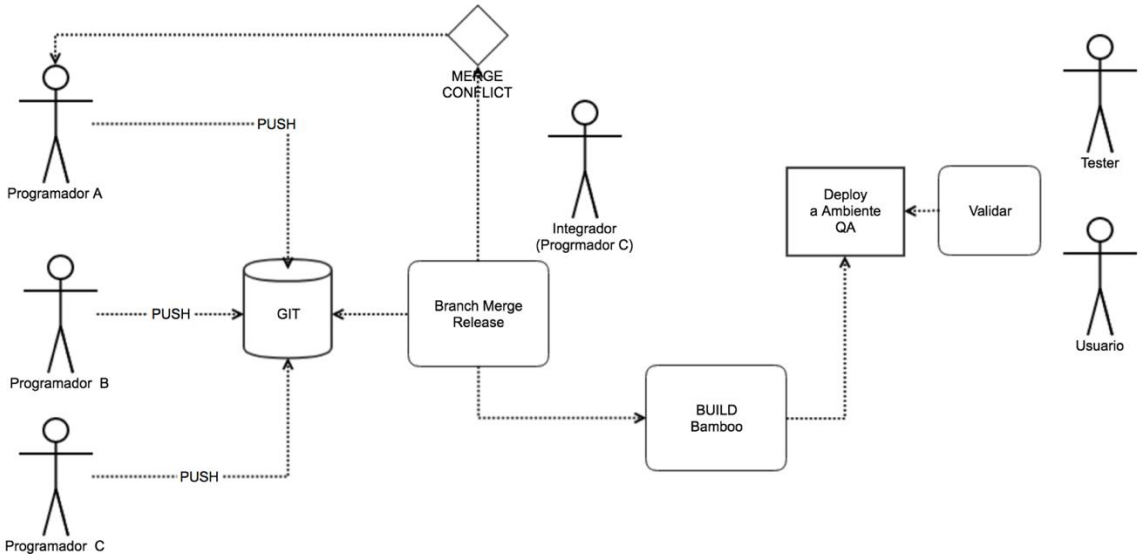

Figura 9. Actividades de QA.

La rama *release-branch* se etiqueta (*tag*) en el repositorio git. Cuando esto ocurre se gatilla un evento de compilación (build) en el servidor de integración continua, Previred utiliza la herramienta Bamboo para estos efectos. Los artefactos construidos quedan almacenados en un repositorio en Bamboo. Este proceso implanta la versión compilada en el ambiente de QA, reemplazando cualquier versión anterior del sistema que se encontrara en este ambiente. A partir de ese momento está disponible para ser validada por los tester y los usuarios. Las pruebas de integración y el Demo Day se realizan en este ambiente de QA.

3.1.3 Operaciones

Paso a producción

El paso a producción es un proceso manual. El administrador de la configuración (parte del equipo de QA) descarga los artefactos generados desde Bamboo y prepara un paquete que es entregado al equipo de sistemas.

Hay dos modos de pasar a producción en la actualidad. Un modo corresponde a la realización de un cambio directo, esto es, detener el servicio, instalar los paquetes en el ambiente de producción y reiniciar el servicio, reemplazando así la versión instalada por una nueva. La Figura 10 ilustra este procedimiento.

El otro modo, ilustrado en la Figura 11, corresponde a la implantación en un ambiente de preparación, denominado pivote, donde se configura e instalan todas las piezas de software e infraestructura necesarias para operar el servicio. Cuando la instalación queda lista en el ambiente pivote, se realizan algunas pruebas básicas en este y posteriormente se ejecuta un re-enrutamiento del servicio para que apunte al ambiente pivote. En ese momento el ambiente pivote pasa a ser el ambiente productivo y viceversa. La instalación usando servidores pivotes es menos utilizada y se usa en proyectos de alta complejidad, pero es el modelo al que se aspira a llevar todas las aplicaciones.

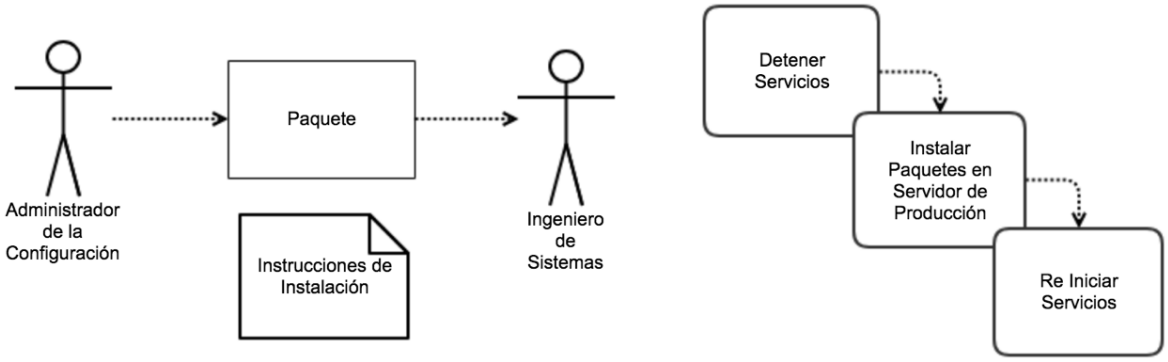

Figura 10. Instalación en producción sin servidor pivote.

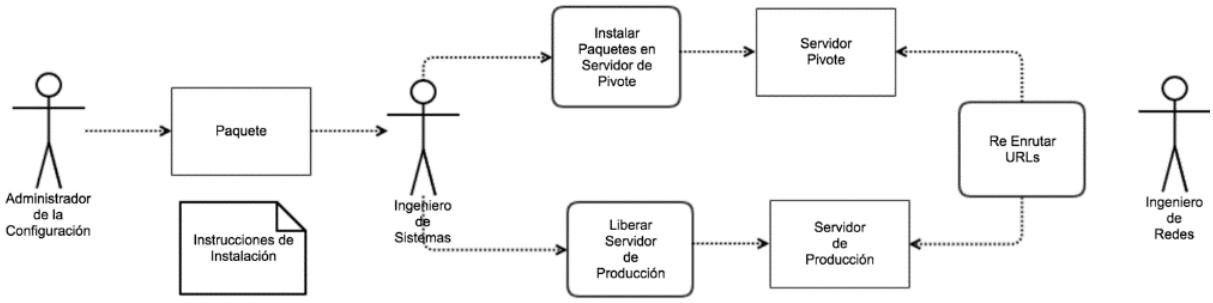

Figura 11. Instalación usando servidores pivotes.

Monitoreo y reporte de errores

Como se aprecia en la Figura 12, en Previred, para los ambientes productivos se monitorean las aplicaciones usando el servicio externo Atentus (S.A., n.d.). Atentus es una empresa nacional creada en 2001, cuyo servicio consiste en acceder periódicamente y desde distintos ISP a una o más URLs expuestas por Previred y registrar las respuestas y el tiempo de respuesta. Esto permite saber si los sitios se encuentran activos y cumplen los niveles de servicio comprometidos.

El monitoreo de la infraestructura del ambiente de producción es realizado utilizando Nagios (Nagios, n.d.). Esta herramienta monitorea parámetros de hardware e infraestructura, como uso de memoria, uso de CPU, estado de los discos, etc.

Por último, las bitácoras de aplicación (logs) son concentrados periódicamente en un servidor al cual pueden acceder los ingenieros de sistemas o el equipo de QA en cualquier momento.

Para el reporte de errores en producción se crean tickets en la herramienta Atlassian JIRA, los que son revisados posteriormente por el equipo de desarrollo y se incorporan al Backlog de desarrollo. Además, se informa a los usuarios de situaciones anómalas mediante correo electrónico.

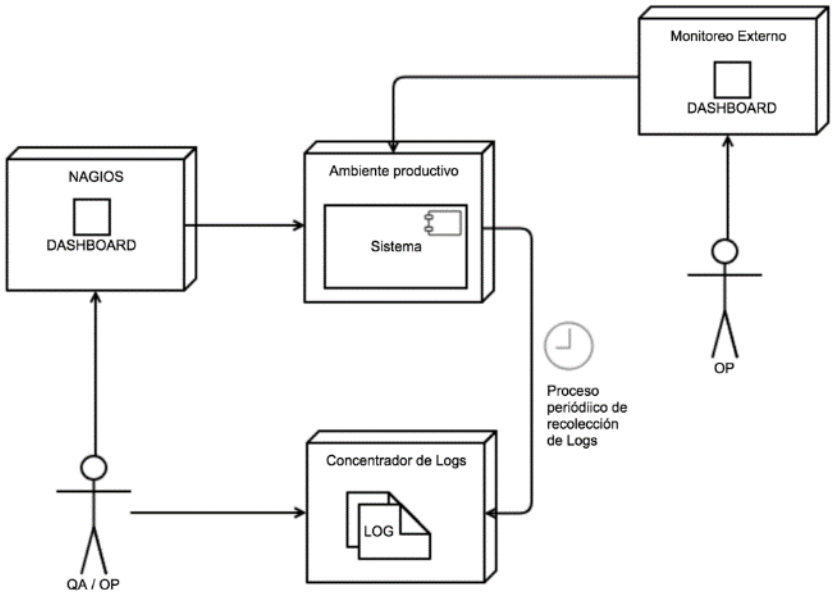

Figura 12. Monitoreo de aplicaciones y concentrador de logs.

3.2 Tecnologías de apoyo al proceso de desarrollo actual

En esta sección se describen las tecnologías, herramientas e infraestructura que soporta todo el proceso de desarrollo en Previred.

3.2.1 Ambientes

Para apoyar el proceso de desarrollo actual, Previred dispone de diversos ambientes, que corresponden a servidores, virtuales o físicos, donde se despliegan los distintos artefactos producidos en cada fase del proceso.

La Figura 13 muestra los ambientes disponibles y los actores que intervienen en el despliegue de los artefactos en cada uno de estos. Estos ambientes se describen a continuación:

- **Ambientes de Desarrollo:** a los que acceden programadores y testers. El despliegue se hace manualmente por parte de los programadores.
- **Ambiente de QA:** donde acceden los testers y también los usuarios (estos últimos no se incluyeron en la figura). El despliegue en este ambiente es semiautomático, apoyado por la herramienta de integración continua Bamboo.

- **Ambiente Pivote:** ocupado cuando se realizan cambios mayores. El despliegue es realizado por los ingenieros de sistemas. En esta fase interviene el ingeniero de redes, quien se encarga de re direccionar el tráfico cuando se realiza el cambio, tal como se describió anteriormente en la Sección 3.1.3.
- **Ambiente Productivo:** que es donde residen las aplicaciones usadas por los usuarios finales.

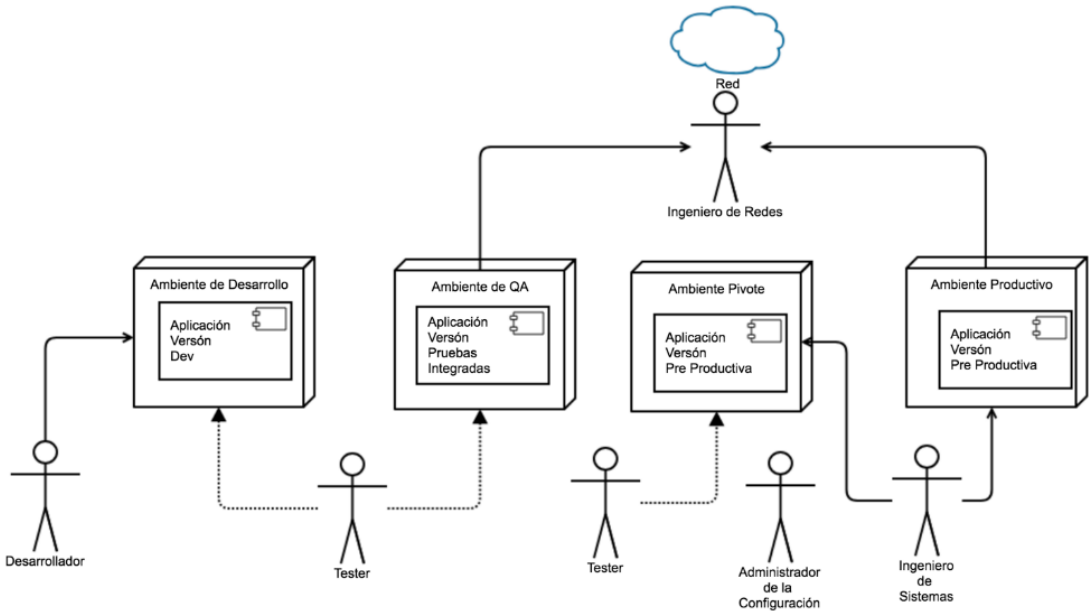

Figura 13. Distintos Ambientes de Trabajo para las fases del proceso de desarrollo.

3.2.2 Herramientas

Tal como se muestra en la Figura 14, el proceso se encuentra apoyado por las siguientes herramientas:

- **Atlassian BitBucket**: gestor de repositorios de código fuente git.
- **SonarQube:** herramienta de análisis de código estático.
- **Nexus**: repositorio de artefactos binarios (bibliotecas).
- **Atlassian Bamboo**: herramienta de Integración Continua.
- **Atlassian Jira y Confluence**: herramientas que apoyan los procesos de gestión de proyectos, gestión de conocimiento y gestión de incidencias.
- **Control de Cambio**: corresponde a un documento con las instrucciones para instalar los artefactos en el ambiente productivo, es elaborado por el Administrador de la Configuración y se almacena en Confluence.

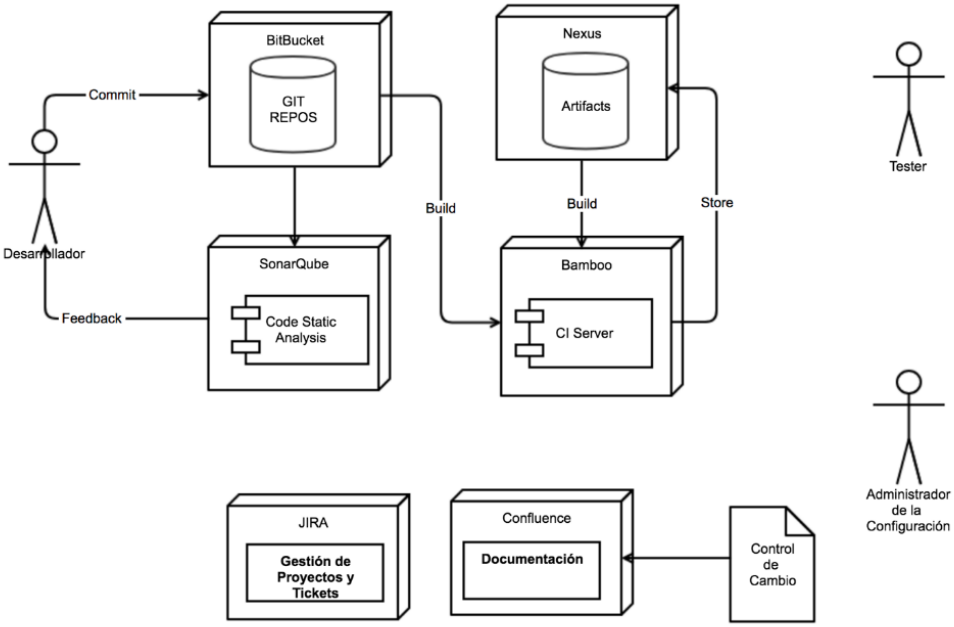

Figura 14. Herramientas que apoyan actualmente al proceso de desarrollo.

3.3 Identificación de problemas en el proceso de desarrollo

El proceso de desarrollo, validación y operación de Previred está siendo utilizado desde 2008 y ha sido aplicado en alrededor de treinta aplicaciones que se encuentran hoy en día en producción. Sin embargo, el proceso presenta problemas que representan un obstáculo para la organización. Con el fin de identificar claramente estos problemas, se realizó un diagnóstico cualitativo con los principales stakeholders dentro de la organización. Este diagnóstico consistió en una entrevista semiestructurada (Cabtree, n.d.) a diferentes actores de la organización, cubriendo todos los roles principales del proceso descritos en la Sección 3.1. El propósito de este diagnóstico fue refinar y caracterizar con mayor precisión el problema a resolver en este trabajo de tesis. Con esto se pudo identificar cuáles son las actividades que pueden ser apoyadas de mejor manera por herramientas y técnicas de DevOps (descritas en la Sección 2.2).

Para ejecutar este diagnóstico primero se elaboró un protocolo de identificación de problemas (descrito en la Sección 3.3.1). Después se ejecutó el protocolo con los principales actores e interesados en esta parte del proceso. Además, con esta actividad se pudo determinar cuáles serían los principales aspectos de la solución construida, mediante la selección de características que tengan un impacto en sus labores diarias (de acuerdo a los resultados detallados en la Sección 3.3.3).

3.3.1 Protocolo de identificación de problemas

Objetivo

El protocolo de identificación de problemas tiene por objetivo identificar el grado de insatisfacción que tiene cada actor con el actual proceso de desarrollo.

Medición

Para medir la insatisfacción, se entrevista a cada actor y se levantan sus principales "dolores", es decir, todos aquellos elementos del actual proceso de implantación del software que consideran que no cumplen con sus necesidades o expectativas, o que les generan problemas o molestias no deseadas, y que por consecuencia no les permiten realizar su labor con fluidez y tranquilidad.

Formato

Para ejecutar el protocolo se realizan entrevistas de quince minutos con representantes de la organización interesados en el proceso. Al inicio de la entrevista se les presenta las fases y actividades del proceso de desarrollo tal como lo ilustra la Figura 7. Posteriormente se les pide que den su diagnóstico de la situación actual y que identifiquen sus principales preocupaciones, molestias o dolores. Después se les pide que elijan a lo más diez aspectos principales y que los prioricen. Para cada preocupación o dolor se les asignó un valor entre 1 y 5, donde 5 indica la máxima prioridad. Después de las entrevistas se clasifican y homologan las preocupaciones, creando una lista unificada.

3.3.2 Ejecución de las entrevistas

Durante un periodo de aproximadamente un mes se realizaron entrevistas a un grupo de personas que representan los siguientes roles dentro de la organización:

- Gerentes y Subgerentes interesados en el proceso.
- Jefaturas asociadas al proceso.
- Ejecutores del proceso.
- Otros interesados en el proceso.

Para cada rol se entrevistaron a las siguientes personas

- Gerencia: Gerente del Área Comercial, Subgerente de Sistemas
- Jefaturas: Jefa del Área de Calidad, Jefa del Área de Desarrollo
- Ejecutores del proceso: dos Programadores, un Arquitecto de Software, un Tester, un Administrador de la Configuración, y un Ingeniero de Sistemas
- Otros: Un Usuario y un Product Owner.

3.3.3 Resultados

Durante las entrevistas se identificaron veintitrés preocupaciones, o “dolores”, que se muestran en la Tabla 1:

#	Preocupación o dolor	Descripción
1	Ambientes no homologados	Los ambientes de desarrollo, testing y producción no presentan las mismas características, produciendo diferencias que generan errores o retrasos en la instalación.
2	Cambios en los ambientes sin aviso	Se ha detectado que hay cambios en los ambientes no informados a ninguna de las partes, los que provocan errores o demora en el diagnóstico de los problemas.
3	Competencias del equipo de Sistemas	Se ha detectado falta de conocimiento o habilidades por parte del equipo que opera la infraestructura, lo que no ayuda a diagnosticar a tiempo y adecuadamente los problemas.
4	Costos	El proceso de desarrollo no es eficiente en costos.
5	Cumplimiento de Plazos	Muchos desarrollos se realizan para cumplir alguna normativa que afecta a nuestros clientes y que tienen plazos estrictos que no pueden ser modificados ni negociados.
6	Data Center lento en aplicar los cambios	A pesar de que se programa con anticipación la ejecución de una tarea de implantación de un cambio, el Data Center toma más tiempo del necesario en la aplicación del mismo.
7	Data Center poco prolijo al aplicar cambios	Hemos tenido malas experiencias con el proceso de instalación de los cambios en ambiente productivo por parte del proveedor de Data Center.
8	Diagnósticos de Fallas inadecuados	Los diagnósticos que se hacen de los fallos no corresponden a la causa real, o no contienen los detalles necesarios para que el equipo de desarrollo pueda abordarlos.
9	Errores colaterales	Existen muchos errores colaterales, es decir, fallos en partes de los sistemas que no deberían verse afectadas por el cambio a realizar.
10	Duración de las etapas del paso a producción	Las etapas de promoción desde ambientes de QA a Producción requieren, por parte del Data Center, que sean entregadas con 48 horas de anticipación, además se deben programar con una semana de anticipación. Además su ejecución puede tomar varias horas.

11	Esfuerzo necesario para validar cambios	Durante la fase de testing se producen demasiadas iteraciones entre el equipo de desarrollo y testing.
12	Excesivo Papeleo	El proceso de promoción de piezas de software a los distintos ambientes requiere documentación excesiva elaborada por necesidades de control y delimitación de responsabilidades.
13	Falta clarificar alcance de los cambios	Tanto usuarios como el equipo de QA no tiene la información adecuada para entender los alcances de un cambio que se está recibiendo.
14	Falta de bitácoras detalladas	Las bitácoras, o logs, que generan las aplicaciones y la infraestructura no son adecuadas y no ayudan al diagnóstico de problemas.
15	Falta de tiempo para las tareas	Hay una presión para cumplir con plazos, y no se dispone del tiempo adecuado para ejecutar las tareas adicionales necesarias en cada actividad.
16	Falta infraestructura	No se dispone de suficientes recursos de infraestructura para construir ambientes de desarrollo o testing que satisfagan las necesidades del negocio.
17	Monitoreo inadecuado de los ambientes productivos	Muchas fallas se detectan ante reclamos de clientes, en vez de ser detectadas por herramientas de monitoreo, las que si bien existen, no permiten tomar medidas oportunas.
18	Poco dominio de la arquitectura	Se percibe que tanto el equipo de sistemas como el desarrollo no dominan adecuadamente la arquitectura de software y hardware que soporta las aplicaciones, lo que ralentiza el proceso de diagnóstico.
19	Prolijidad de la entrega por parte de los desarrolladores	Falta de pruebas unitarias, y de borde por parte de los desarrolladores antes de entregar, lo que aumenta los ciclos de prueba.
20	Tiempo empleado en corregir una falla	El tiempo de detección, análisis y corrección de una falla no es aceptable dadas las necesidades del negocio.
21	Tiempos de Aprobación	Durante la promoción a distintos ambientes, el tiempo de aprobación por parte de los usuarios es excesivo, lo que impide liberar recursos y tareas.
22	Tiempos de Aprovisionamiento de infraestructura	Obtener recursos de infraestructura puede tomar varias semanas.

23	Utilidad de lo entregado	Es importante que los cambios entregados sean relevantes para las necesidades del negocio.

Tabla 1. Preocupaciones o dolores hallados en las entrevistas.

#	Problema identificado	Votos	Gerencia	Usuario	Subgerencia Sistemas	Jefatura Desarrollo	Jefatura Calidad	Arquitecto	Dev	Tester	Ingeniero Sistemas	Administrador de la Configuración	Votación: Peso Total
1	Ambientes no homologados	5			5		4	3	3			4	19
15	Falta de tiempo para las tareas	4					5		5	5	3		18
9	Disminuir errores colaterales	4	3	5		5				3			16
12	Excesivo Papelo	4			3				4	4	5		16
7	Datacenter poco prolijo al aplicar cambios	3		5							5	5	15
20	Tiempo empleado en corregir una falla	4	4	4		4			3				15
8	Diagnósticos de Fallas inadecuados	3				5		5				5	15
17	Monitoreo Inadecuado ambientes productivos	4		4			3		5		3		15
16	Falta infraestructura	4							3	3	5	3	14
21	Tiempos de Aprobación	3					5			5		3	13
22	Tiempos de Aprovisionamiento de infraestructura	3			4			3				5	12
2	Cambios en los ambientes sin aviso	3					3			3		5	11
14	Falta de bitácoras detalladas	3						5			3	3	11
18	Poco dominio de la arquitectura	3		3		3		5					11
4	Costos	2	5		5								10
23	Utilidad de lo entregado	2	4	5									9
5	Cumplimiento de Plazos	2	5	3									8
10	Duración de las etapas del paso a producción	2		5		3							8
11	Esfuerzo necesario para validar cambios	2					5			3			8
19	Prolijidad de la entrega por parte de los desarrolladores	2					4	4					8
3	Competencias del equipo de sistemas	2				3		4					7
13	Falta clarificar alcance de los cambios	2					3			4			7
6	Datacenter lento en aplicar cambios	3		1		3					3		7

Tabla 2. Votación en el protocolo.

Estos dolores fueron votados por cada uno de los participantes, con el fin de encontrar el peso de cada uno. La Tabla 2 recoge el resultado de esta votación.

Cada entrevistado asignó un valor entre 1 y 5 a cada uno de los dolores que declaró. Un valor 5 es que el dolor tiene la mayor importancia. La columna final contiene la suma de la votación. El resultado en la Tabla 2 está ordenado desde la votación total mayor a la votación total menor.

3.4 Análisis de la situación actual

En base a los resultados obtenidos de la ejecución del protocolo, se realizó un análisis experto para priorizar los dolores identificados y enfocar la solución hacia la cobertura más amplia de estas preocupaciones. Esta sección explica estos *drivers de decisión*.

3.4.1 Técnicas y herramientas DevOps y su impacto en cada preocupación

La Tabla 3 muestra el nivel de impacto que cada técnica o herramienta DevOps tiene sobre cada uno de los dolores identificados previamente. Esta tabla se elaboró mediante consulta experta, en la que participaron un arquitecto de software, la jefa de desarrollo, la jefa de control de calidad y el autor de esta tesis (subgerente de tecnología).

Cada celda en la tabla puede tener un valor entre 1 y 5, donde 1 indica muy poco relevante, y 5 muy relevante. Una celda en blanco es porque los participantes consideran que la técnica o herramienta no aplica o no tiene impacto alguno con la preocupación o dolor identificado.

La columna Cobertura es la suma de los puntajes asignados en cada dimensión. Al pie de la tabla se calcula la suma por cada columna y su promedio, siendo este último un indicador de la utilidad de la técnica o herramienta en la solución de los dolores identificados.

La selección de las técnicas o herramientas de DevOps de mayor impacto se realizó tomando como punto de corte un valor de 3.0 o superior en el promedio. Para el conjunto de problemas identificados, éstas son:

- Gestión del Conocimiento.
- Generación y Concentración de Logs.
- Testing Automatizado.
- Implantación (Deployment) automatizado.
- Plataforma como servicio.
- Cambio en el proceso.

#	Problema identificado	Votos	Tecnicas/Herramientas DevOps								
			Plataforma como servicio	Testing automatizado	Gestión de configuración de plataforma	Deploy automatizado	Gestión de conocimiento	Generación y concentración de logs	Cambio en el proceso	Análisis de código	Cobertura Técnicas/Herramientas DevOps
4	Costos	2	3	4	4	4	0	0	4	0	19
11	Esfuerzo necesario para validar cambios	2	1	5	1	3	3	3	3	0	19
9	Disminuir errores colaterales	4	0	5	2	2	3	0	3	3	18
20	Tiempo empleado en corregir una falla	4	1	2	0	3	4	4	2	2	18
5	Cumplimiento de Plazos	2	4	2	4	3	0	0	5	0	18
10	Duración de las etapas del paso a producción	2	4	3	1	5	0	0	5	0	18
7	Datacenter poco prolijo al aplicar cambios	3	4	0	4	5	2	1	1	0	17
8	Diagnósticos de Fallas inadecuados	3	0	2	2	0	5	4	0	2	15
19	Prolijidad de la entrega por parte de los desarrolladores	2	0	5	0	0	3	0	2	5	15
1	Ambientes no homologados	5	5	4	3	2	0	0	0	0	14
3	Competencias del equipo de sistemas	2	2	0	2	2	5	0	3	0	14
6	Datacenter lento en aplicar cambios	3	5	0	2	5	0	0	2	0	14
12	Excesivo Papelo	4	2	0	3	5	0	0	3	0	13
23	Utilidad de lo entregado	2	0	3	0	0	5	0	5	0	13
2	Cambios en los ambientes sin aviso	3	5	0	5	0	0	0	2	0	12
14	Falta de bitácoras detalladas	3	2	0	2	0	0	5	0	2	11
18	Poco dominio de la arquitectura	3	2	0	2	0	5	0	1	1	11
21	Tiempos de Aprobación	3	0	0	0	2	3	0	5	0	10
13	Falta clarificar alcance de los cambios	2	0	0	0	0	5	0	5	0	10
15	Falta de tiempo para las tareas	4	1	3	0	3	0	0	2	0	9
16	Falta infraestructura	4	5	0	3	0	0	0	0	0	8
17	Monitoreo Inadecuado ambientes productivos	4	3	0	0	0	0	4	0	0	7
22	Tiempos de Aprovisionamiento de infraestructura	3	5	0	2	0	0	0	0	0	7
			54	38	42	44	43	21	53	15	
			Suma								
			3,2	3,5	2,6	3,4	3,9	3,5	3,1	2,5	
			Promedio								
			UTILIDAD								

Tabla 3. Utilidad de cada herramienta o técnica DevOps pertinente.

3.4.2 Impacto de los problemas en las fases del proceso de desarrollo

Es relevante identificar en qué grado cada etapa del proceso de desarrollo se ve afectada por cada uno de los dolores o preocupaciones identificados. Realizando un análisis similar al de la Sección 3.4.1, se identificaron que las principales actividades que se ven impactadas por los problemas levantados en el protocolo son: Construcción, Pruebas, Deploy, Monitoreo y Demo Day.

Esto se determinó a partir de los valores obtenidos por la Tabla 4, tomando como punto de corte un valor promedio de 4.25 o superior.

Cada celda en la Tabla 4 corresponde a una medida del impacto que cada dolor o preocupación para cada actividad del proceso de desarrollo. En este caso se incluyeron todas las actividades del proceso, no sólo las que se abordan con una solución basada en técnicas de DevOps. La tabla fue elaborada por el mismo equipo mencionado en la Sección 3.4.1.

La columna “Impacto del dolor en el ciclo de vida” es la suma de los valores en cada fila para el dolor respectivo. La fila Promedio representa el grado en que los dolores afectan a cada actividad del proceso.

			Ciclo vida del Software									
#	Problema Identificado	Votos	**Planificación**	**Análisis**	**Codificación**	**Testing**	**Entrega**	**Implantación**	**Verficación**	**Monitoreo**	**Aprobación**	**Impacto del dolor en el ciclo de vida**
7	DC poco prolijo al aplicar cambios	3	0	3	3	3	5	5	5	4	5	38
5	Cumplimiento de Plazos	2	5	3	5	5	5	5	5	0	5	38
13	Falta clarificar alcance de los cambios	2	3	5	5	5	3	3	5	0	5	37
23	Utilidad de lo entregado	2	3	5	5	5	0	0	2	3	5	33
4	Costos	2	4	5	5	5	0	5	0	3	0	32
10	Duración de las etapas del paso a producción	2	1	4	5	5	3	3	3	0	3	30
2	Cambios en los ambientes sin aviso	3	0	0	3	3	5	5	5	5	0	30
1	Ambientes no homologados	5	0	0	4	5	5	5	4	0	3	29
16	Falta infraestructura	4	0	0	3	5	5	5	3	3	0	29
3	Competencias del equipo Ops	2	0	3	3	5	0	5	3	5	0	29
9	Disminuir errores colaterales	4	4	4	5	5	0	0	0	5	0	28
11	Esfuerzo necesario para validar cambios	2	3	0	0	5	0	0	5	5	5	28
22	Tiempos de Aprovisionamiento de infraestructura	3	3	0	0	0	0	5	5	5	5	28
21	Tiempos de Aprobación	3	3	3	3	4	3	3	3	0	5	27
20	Tiempo empleado en corregir una falla	4	5	5	5	5	0	0	0	0	0	25
12	Excesivo Papelo	4	3	0	3	3	5	5	0	0	3	25
8	Diagnósticos de Fallas inadecuados	3	0	4	5	5	0	0	0	5	0	24
15	Falta de tiempo para las tareas	4	5	3	5	5	0	3	0	0	3	24
14	Falta de bitácoras detalladas	3	0	5	5	5	0	0	0	5	0	23
19	Prolijidad de la entrega por parte de los devs	2	0	0	5	5	0	0	5	0	5	20
6	DC lento en aplicar cambios	3	0	0	0	0	3	5	0	5	0	18
18	Poco dominio de la arquitectura	3	0	5	5	3	0	0	0	3	0	16
17	Monitoreo Inadecuado ambientes productivos	4	0	3	3	0	0	0	0	5	0	16
			42	60	85	91	42	62	53	61	52	
			Medida del dolor en cada fase									
			3.5	4	4.25	4.55	4.2	4.43	4.1	4.36	4.3	

Tabla 4. Medida de cada dolor en las actividades del proceso de desarrollo.

3.4.3 Priorización

Considerando el análisis anterior, la información se consolidó en la Tabla 5.

#	Problema Identificado	Votación: Peso Total	Cobertura Técnicas/Herramientas DevOps	Peso x Cobertura	Planificación	Impacto del dolor en el ciclo de vida	Prioridad x Impacto	Peso x Cobertura x Aprobación
7	Data Center poco prolijo al aplicar cambios	15	17	255	0	33	495	8415
9	Disminuir errores colaterales	16	18	288	4	28	448	8064
20	Tiempo empleado en corregir una falla	15	18	270	5	25	375	6750
1	Ambientes no homologados	19	14	266	0	24	456	6384
4	Costos del proceso de desarrollo	10	19	190	4	32	320	6080
8	Diagnósticos de Fallas inadecuados	15	15	225	0	24	360	5400
5	Cumplimiento de Plazos	8	18	144	5	33	264	4752
11	Esfuerzo necesario para validar cambios	8	19	152	3	28	224	4256
12	Excesivo Papelo	16	13	208	3	20	320	4160
10	Duración de las etapas del paso a producción	8	18	144	1	27	216	3888
15	Falta de tiempo para las tareas	18	9	162	5	24	432	3888
23	Utilidad de lo entregado	9	13	117	3	33	297	3861
2	Cambios en los ambientes sin aviso	11	12	132	0	25	275	3300
21	Tiempos de Aprobación	13	10	130	3	24	312	3120
3	Competencias del equipo de Sistema	7	14	98	0	29	203	2842
14	Falta de bitácoras detalladas	11	11	121	0	23	253	2783
16	Falta infraestructura	14	8	112	0	24	336	2688
19	Prolijidad de la entrega por parte de los desarrolladores	8	15	120	0	20	160	2400
13	Falta clarificar alcance de los cambios	7	10	70	3	34	238	2380
22	Tiempos de Aprovisionamiento de infraestructura	12	7	84	3	28	336	2352
18	Poco dominio de la arquitectura	11	11	121	0	16	176	1936
17	Monitoreo Inadecuado ambientes productivos	15	7	105	0	16	240	1680
6	DC lento en aplicar cambios	7	14	98	0	15	105	1470

Tabla 5. Resumen y priorización de los problemas identificados.

Para obtener la prioridad se calculó una columna adicional que corresponde a la fórmula: Votación x Cobertura x Impacto; la tabla está ordenada por esta columna. Con esta puntuación y tras ordenar de mayor a menor por el puntaje obtenido, lo que se obtiene es que los problemas principales a abordar son:

1. Data Center poco prolijo al aplicar cambios.
2. Disminuir errores colaterales.
3. Tiempo empleado en corregir una falla.
4. Ambientes no homologados.
5. Costos del proceso de desarrollo.
6. Diagnóstico de Fallas Inadecuados.
7. Cumplimiento de Plazos.
8. Esfuerzo necesario para validar cambios.
9. Excesivo Papeleo.
10. Duración de las etapas del paso a producción.

Este análisis nos da un marco de referencia para decidir qué técnicas de DevOps utilizar en la solución (Capítulo 4) y para realizar la validación de la solución (Capítulo 5).

4. Arquitectura de la Plataforma

Para resolver los dolores identificados en el Capítulo 3 es necesario refinar el proceso de desarrollo de Previred y proveer una plataforma tecnológica que promueva la automatización. En este capítulo se describe la plataforma construida, los ajustes al proceso de desarrollo y el soporte tecnológico de la misma.

La Sección 4.1 describe el contexto, los objetivos y los principios que gobiernan las decisiones tomadas en el refinamiento del proceso y en la definición de la arquitectura de la plataforma. La Sección 4.2 describe el proceso de desarrollo y operación refinado. La Sección 4.3 documenta la plataforma tecnológica que da soporte al nuevo proceso y las herramientas seleccionadas para dicha plataforma.

4.1 Contexto, objetivos y principios

En los últimos cinco años Previred ha implementado diversos servicios en la industria de la seguridad social chilena. Estos servicios si bien no son tan masivos, como el sitio de recaudación www.previred.com, son críticos en cada uno de los nichos de negocio en los que operan. Las exigencias de nuestros clientes son cada vez mayores, entre las que se encuentran:

- **Servicios críticos de impacto nacional, o a nivel industria.** Previred opera servicios transaccionales usados de forma transversal por todas las AFP. Si estos servicios están abajo, se impactan los niveles de servicio de toda la industria. Incluso, algunos servicios en Previred tienen impacto a nivel nacional, como el pago de cotizaciones, traspaso de fondos y de cuentas, evaluación de créditos en línea, entre otros.
- **Cambios normativos.** Muchos de los servicios provistos por Previred se construyen para cumplir con las normativas emanadas por la Superintendencia de Pensiones. Estas normativas son siempre revisadas por la autoridad, la que emite cambios que deben ser implementados muchas veces en plazos muy acotados.
- **Alta disponibilidad.** Las aplicaciones deben estar disponibles no sólo en horarios hábiles, sino que en muchos casos se espera que las aplicaciones estén disponibles en modalidad cercana a 24x7. Hay cada vez menos tolerancia ante caídas de los sistemas, y se espera que, ante fallos, se restauren los servicios en minutos.

Esto nos da una idea del contexto en que se desarrollan las aplicaciones en Previred, todo lo cual plantea algunos desafíos particulares. Dado esto, es que Previred tiene incorporada una cultura de mejora continua de procesos, gracias a la cual es posible abordar un cambio tan relevante como el descrito en la presente tesis.

4.1.1 Objetivos

La organización se encuentra al inicio de un plan estratégico a cuatro años (2018-2021), en cuya visión se establece el desafío de *"ser el más ágil y eficiente socio estratégico del Sistema de Seguridad Social de Chile, otorgando, más y mejores servicios a los trabajadores, empleadores e instituciones que lo componen"*. Este desafío se traduce en

un mayor impulso por parte de las áreas de negocios para generar soluciones con mayor rapidez.

Es en este contexto que la solución a implementar debe considerar los siguientes objetivos de negocio definidos por las áreas de tecnología:

- G1: Adaptarse al proceso de desarrollo ágil adoptado por Previred.
- G2: Permitir reducir el tiempo de entrega de resultados concretos a los usuarios finales (en otras palabras, reducir el *"time to market"*)
- G3: Mantener la reputación de Previred en la industria de entregar soluciones seguras, innovadoras y de calidad.
- G4: Evitar la interrupción de los servicios, y reducir al máximo posible el tiempo de solución de problemas.
- G5: Evitar o reducir al mínimo el fallo humano en la puesta en producción de servicios.
- G6: Liberar tiempo de las personas que integran el área de tecnología.

4.1.2 Principios técnicos y de negocio

Con el fin de cumplir con los objetivos de negocio planteados, definimos los siguientes principios de negocio que debe respetar nuestro diseño arquitectónico:

- B1: El proceso debe ser transparente, todos los stakeholders pueden observar en qué etapa del proceso se encuentra el desarrollo e implantación de un nuevo requerimiento.
- B2: La solución debe permitir que cualquier falla en el proceso sea detectada lo más pronto posible y se debe informar a los stakeholders de esto.
- B3: Toda operación repetitiva debe automatizarse.
- B4: No debe haber disrupción apreciable de los servicios que están operando ante la aplicación de cambios en los mismos.

Estos principios implican que se deben garantizar los siguientes principios tecnológicos:

- T1: Las piezas de software construidas por el equipo de desarrollo de Previred deben ser generadas mediante procesos automatizados y repetibles (building automático de piezas).
- T2: Los ambientes donde se implanta el software deben ser inmutables en cuanto a la infraestructura básica que sostiene la operación (sistemas operativos, versiones de máquinas virtuales, intérpretes, bases de datos, software base, etc.) aunque sus capacidades pueden variar (memoria RAM disponible, cantidad de procesadores, espacio en disco, velocidad, etc.).
- T3: La configuración de las aplicaciones debe ser paramétrica y debe ser ajustada según los distintos ambientes a los que se promueve el software construido.
- T4: La configuración de las aplicaciones debe respetar las normas de seguridad de Previred (por ejemplo, las credenciales de acceso a una base de datos no deben ser accesible por personal no autorizado, y deben estar disponibles para la aplicación en el momento de despliegue).
- T5: Debe ser posible volver atrás una versión del software ante un fallo crítico.

4.2 El nuevo proceso

Para lograr los objetivos planteados y resolver los dolores organizacionales, es necesario refinar el proceso de desarrollo y operación de Previred. En la Sección 3.1 y en específico en la Figura 7 (replicada en la Figura 15 para facilitar la lectura), se detalla el proceso de desarrollo y operación de Previred, previo a la adopción de las técnicas de entrega continua impulsadas por este trabajo de tesis. La Figura 16 muestra el nuevo proceso de desarrollo implantado. Los cambios realizados al proceso se deben principalmente por la introducción de técnicas y herramientas de automatización, logrando así tener un pipeline de entrega continua.

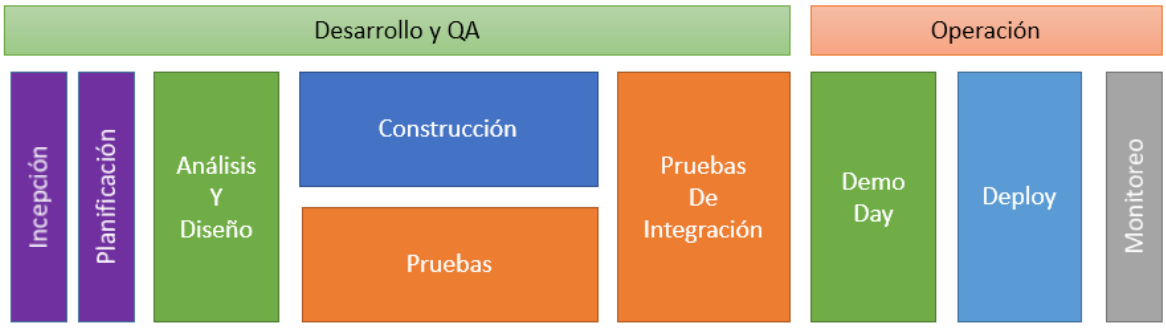

Figura 15. Proceso de desarrollo actual (copia de la Figura 7).

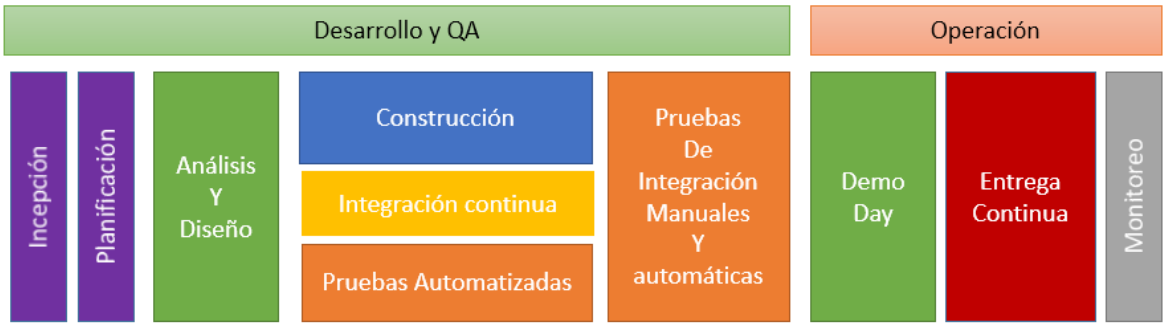

Figura 16. Proceso de desarrollo refinado.

Uno de los cambios realizados al proceso fue en la actividad de Construcción (en azul en la Figura 16) que consistió en la introducción de técnicas de integración continua. En el nuevo proceso, durante la implementación de los sistemas, los desarrolladores deben aumentar la frecuencia con la que liberan a los otros desarrolladores los cambios que estén realizando. De esta forma, la integración del código fuente se facilita, ya que, aunque ocurre con mayor frecuencia, lo hace sobre cambios más pequeños, evitando así la divergencia en el código fuente de cada desarrollador y simplificando la resolución de inconsistencias (al momento de hacer merge). Además, esto implicó que los desarrolladores debieron adoptar herramientas que permiten el build automatizado de las componentes de software, además de elaborar scripts que luego son ejecutados en la actividad de integración continua. Estas decisiones están alineadas con los objetivos G1 (adaptación al proceso de desarrollo ágil de Previred) y G2 (reducir “time to market”).

También esto nos permite garantizar la adherencia al principio B2, que nos exige detectar cualquier falla lo más pronto posible.

Además, el nuevo proceso agrega formalmente la actividad de integración continua (en amarillo en la Figura 16), para reflejar el hecho de que, ante cada integración del código fuente que queda disponible en el repositorio, se realiza la actividad de compilar (build), validar la adhesión a estándares de codificación y buenas prácticas (análisis de código) y verificar la correctitud de los sistemas (ejecución de pruebas). Esta actividad se realiza en forma automatizada, ya que es una tarea repetitiva, cumpliendo con los principios B3 y T1. Con esto garantizamos los objetivos G3 (reputación de entregar soluciones seguras, innovadoras y de calidad) y G5 (evitar o reducir el fallo humano).

La ejecución de pruebas manuales requiere mucho esfuerzo y es propensa a errores humanos. En el proceso anterior, se contaba con dos actividades de pruebas: pruebas funcionales y pruebas de integración (en naranja en la Figura 15). En el nuevo proceso, se buscó reemplazar estas actividades manuales por una automática. Para ello, el equipo de desarrollo y el equipo de QA se encargan de construir (implementar) pruebas unitarias y de integración automatizadas (en naranja en la Figura 16), las cuales son ejecutadas automáticamente en la actividad de integración continua (en amarillo en la Figura 16). Como consecuencia, hay un esfuerzo considerable en la construcción de las pruebas automatizadas. Sin embargo, este esfuerzo se amortiza en el tiempo ya que la ejecución de las pruebas manuales, que también requieren mucho esfuerzo, se reduce significativamente. Con esto nos alineamos al objetivo G6 (liberar tiempo de las personas). El nuevo proceso exige que las pruebas unitarias sean automatizadas en su totalidad. En cambio, también reconoce que la automatización de las pruebas de integración es un proceso paulatino, y que podría no alcanzar una cobertura del 100% de las pruebas, por lo que la actividad de pruebas de integración contempla la ejecución de las que fueron automatizadas y la ejecución manual de las que no lo fueron.

En el proceso de Previred, la actividad de puesta en producción era una actividad manual llamada Deploy (en celeste en la Figura 15). Esta actividad requería especificar una Solicitud de Cambio, consistente en un instructivo para los operadores del Data Center, que indicaba todas las modificaciones que debían realizarse en la plataforma productiva. Como se capturó en la Sección 3.3 uno de los dolores principales es que el Data Center demora en la ejecución de estos cambios, que se cometían errores de interpretación de los instructivos, y que, de haber errores, no era posible volver atrás. En el nuevo proceso se reemplazó la actividad de Deploy por la de entrega continua (en rojo en la Figura 16), la cual automatiza la puesta en producción de los sistemas, sin necesidad de intervención manual. De esta manera, el proceso de puesta en producción es repetible, de bajo costo, y puede volverse atrás ante un caso de fallo mediante la implantación automatizada de la versión anterior. Con esto se satisface el principio B4 que nos exige evitar la disrupción de los servicios ante la aplicación de cambios, y se alinea con los objetivos G2 (reducir el “time to market”) y G4 (evitar la interrupción de los servicios). Para que esta actividad sea posible, es necesario contar con una plataforma de producción que pueda gobernarse mediante scripts, esto es, una plataforma como servicio. La arquitectura descrita en la Sección 4.3 muestra cómo se logra esta capacidad mediante el uso de una

plataforma de contenedores y una herramienta de orquestación de la puesta en producción.

Tal como se discutió en la Sección 2.2, el nivel de entrega continua difiere del nivel de implantación continua en que, en el primero, la puesta en producción es manual y en el segundo automática. En Previred, alcanzar el nivel de implantación continua no es factible ya que la actividad del Demo Day (en verde en la Figura 15) es relevante, ya que consiste en la prueba de aceptación de los interesados, la cual es requerida para proceder a la implantación. Sin embargo, en el nuevo proceso y con la nueva plataforma, la puesta en producción sí es automatizada en el sentido de que se ejecuta automáticamente sin intervención manual, pero no es automática la decisión de pasar a producción. Por este motivo declaramos que el nuevo proceso es de entrega continua, aunque también hayamos logrado que la actividad de implantación esté automatizada.

El uso de una plataforma basada en contenedores nos permite satisfacer los principales principios tecnológicos propuestos. En particular, el principio T2 que nos exige generar ambientes inmutables y T3 que nos pide que la configuración de estos ambientes sea paramétrica.

Por último, con estos cambios nos hacemos cargo de gran parte de los dolores destacados en la Sección 3.3.3:

- “Data Center poco prolijo al aplicar cambios”, con la entrega continua.
- “Errores colaterales”, al incorporar pruebas automatizadas a lo largo del proceso de construcción.
- “Ambientes no homologados”, con el uso de una plataforma basada en contenedores que asegura ambientes inmutables
- “Tiempo empleado en corregir una falla”, al automatizar las tareas de building, testing e implantación de las soluciones.
- “Excesivo papeleo”, al dejar de utilizar los documentos de control de cambio reemplazándolos por la automatización de la puesta en producción.
- “Duración de las etapas del paso a producción”, pues al automatizar y al aplicar entrega continua se pueden hacer entregas más frecuentes y de menor tamaño.
- “Esfuerzo necesario para validar cambios”, gracias a la automatización y el uso de las técnicas de entrega continua.

4.3 Tecnología

4.3.1 Plataforma DevOps

La arquitectura de la plataforma DevOps construida se basa en lo descrito en la Sección 2 y siguiendo la estructura propuesta en [1]. Un esquema de la arquitectura y sus principales componentes se muestra en la Figura 17.

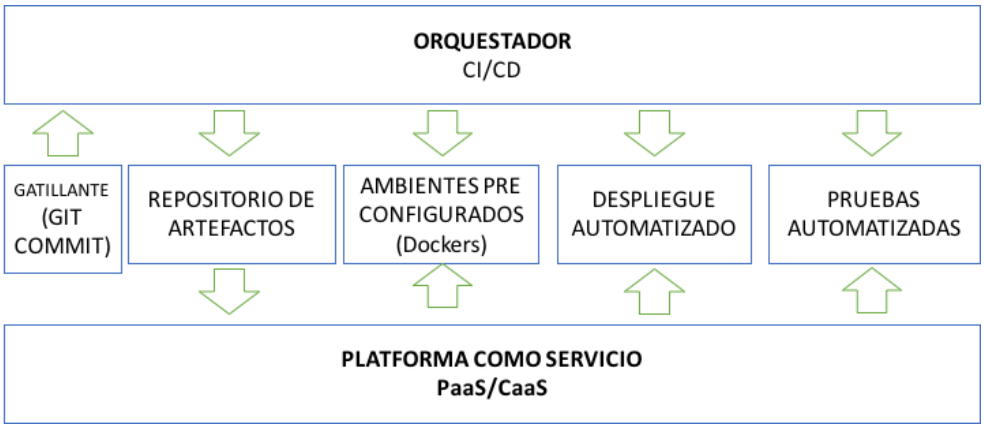

Figura 17. Plataforma de DevOps propuesta.

Dos elementos esenciales, que son la base de la arquitectura, son el Orquestador de Integración continua y la Plataforma de Gestión de Contenedores.

En la Figura 18 se muestra de forma más detallada la solución implementada, destacando sus principales componentes, junto con los actores que hacen uso de la misma.

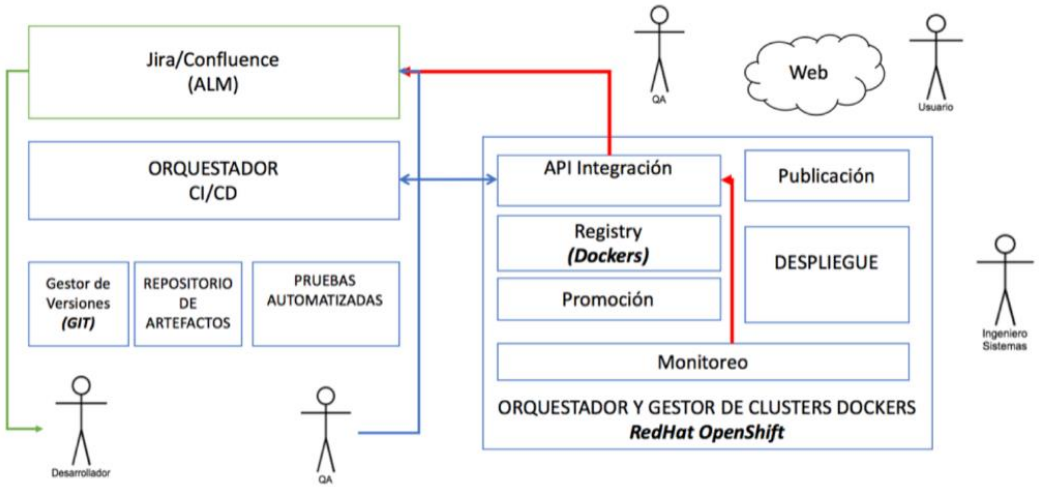

Figura 18. Vista más detallada de la arquitectura propuesta que incluye los roles que interactúan con la misma.

En el diagrama se aprecian los distintos componentes de la arquitectura. Se ha incorporado la suite de ALM, puesto que se espera integrarla también en el proyecto. Esta suite ALM, basada en Jira y Confluence, es donde se ingresan los requisitos que deben ser desarrollados. Toda la documentación, que incluye las especificaciones y diseños, se guarda en Confluence.

Las tareas específicas que se agrupan en historias de usuario, que se construyen en sprints de acuerdo a la metodología Scrum, son abordadas por los desarrolladores. El rol del desarrollador es producir código que se incorpora en el gestor de versiones. Las distintas características (features) desarrolladas se agrupan en un release. Al liberarse un reléase, el orquestador de integración continua prepara un nuevo artefacto entregable, que es almacenado en el repositorio de artefactos.

El orquestador ejecuta las pruebas unitarias automatizadas sobre el código y genera reportes, los que son inspeccionados por el equipo QA. Si hay errores, éstos se ingresan en Jira y se inicia un ciclo de depuración. Cuando el release se considera terminado, se promueve al ambiente QA donde se pueden realizar pruebas de estrés y los tests de aceptación de usuario, junto con pruebas funcionales integradas. Durante el proceso de promoción se construye un contenedor, autocontenido e inmutable, que se despliega en el ambiente QA.

Si el proceso de validación satisface los criterios de aceptación, viene una nueva promoción, y se clona el contenedor, el que se despliega en el ambiente de Producción. Lo que ocurre en este punto es la publicación del contenedor. Si se trata de una primera versión simplemente se deja disponible el contenedor a través en una URL pública. Si se trata de una nueva versión, se debe dar de baja el contenedor actual, y se reemplaza por uno nuevo. Esto se ilustra en la Figura 19.

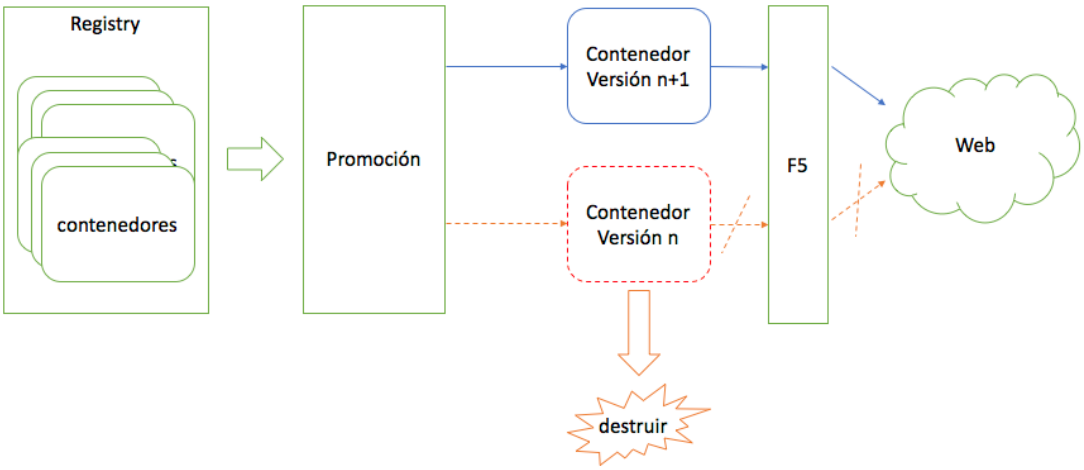

Figura 19. Promoción de una nueva versión reemplazando la anterior.

La principal diferencia es que los ambientes ya no son implementados por máquinas virtuales, las que son reemplazadas por contenedores. Tal como explicamos en la Sección 2.3, los contenedores son una tecnología que habilita de mejor manera la promesa de la entrega continua. En el antiguo proceso, las piezas de software se compilan e instalan en un ambiente, que corresponde a una máquina virtual. En el nuevo proceso, las piezas de software se compilan dentro de un contenedor de Docker que es almacenado en una imagen que se registra en el Registry (repositorio de artefactos), que es un almacén de las imágenes de los contenedores generadas por el servicio. Estas imágenes son “promovidas”, es decir, se acoplan a los parámetros de un ambiente, para ser ejecutadas como un contenedor de Docker.

La realización de la plataforma DevOps requiere de herramientas tecnológicas de diversos tipos, y, para las cuales, hay múltiples alternativas. De forma de tomar una decisión informada y adecuada a la cultura organizacional de Previred, se definieron un conjunto de criterios de selección, se realizó un relevamiento de las tecnologías candidatas, se evaluaron, y se tomó una decisión. A continuación, se describe este proceso.

4.3.2 Selección de Herramientas

Para seleccionar las herramientas tecnológicas que den soporte a la plataforma DevOps definida en la Sección 4.3.1, se definieron una serie de criterios en conjunto con el equipo de Tecnología de Apoyo al Giro. Estos criterios capturan las necesidades y restricciones deseables y esenciales para mantener una coherencia con las arquitecturas implantadas en la organización.

El objetivo de establecer estos criterios es ordenar la selección y establecer una justificación racional para la elección de las herramientas. Además, se busca que con estos criterios se reflejen los aspectos culturales de Previred, los que definen sus prácticas y entornos de ejecución. A continuación, se enumeran los criterios definidos:

- **Open Source o Propietario:** Se privilegia que la herramienta sea Open Source, salvo que sea complementaria con herramientas que ya estén en uso en Previred.
- **Costo:** El valor del licenciamiento o del soporte debe estar dentro de los presupuestos aprobados.
- **Integración con herramientas ya en uso:** Ante dos herramientas similares, se debe privilegiar aquella que tenga una integración más sencilla con herramientas que ya se utilizan en Previred.
- **Soporte técnico:** El nivel de soporte técnico, de existir, debe brindar acceso local (con un proveedor dentro del país). Si es un proyecto Open Source, se debe privilegiar aquellos que tienen empresas que brindan el soporte, preferentemente con presencia local.
- **Compatibilidad con la arquitectura actual:** Previred soporta sus aplicaciones sobre plataformas basadas en la Java Virtual Machine (JVM) y soportando los estándares Java Enterprise. Se deben privilegiar herramientas que también operen sobre este ambiente.
- **Seguridad:** La herramienta debe proveer mecanismos adecuados que resguarden los tres pilares de la seguridad de la información: confidencialidad, integridad y disponibilidad.

Valoración y Ponderación de estos criterios.

Para cada criterio se define una valoración, que va de bajo a alto, en el nivel de cumplimiento. A un cumplimiento bajo se le da 1 punto y al nivel más alto se le dan 3 puntos. De este modo, cada herramienta puede tener un mínimo de 6 puntos y un máximo de 18 puntos (uno o tres por criterio, respectivamente). Todos los criterios tienen la misma ponderación.

A continuación, se describen los valores para cada criterio:

- ***Open Source:***
  - **Alto**: Si el proyecto es Open Source (3 puntos).
  - **Medio**: Propietario, pero se puede integrar con productos ya adoptados por Previred (2 puntos).
  - **Bajo**: Propietario, y no tiene integración conocida con productos ya adoptados (1 punto).

- ***Costo:***
  - **Alto**: El costo es bajo, inferior a los US$ 20.000 anuales (3 puntos).
  - **Medio**: Entre US$ 20.000 y US$50.000 anuales (2 puntos).
  - **Bajo:** Sobre los US$50.000 anuales (1 punto).
- ***Integración con herramientas actuales***
  - **Alto**: Es parte de una suite ya adoptada, o tiene una integración sencilla (3 puntos).
  - **Media**: Es posible, pero requiere esfuerzo mayor a dos días (2 puntos).
  - **Bajo**: No es posible integrarla o requiere un esfuerzo de varios días (1 punto).
- ***Soporte técnico***
  - **Alto**: Existe y hay proveedores locales (3 puntos).
  - **Medio**: Hay, pero es internacional (2 puntos).
  - **Bajo**: No hay o no se logra identificar (1 punto).
- ***Compatibilidad con la arquitectura actual***
  - **Alto**: Es una solución basada en Java o se integra bien con RedHat Linux (3 puntos).
  - **Medio**: Es una solución basada en una arquitectura Open Source (por ejemplo, PHP, Python), pero no basada en la máquina virtual de Java (JVM) (2 puntos).
  - **Bajo**: Es una solución propietaria, o se desconoce la arquitectura de base, u opera en sólo en ambientes .Net y/o Windows (1 punto).
- ***Seguridad***
  - **Alto**: Tiene control de acceso y se puede integrar con directorios de usuarios provistos por Previred. Además, tiene una baja cantidad de vulnerabilidades conocidas. Por último, es posible monitorear o registrar en bitácoras el uso de la herramienta (3 puntos).
  - **Media**: Tiene control de acceso, perfilamiento, y una cantidad moderada de vulnerabilidades conocidas. No hay bitácora de uso (2 puntos).
  - **Baja**: Se han conocido vulnerabilidades graves, o no permite controlar el acceso ni monitorear su uso (1 punto).

Relevamiento de Herramientas Candidatas

Para seleccionar las herramientas candidatas, realizamos un relevamiento en la literatura disponible y enumerada en la Bibliografía, y en particular, en lo descrito en el libro “The DevOps 2.0 Toolkit” de V. Farcic (Farcic, 2016). Complementariamente, en conjunto con el equipo de Previred, buscamos recursos en Internet y en la literatura. Además, recabamos información de nuestros proveedores, IBM y RedHat, principalmente.

Del estudio realizado, se identificaron distintos tipos de herramientas necesarias para cada fase del proceso de integración y entrega continua, y que permiten realizar la plataforma DevOps propuesta. En la Figura 20 se presentan estos tipos de herramientas mediante un diagrama.

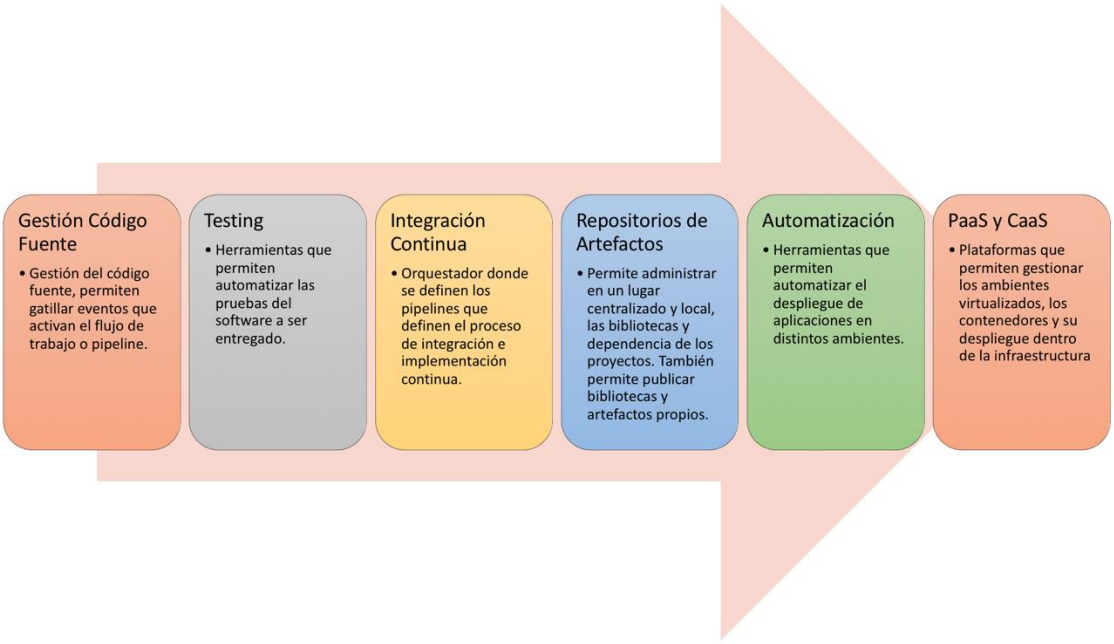

Figura 20. Tipos de herramientas necesarias para implementar la arquitectura propuesta.

4.3.3 Evaluación de Herramientas

Para evaluar las herramientas, se siguió el siguiente protocolo:

1. Establecimiento de los criterios relevantes que deben satisfacer las tecnologías que conformen la arquitectura de integración y entrega continua de Previred. Para ello, se realizó una reunión de brainstorming con el equipo de arquitectura de software. Los criterios son los descritos anteriormente en la Sección 4.3.2.
2. Elaboración de una primera propuesta de arquitectura, la que se presentó a los equipos de desarrollo, calidad, arquitectura de sistemas e infraestructura. Esta arquitectura general se refinó con las actividades siguientes y corresponde a la expuesta en la Sección 4.3.1.
3. Investigación propia en la literatura e Internet, de herramientas que permiten habilitar la arquitectura propuesta.
4. Reuniones con proveedores de herramientas, conocidos y que pueden brindar soporte y capacitación, para conocer sus propuestas.
5. Solicitud de cotizaciones formales de costos a los proveedores de herramientas.
6. Una vez levantada la información respecto a herramientas, se elaboró una lista, la que se evaluó usando como instrumento una matriz de evaluación basada en la puntuación de los criterios de selección.
7. Finalmente, se presentó la propuesta a la subgerencia encargada del área de sistemas, y se solicitó a la Gerencia de Operaciones y Tecnología y a la Gerencia General, la aprobación de presupuesto para proceder con el proyecto.

Ejecución

Para ejecutar esta parte se realizaron las reuniones internas mencionadas en la sección anterior, y se agendaron entrevistas con los dos principales proveedores de Previred: IBM

en su rol de Data Center, y RedHat como principal proveedor de plataformas de software base.

Posteriormente se realizaron las reuniones con los proveedores, donde revisaron distintas alternativas. Finalmente, por razones de compatibilidad con nuestra cultura y arquitectura, se optó por continuar el proceso con RedHat. En paralelo, el autor junto con miembros de su equipo, investigaron la literatura y recursos en Internet de diversas herramientas, y con esta información se alimentaron las matrices, las que fueron contrastadas y recopiladas en una reunión del equipo. Como actividad complementaria, se realizaron pruebas de algunas herramientas y se planificó con RedHat la ejecución de la prueba de concepto (PoC).

Selección de Herramientas

Luego de realizar las actividades descritas anteriormente, se seleccionó el siguiente conjunto de herramientas:

- *Gestión Código Fuente.* No fue necesario seleccionar una herramienta pues se decidió continuar utilizando la herramienta actual ya usada por la empresa, específicamente Atlassian Bitbucket, un gestor de versiones basado en git.
- *Testing.* Se seguirán usando las herramientas ya integradas en la empresa: SonarQube, Selenium, Junit y Jmeter.
- *Integración Continua.* Se evaluaron dos herramientas, Jenkins y Bamboo.
- *Repositorios de Artefactos.* Se evaluaron SonaType Nexus y Artifactory.
- *Automatización.* Se evaluaron las herramientas Chef, Ansible y Puppet.
- *PaaS y CaaS.* Se evaluaron Docker Swarm, Kubernetes y OpenShift.

Matriz de Evaluación

Una vez seleccionada las herramientas, se trabajó en una matriz de evaluación que se muestra en la Tabla 6, y que describe el resultado de la aplicación del protocolo descrito anteriormente. La matriz se completó en base a los resultados de las reuniones de evaluación realizadas por el equipo de tecnología de Previred.

Herramienta	Descripción	Criterios	Puntaje	Total
Jenkins	Herramienta de Integración Continua	Open Source	3	16
		Costo	3	
		Integración	3	
		Soporte	2	
		Compatibilidad	3	
		Seguridad	2	
Bamboo		Open Source	1	16

	Herramienta de Integración Continua	Costo	3	
		Integración	3	
		Soporte	3	
		Compatibilidad	3	
		Seguridad	3	
Chef	Herramienta de scripting para automatizar la configuración y el despliegue de ambientes	Open Source	2	14
		Costo	2	
		Integración	2	
		Soporte	3	
		Compatibilidad	2	
		Seguridad	3	
SonarQube	Herramienta que realiza análisis estático de código y revisa el cumplimiento de diversas métricas de calidad de software	Open Source	3	16
		Costo	3	
		Integración	3	
		Soporte	2	
		Compatibilidad	3	
		Seguridad	2	
JMeter	Herramienta para automatizar tests funcionales y de carga	Open Source	3	16
		Costo	3	
		Integración	3	
		Soporte	2	
		Compatibilidad	3	
		Seguridad	2	
Selenium	Herramienta para automatizar tests funcionales	Open Source	3	16
		Costo	3	
		Integración	3	
		Soporte	2	
		Compatibilidad	3	
		Seguridad	2	

Ansible	Herramienta de scripting para automatizar la configuración y el despliegue de ambientes	Open Source	3	17
		Costo	3	
		Integración	3	
		Soporte	3	
		Compatibilidad	2	
		Seguridad	3	
Puppet	Herramienta de scripting para automatizar la configuración y el despliegue de ambientes	Open Source	2	14
		Costo	2	
		Integración	2	
		Soporte	2	
		Compatibilidad	3	
		Seguridad	3	
Docker Swarm	Herramienta para administrar clusters de contenedores	Open Source	3	15
		Costo	3	
		Integración	2	
		Soporte	2	
		Compatibilidad	2	
		Seguridad	3	
OpenShift	Herramienta para administrar clusters de contenedores	Open Source	3	17
		Costo	2	
		Integración	3	
		Soporte	3	
		Compatibilidad	3	
		Seguridad	3	
SonaType Nexus	Repositorio de Artefactos	Open Source	3	16
		Costo	3	
		Integración	2	
		Soporte	2	
		Compatibilidad	3	

		Seguridad	3	
Artifactory	Repositorio de Artefactos	Open Source	3	15
		Costo	2	
		Integración	2	
		Soporte	2	
		Compatibilidad	3	
		Seguridad	3	

Tabla 6. Matriz de Evaluación de las herramientas.

En base a la matriz de evaluación de herramientas de la Tabla 6, la suite de herramientas seleccionadas para implementar la plataforma DevOps queda tal como se describe en la Figura 21, y que detallamos a continuación.

Originalmente se consideró continuar usando Bamboo como herramienta de integración continua, pero tras evaluaciones técnicas más exhaustivas y dado que el nivel de integración con OpenShift de esta herramienta es menos maduro, se decidió cambiar Bamboo por Jenkins.

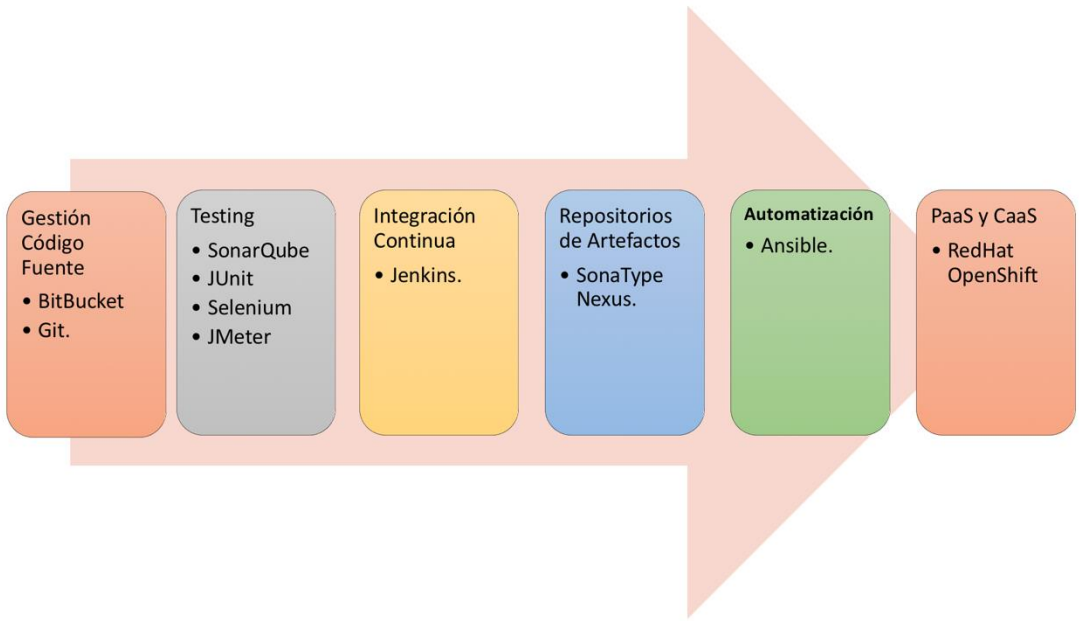

Figura 21. Herramientas seleccionadas para implantar la arquitectura DevOps en Previred.

Para la gestión de código fuente se eligió Atlassian Bitbucket, que corresponde a un gestor de repositorios de código fuente basados en git. Este producto ya se usa en Previred. Su rol en la arquitectura es el de ser gatillador de eventos de implantación en los distintos ambientes.

Para testing escogimos a SonarQube, JUnit, Selenium y herramientas ya usadas por los equipos de calidad y desarrollo para realizar pruebas de integración automatizadas de las aplicaciones.

Jenkins, es el orquestador principal de los procesos de compilación (build), instalación, configuración, despliegue y testing.

SonaType Nexus, es un repositorio de artefactos en donde se guardan todos los objetos binarios que son usados para compilar, o que deben ser desplegados en los distintos ambientes.

Ansible, es una herramienta que permite automatizar la configuración de ambientes contenedores de aplicaciones, y que además permite automatizar tareas de configuración necesarias para el despliegue (instalación de los artefactos en los distintos ambientes).

RedHat OpenShift, es el orquestador y gestor de clusters Docker, que corresponden a los contenedores donde residen las aplicaciones.

4.3.4 Arquitectura Final

La Figura 22 muestra la arquitectura final implantada.

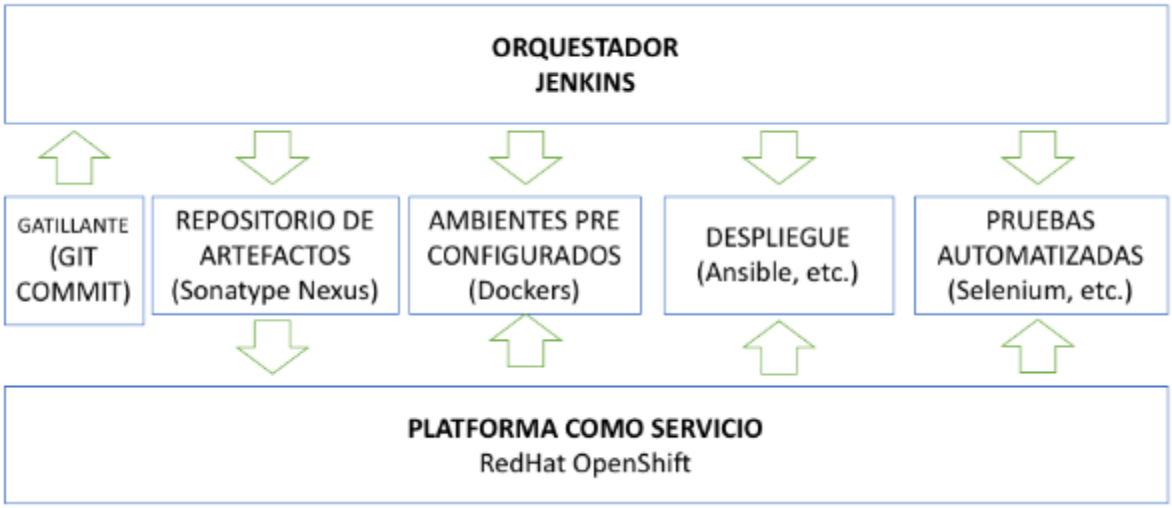

Figura 22. Arquitectura Final.

Una pieza esencial de la arquitectura es OpenShift Container Platform (OCP) [21]. Esta es una plataforma de gestión de contenedores desarrollada por RedHat. OCP es un producto de Plataforma como Servicio (PaaS) on premise, que soporta contenedores Docker, y que orquesta y gestiona estos contenedores usando Kubernetes. La Figura 23 describe la arquitectura de OCP, tal como la documenta su proveedor.

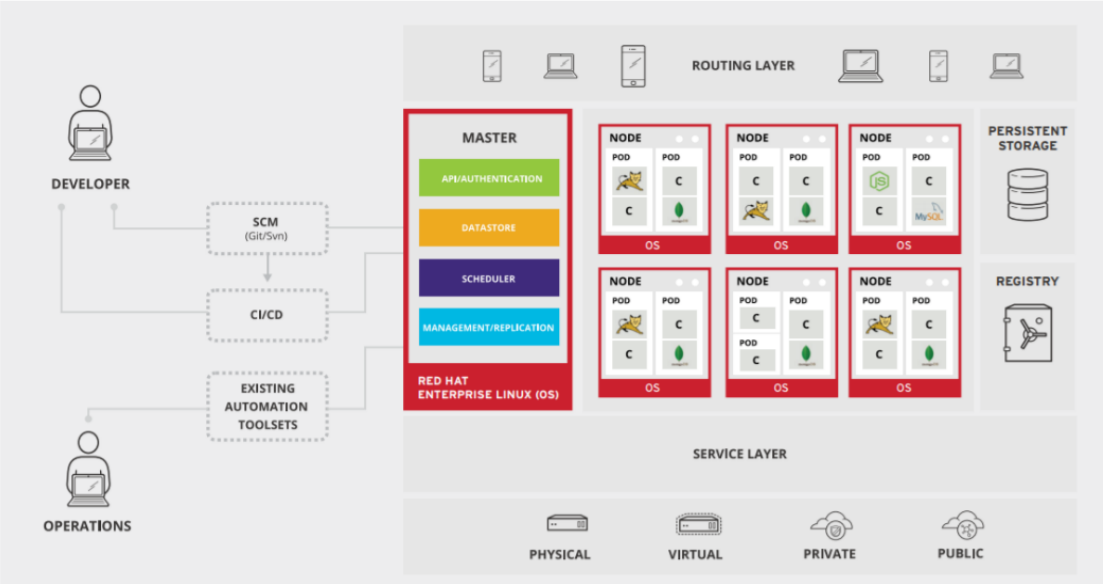

Figura 23. La arquitectura OpenShift Container Platform [13].

OCP se integra con herramientas de integración e implantación continua (CI-CD, por las siglas en inglés de Continuous Integration y Continuous Deployment), de gestión de código fuente y configuración (SCM, por las siglas en inglés de Software Configuration Management), y de automatización ya disponibles en la empresa. Los elementos esenciales de la arquitectura OCP son:

- Master: que contiene los módulos esenciales de control:
    - API-Authentication: que provee una API REST que permite integrar OCP con herramientas externas. Además, provee los servicios de autenticación tanto de usuarios, como de aplicaciones que usan o se integran con OCP.
    - DataStore: donde se almacenan las configuraciones.
    - Scheduler: que permite organizar tareas y programar sus ejecuciones de manera organizada y sincronizada.
    - Management/Replication: que permite gestionar la plataforma y replicar los servicios, a través de clusters.
    - Node: que corresponde a una máquina virtual que puede contener uno o más pods
    - Pod: es un conjunto de uno o más contenedores, que comparten almacenamiento y red. Cada pod tiene una especificación de cómo ejecutar los contenedores. Un pod modela un “host lógico” específico de la aplicación, contiene varios contenedores de aplicación que tienen un nivel mayor de acoplamiento. Los pods facilitan la gestión y el escalamiento horizontal de las aplicaciones ("Kubernetes Concepts, Pods", s.f.).
- Service Layer: es la interfaz con la infraestructura física.
- Routing Layer: es la interfaz de comunicaciones y enrutamiento de redes. En nuestro caso se integrará con el balanceador de tráfico F5 (Routing in RedHat OpenShift Container Platform with F5 Just Got Easier, s.f.).

Orquestador de CI/CD Jenkins

La otra pieza importante dentro de la arquitectura propuesta es el orquestador de integración continua. En nuestro caso hemos decidido usar Jenkins. Este es un cambio importante en nuestro ambiente, puesto que ya teníamos experiencia con Bamboo, que es una herramienta de la Suite de Atlassian, usada para la gestión del ciclo de vida del software (ALM) ("Guía de productos Atlassian", s.f.) (Wikipedia Artículo: Application Lifecycle Management (ALM), s.f.).

La decisión anterior se debe a que Jenkins tiene un grado mayor de integración con OpenShift, que es la plataforma como servicio elegida.

Jenkins es un servidor de integración continua que automatiza la compilación de las piezas de software, el testing automatizado de éstas mediante pruebas unitarias e integradas, y la instalación de estas piezas en distintos ambientes. En nuestro caso, Jenkins orquestará los procesos de OpenShift para el despliegue de las piezas en los distintos contenedores.

Tanto Jenkins como OpenShift proveen APIs de integración, usando el protocolo REST. Además, hemos integrado Jenkins con SonarQube, una herramienta de análisis estático de código. El código fuente se almacena en repositorios git usando el servidor Atlassian Bitbucket para su administración.

El *pipeline* de entrega definido está basado en lo descrito por S. Sadeghianfar en (Sadeghianfar, s.f.), pero adaptado a nuestra realidad. La generalización está en los pasos finales, puesto que podemos promover la imagen Docker a distintos ambientes en nuestra arquitectura. La Figura 24 muestra el pipeline definido.

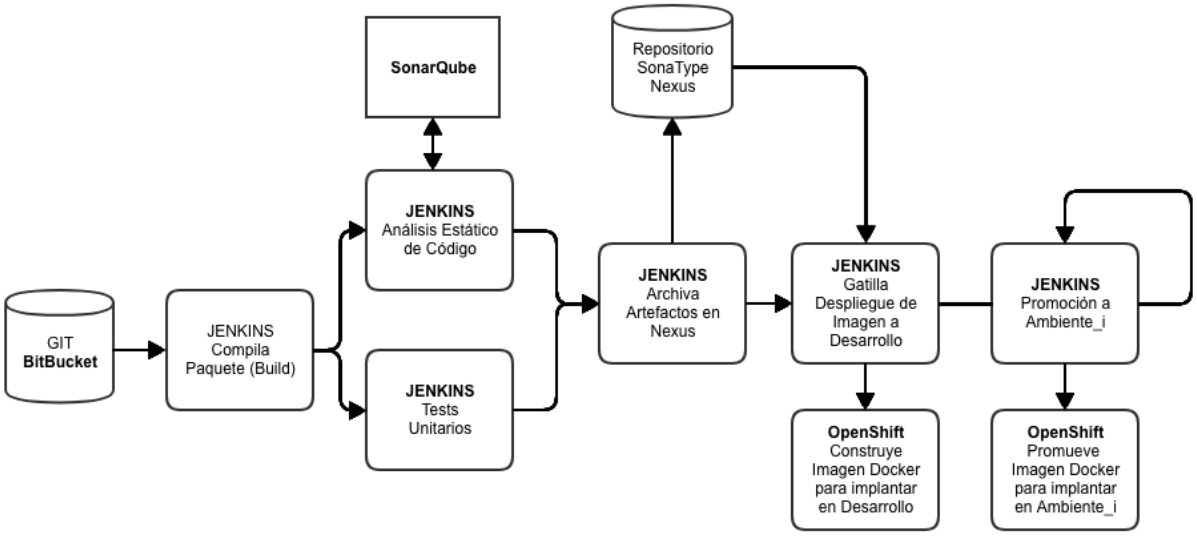

Figura 24. Pipeline integrado entre Jenkins y OpenShift.

5. Validación de la solución

La validación de la solución de entrega continua para Previred se realizó utilizando tres mecanismos complementarios. Primero, se realizó una validación de factibilidad técnica, la que permitió demostrar que la arquitectura de la plataforma definida, y las tecnologías seleccionadas, funcionaban integradamente para lograr su objetivo. Segundo, se realizó un piloto aplicado al desarrollo de un nuevo servicio en Previred. Por último, se realizó una validación de utilidad e impacto de la solución planteada, evaluando con los principales interesados cuán efectivos son el nuevo proceso y la plataforma implantada en la resolución o reducción de los dolores organizacionales detectados.

En la Sección 5.1 detallamos el estudio de factibilidad técnica realizado. La Sección 5.2 describe el piloto y su ejecución. Finalmente, en la Sección 5.3 se describe la validación cualitativa, con su ejecución y los resultados obtenidos de la evaluación realizada con los interesados e involucrados en la plataforma.

5.1 Validación de factibilidad técnica

En conjunto con el equipo de Tecnología de Previred y un equipo técnico de RedHat, se realizó primero una prueba de concepto (PoC) de la plataforma tecnológica para validar la integración de distintas piezas de la arquitectura y el logro de los objetivos de automatización. Esta PoC terminó con un workshop realizado en las oficinas de Previred, donde los equipos que no participaron de la PoC pudieron probar la plataforma y discutir su arquitectura.

La PoC consistió en desplegar dos servicios de Previred dentro de una instalación reducida de OpenShift. Esta versión de OpenShift se instaló en nuestro Data Center en IBM. La prueba consistió en la integración de Atlassian Bitbucket (gestor de código fuente basado en git), el repositorio de artefactos SonaType Nexus, y el despliegue automático de las aplicaciones en dos ambientes de RedHat OpenShift.

Además, en paralelo a la PoC, y en un ambiente distinto, se implantaron pruebas unitarias y revisión de código sobre SonarQube.

Tal como se explicó en la Sección 4.3.3, Atlassian Bamboo y Jenkins eran las dos herramientas candidatas para lograr la integración continua. Por un lado, el equipo tenía experiencia en Atlassian Bamboo y, aunque es una herramienta propietaria y con costo de licenciamiento, Previred ya contaba con ésta y otras herramientas del proveedor. Por otro lado, Jenkins en una herramienta open source con una comunidad internacional grande y activa. La evaluación de ambas herramientas dio un puntaje de 16 (como se indica en la Tabla 6), por lo que los criterios de selección definidos no fueron suficientes para decidir entre una u otra herramienta. Durante la planificación de la PoC, se decidió probar con Atlassian Bambo, y en caso de no tener éxito, utilizar Jenkins. Durante la ejecución de la PoC, tuvimos que descartar Atlassian Bamboo debido a que su integración con OpenShift requería el uso de plugins para los cuales no contábamos con las licencias correspondientes y a que el equipo de RedHat declaró tener mayor experiencia usando Jenkins. Por este motivo, la PoC se realizó con Jenkins y, finalmente, se decidió que sería ésta la herramienta de integración continua para la plataforma de Previred.

5.1.1 Ejecución de la prueba de concepto

Durante la prueba de concepto se validaron las etapas destacadas de color naranja en la Figura 25. La construcción se validó a través del pipeline de construcción implantado mediante la herramienta Jenkins. En esta etapa también se validó el uso de Git Flow. La integración continua permitió probar la construcción de scripts de building automatizado, junto con el despliegue de los mismos en la plataforma OpenShift. Para las actividades de pruebas automatizadas y de pruebas de integración se validó el uso de JUnit, JMeter y el analizador estático de código SonarQube. Para la entrega continua se probó que Jenkins y Openshift trabajaran de manera coordinada, y se realizó una promoción de los contenedores desde Desarrollo a QA.

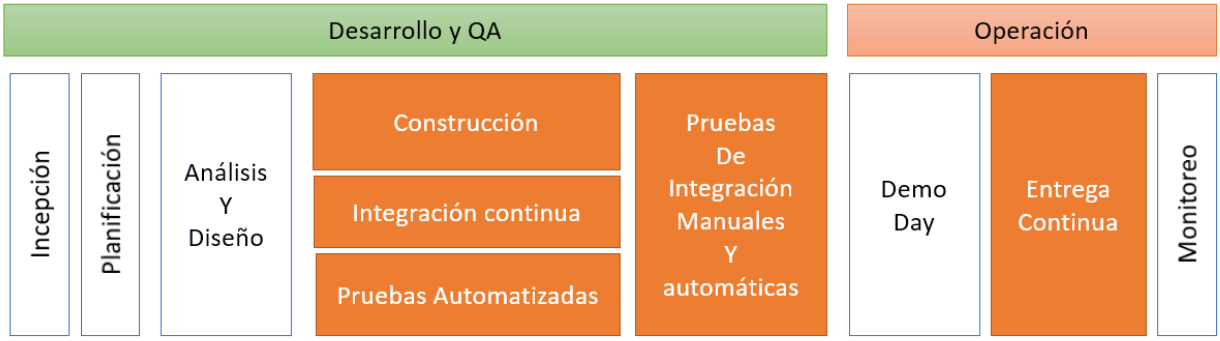

Figura 25. Etapas que fueron validadas en la prueba de concepto.

Todas estas actividades permitieron validar el pipeline definido que se mostró en la Figura 24. En concreto, durante esta prueba de concepto se colocaron en un repositorio git el código fuente de dos sistemas desarrollados por Previred. Se configuró Jenkins para que ejecutara un build de las aplicaciones a partir del código fuente en git, y el resultado se depositó en un repositorio Nexus. Luego se gatilló la construcción de una imagen Docker que se implantó en ambiente de Desarrollo. Luego se realizó una promoción automática de esta imagen al ambiente de QA, en donde se realizaron pruebas de integración manuales y automáticas. Todos estos ambientes ahora operan sobre contenedores Docker.

5.1.2 Resultados de la Prueba de Concepto

Los principales resultados de esta prueba de concepto fueron las siguientes:

- Es factible usar OpenShift para desplegar aplicaciones construidas por los equipos de desarrollo de Previred.
- Se pudo estimar la capacidad requerida por la plataforma para estimar las inversiones necesarias para adquirirla.
- Se probó la integración con el balanceador F5 y se resolvieron dudas sobre cómo se integran ambas herramientas.
- Se realizó un workshop que permitió a los arquitectos conocer en profundidad la plataforma y aprender cómo programar sus propios scripts de automatización.
- Se decidió abandonar Bamboo y reemplazarlo por Jenkins como orquestador de integración.

5.2 Piloto

Con el resultado exitoso de la prueba de concepto se obtuvo la autorización de la alta gerencia para avanzar con un piloto, con el objetivo de aplicar el proceso y la arquitectura en un proyecto de desarrollo de un nuevo servicio.

El piloto consideró la ejecución de tres sprints, de cuatro semanas de duración cada uno. Al final de cada sprint se realizó la implantación del sistema en la plataforma OpenShift provista. En esta sección se explica la ejecución del piloto.

5.2.1 Despliegue

En conjunto con el proveedor RedHat se procedió a implantar una plataforma en los ambientes productivos de Previred. Cabe señalar que Previred opera en dos Data Center, uno principal y otro de contingencia. Entre ambos sitios existe un mecanismo de respaldo en línea. De este modo, en caso de producirse una disrupción en el sitio principal, la empresa puede activar el sitio de contingencia y operar desde el segundo Data Center. Es por esta razón que la plataforma se encuentra instalada y activa en los dos sitios, tal como se expone la Figura 26.

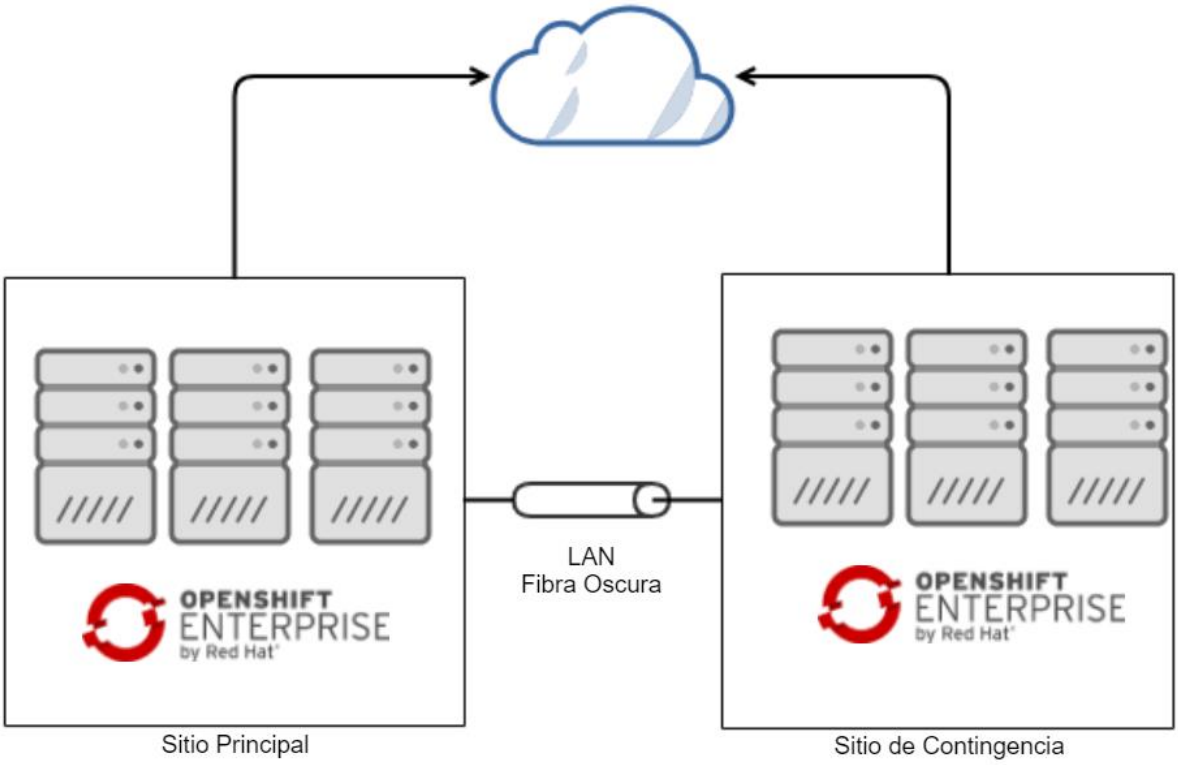

Figura 26. Instalación de OpenShift en los Data Centers de Previred.

En cada sitio se implantó la plataforma de contenedores OpenShift. Los contenedores desplegados en esta plataforma se etiquetan de acuerdo al ambiente al que pertenecen (Desarrollo, QA y Producción). Además de la plataforma OpenShift en el Data Center se desplegaron las siguientes componentes adicionales:

- Repositorio GIT Bitbucket
- Motor de Integración Jenkins
- Ansible Tower
- SonarQube
- SonaType Nexus

La Figura 27 muestra gráficamente cuáles componentes fueron instaladas en cada sitio.

Figura 27. Despliegue de componentes para el piloto.

5.2.2 Ejecución del Piloto

Durante el piloto se validaron todas las etapas del nuevo proceso de desarrollo, que se destacan en naranja en la Figura 28. Se excluyó la etapa Incepción puesto que el proyecto ya estaba definido de antemano, y el monitoreo, pues está fuera del alcance del trabajo y Previred planea abordarlo en el futuro. El piloto se alineó a la ejecución de un proyecto interno de Previred, y se ejecutó durante tres sprints, por lo que cada iteración contó con su fase de planificación, junto con el análisis y diseño de cada incremento. Al final de cada sprint se realizaron las Demo Day tal como lo indica el proceso. Las etapas de Construcción, Integración continua y pruebas fueron validadas de la misma manera que durante la prueba de concepto. Sin embargo, se realizó un cambio en la fase de entrega, que se explica en la sección 5.2.4.

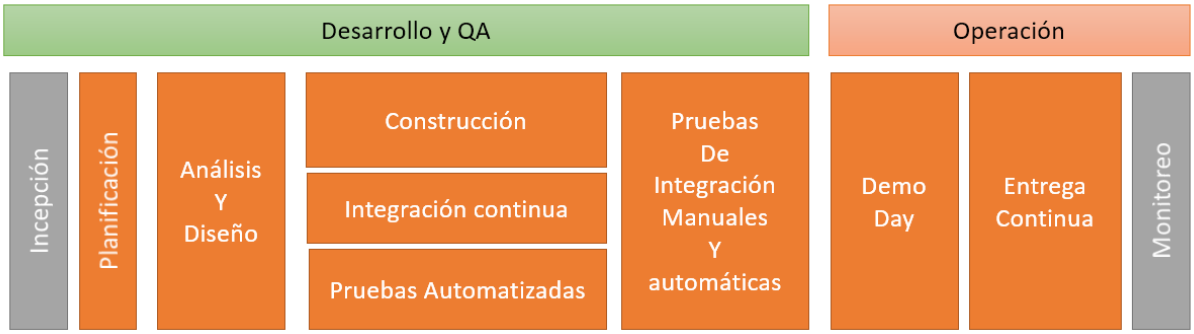

Figura 28. Etapas validadas durante el piloto.

La Figura 29 muestra el pipeline implementado finalmente durante el piloto. Si bien es similar al descrito anteriormente en la Sección 4.3.4, Figura 24, se introdujo un paso manual antes de realizar la promoción a producción. De este modo, la promoción de las imágenes Docker entre ambientes de Desarrollo y QA se realiza de forma semiautomática, la promoción al ambiente de producción final requiere la aprobación formal del usuario responsable (o dueño del negocio). El último paso de promoción al ambiente productivo es semiautomático, pero es ejecutado por un ingeniero de sistemas después de recibir el visto bueno del usuario.

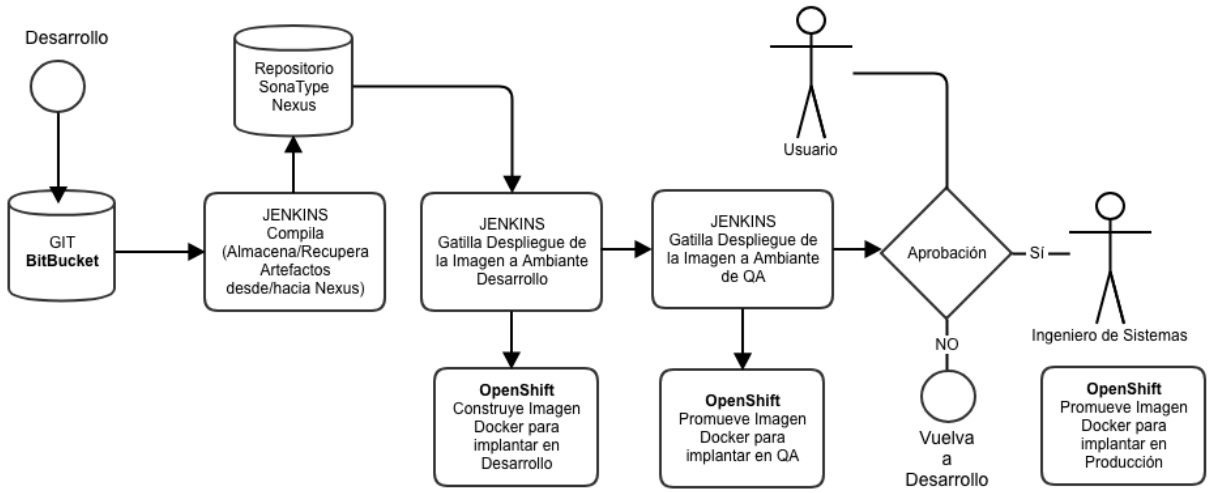

Figura 29. Pipeline implementado durante el piloto.

La razón de la introducción de este paso tiene que ver con cumplir con las formalidades exigidas por nuestra área de auditoría y control de riesgo, y se formalizó dentro del proceso de desarrollo. La aprobación es evidenciada mediante un correo electrónico, el que es almacenado en un gestor documental donde se guarda toda la documentación del proyecto.

5.2.3 Resultados del Piloto

Los principales resultados de este piloto fueron las siguientes:

- Gracias al pipeline de integración continua es posible realizar implantaciones más rápidas y frecuentes de las aplicaciones. Se midió que el proceso de pasar a producción toma menos de cinco minutos. Esto fue aprovechado por el equipo de desarrollo para iterar rápidamente en sus pruebas unitarias.
- No hubo problemas de homologación de ambientes, puesto que la imagen que era promovida era exactamente la misma.
- Se tuvo que modificar la forma en que las aplicaciones administran su configuración, como por ejemplo, obtener variables de ambiente, o valores de parámetros para la operación de las mismas. Para esto los desarrolladores adoptaron el framework Spring Cloud Config [22], que permiten obtener las configuraciones directamente del ambiente al cual se promueve el contenedor.
- Mediante el uso de Spring Cloud Config podemos además garantizar la seguridad de la configuración de ambientes en OpenShift. De este modo, elementos sensibles (como por ejemplo, las claves a las bases de datos productivas) quedaron debidamente custodiados por los miembros del equipo autorizados a modificarlos (ingenieros de sistemas), sin que los desarrolladores pudieran acceder a esta información. Con esto garantizamos los estándares de seguridad exigidos por la organización.
- Es tan rápido el despliegue de aplicaciones, que al principio causó aprensiones, puesto que si bien es muy fácil y rápido desplegar mejoras, con la misma rapidez se pueden implantar errores en ambientes productivos. Es por esto que se agregó el paso de aprobación antes de promover una nueva imagen a producción, de modo de dar tranquilidad a nuestros usuarios.
- Con respecto al proceso de vuelta atrás (rollback) ante una instalación fallida, estos resultaron ser muy simples. En el segundo sprint fue necesario realizar una vuelta atrás, y esta fue tan rápida como el despliegue porque sólo requirió restaurar una imagen anterior. Sin embargo, confirmamos que, si la vuelta atrás implica deshacer un cambio en la base de datos, el proceso de rollback no es tan sencillo.

5.2.4 Análisis de la ejecución del piloto

El primer impacto observado por todos los participantes del proceso fue la reducción de tiempo asociado al paso a producción. Hoy en día, con el proceso actual, la instalación de una nueva aplicación requiere la coordinación con el Data Center con 48 horas de anticipación. Esto obliga a entregar las piezas de software a instalar muchas horas antes. Por otro lado, se deben preparar documentos, lo que implica la inversión de al menos un par de horas por parte de personal del equipo. Además, se requiere programar “ventanas de interrupción del servicio”, de una a dos horas. Esto además fuerza a que las implantaciones se hagan durante horarios no hábiles, quedando muchas instalaciones programadas para la noche o madrugada, lo que obliga a coordinar estas tareas entre varias personas que deben quedarse cumpliendo turnos para verificar la correcta instalación.

Durante el piloto, no hubo interrupción del servicio apreciable por el usuario. Una vez finalizado el desarrollo, la compilación (build) de las piezas de software se ejecuta de manera automática, luego se ejecutan las pruebas unitarias y los desarrolladores reciben un feedback inmediato. Cuando el equipo de desarrollo estima que la labor está concluida el proceso de promover el software al ambiente de QA requiere, literalmente, de presionar un botón. En minutos la nueva versión está disponible para las pruebas de integración, y la fase de aceptación de usuarios (a través de pruebas de humo, y de manera formal en el DemoDay). Una vez aceptado el producto, la instalación es ejecutada de inmediato, sin necesidad de una coordinación con tantas horas de anticipación. Todo el proceso de promoción de aplicaciones entre ambientes no toma más de cinco minutos. Asimismo, la vuelta atrás es igual de rápida (desde la ejecución del piloto se ha aplicado una vuelta atrás sólo una vez).

Dado lo anterior, los pasos a producción se pueden programar en horario hábil. Esto da más tranquilidad a todos y reduce tensiones debido a la necesidad de trabajar en horarios nocturnos. Otro alivio fue la eliminación de documentos de paso a producción, los que fueron reemplazados por un ticket en Jira que contiene las referencias a los cambios realizados que pasan a producción.

Previred ha ejecutado en el último año 352 cambios, o pasos a producción. Si estimamos que la preparación y ejecución de estos requiere unas 12 horas hombre (entre elaboración de documentos, reuniones de coordinación y ejecución del mismo), tenemos un total de 4.224 horas al año. Esto es equivalente al tiempo anual de dos ingenieros de sistemas en jornada completa. Automatizar la entrega continua reduce ese tiempo prácticamente a cero (si consideramos los cinco minutos observados en el piloto, se llega a una cifra menor a 40 horas al año, pero por supuesto, eso asumiendo que se está observando el proceso durante ese tiempo, una labor que es innecesaria puesto que el proceso es automático).

El nivel de cobertura de pruebas para el proyecto del piloto fue del 100%, es decir, se crearon pruebas unitarias y pruebas de integración para todos los módulos implementados. Por supuesto, durante la ejecución del piloto hubo fallos, detectados en ambiente de producción, aunque ninguno de nivel crítico. En algunos casos los fallos fueron corregidos por el equipo en menos de dos horas, gracias a las pruebas automatizadas y el proceso de entrega continua.

5.3 Validación de la solución

Con el fin de evaluar la solución se procedió a aplicar un cuestionario a los mismos participantes de la etapa descrita en la Sección 3.3.2 del protocolo de identificación de problemas.

5.3.1 Protocolo de validación de la solución

Objetivo

El protocolo de validación de la solución tiene por objetivo identificar el grado de cumplimiento y satisfacción que interpreta cada actor con el nuevo proceso y la arquitectura de la plataforma propuesta.

Medición

Para medir la satisfacción, se entrevista a cada actor y se les pide contestar una encuesta con diez preguntas en escala de Likert, con el fin de medir cómo se percibe que los cambios propuestos apuntan a mejorar los principales dolores levantados en la Sección 3.

Formato

Para ejecutar el protocolo se realizan entrevistas de quince minutos. Al inicio de la entrevista se les muestran y explican los diagramas del nuevo proceso y de la arquitectura propuesta, además se les describen los resultados obtenidos tanto en la prueba de concepto, como en el piloto. Después se les pide que contesten la encuesta mostrada en la Tabla 7.

Evaluación del impacto de la arquitectura de entrega continua de Previred

Cargo		Fecha:	

De acuerdo a su percepción, califique en qué situación quedará cada dolor una vez implantada la arquitectura de entrega continua.

	Dolor	Mucho peor	Peor	Igual	Mejor	Mucho mejor	No sé
1	El equipo del Data Center tiene problemas de prolijidad al aplicar cambios.						
2	Queremos disminuir los daños colaterales después de cada cambio o paso a producción.						
3	El tiempo para corregir un fallo en producción es excesivo.						
4	Los distintos ambientes (desarrollo, QA, producción) no están homologados.						
5	Los costos del proceso de desarrollo deben mantenerse dentro de un margen razonable y no ir en aumento						
6	El diagnóstico de la causa de las fallas en producción toma mucho tiempo, y el primer diagnóstico no siempre es el adecuado.						
7	Es esencial cumplir los plazos comprometidos por desarrollo.						
8	El esfuerzo de validación de los cambios es excesivo.						
9	Excesiva burocracia y papeleo para realizar un paso a producción.						
10	La duración de las etapas de paso a producción toman mucho tiempo						

Tabla 7. Encuesta de evaluación cualitativa de la solución.

5.3.2 Ejecución de las entrevistas

Las entrevistas se realizaron a un grupo de personas que representan los siguientes roles dentro de la organización:

- Gerencia: Gerente del Área Comercial, Subgerente de Sistemas
- Jefaturas: Jefa del Área de Calidad, Jefa del Área de Desarrollo
- Dos Desarrolladores, un Arquitecto de Software
- Un Tester, un Administrador de la Configuración, y un Ingeniero de Sistemas,
- Un Usuario y un Product Owner

5.3.3 Resultados

Se obtuvieron ciento veinte respuestas al cuestionario expuesto en la Tabla 7. En la Tabla 8 se muestra el detalle de las respuestas obtenidas. Los valores que se muestran en cada casilla son la cantidad de respuestas que recibió cada respuesta. Por ejemplo, en el Dolor 3 se recibieron 6 respuestas indicando que la situación era Mejor.

	Dolor	Mucho peor	Peor	Igual	Mejor	Mucho mejor	No sé
1	El equipo del Data Center tiene problemas de prolijidad al aplicar cambios.				5	4	3
2	Queremos disminuir los daños colaterales después de cada cambio o paso a producción.			3	5	2	2
3	El tiempo para corregir un fallo en producción es excesivo.			2	6	4	
4	Los distintos ambientes (desarrollo, QA, producción) no están homologados.			2	3	4	3
5	Los costos del proceso de desarrollo deben mantenerse dentro de un margen razonable y no ir en aumento		1	2	4	2	3
6	El diagnóstico de la causa de las fallas en producción toma mucho tiempo, y el primer diagnóstico no siempre es el adecuado.			4	3	2	3
7	Es esencial cumplir los plazos comprometidos por desarrollo.			4	3	2	3
8	El esfuerzo de validación de los cambios es excesivo.			4	4	2	2
9	Excesiva burocracia y papeleo para realizar un paso a producción.				4	6	2
10	La duración de las etapas de paso a producción toman mucho tiempo			1	5	4	2

Tabla 8. Respuestas obtenidas.

El resultado se tabuló de modo que las respuestas arrojan un valor entre 1 y 5 (1 es pMucho Peor, 2 es Peor y así hasta Mucho Mejor que tiene el valor 5). Las respuesta No

Sé tienen un valor 0. El resultado de esta evaluación arroja una nota promedio de 4.1, en una escala de 1 a 5.

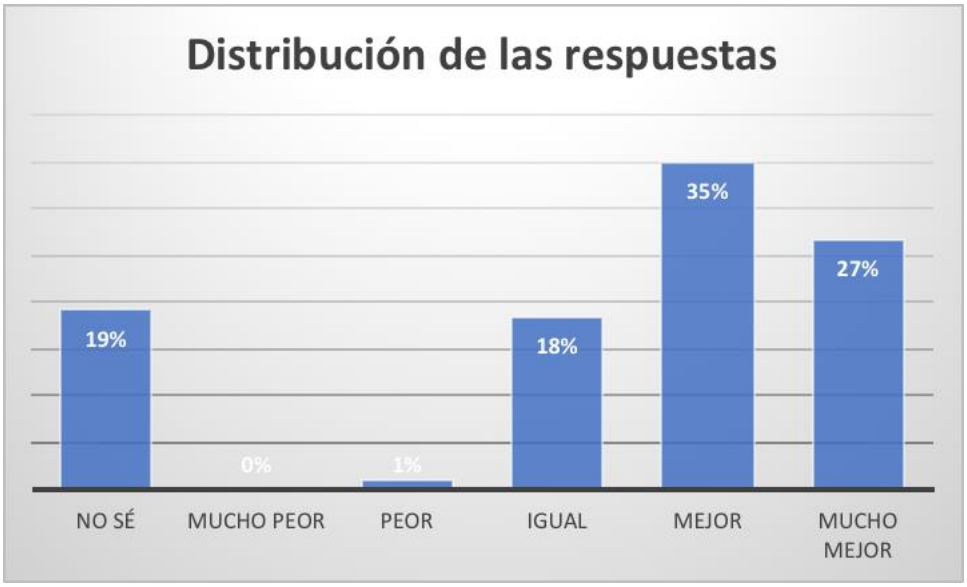

Figura 30. Distribución de las respuestas en la evaluación de la solución.

La Figura 30 muestra la distribución de las respuestas en esta evaluación. El 35% de los entrevistados considera que la solución mejorará la situación actual, basados en los diez principales dolores, mientras que el 27% considera que es mucho mejor que la situación actual. La Tabla 9 muestra los porcentajes obtenidos por las respuestas a cada pregunta de la encuesta.

#	Dolor	Mucho Peor	Peor	Igual	Mejor	Mucho Mejor	No sé
1	El equipo de Data Center tiene problemas de prolijidad al aplicar cambios.	0%	0%	0%	42%	33%	25%
2	Queremos disminuir daños colaterales después de cada cambio o paso a producción.	0%	0%	25%	42%	17%	17%
3	El tiempo para corregir un fallo en producción es excesivo.	0%	0%	17%	50%	33%	0%
4	Los distintos ambientes (desarrollo, QA, Producción) no están homologados.	0%	0%	17%	25%	33%	25%
5	Los costos del proceso de desarrollo deben mantenerse dentro de un margen razonable y no ir en aumento.	0%	8%	17%	33%	17%	25%
6	El diagnóstico de las fallas en producción toma mucho tiempo y el primer diagnóstico no siempre es el adecuado.	0%	0%	33%	25%	17%	25%
7	Es esencial cumplir los plazos comprometidos por desarrollo.	0%	0%	33%	25%	17%	25%
8	El esfuerzo de validación de los cambios es excesivo.	0%	0%	33%	33%	17%	17%
9	Excesiva burocracia y papeleo para realizar un paso a producción.	0%	0%	0%	33%	50%	17%
10	La duración de las etapas de paso a producción toman mucho tiempo.	0%	0%	8%	42%	33%	17%

Tabla 9. Distribución de las respuestas.

5.3.4 Discusión

Los dos dolores que se perciben como mejor resueltos son el número 9 (“Excesiva burocracia y papeleo para realizar un paso a producción”) y el número 3 (“El tiempo para corregir un fallo en producción es excesivo”), ambos con un 83% de respuestas entre 4 y 5. Esto se explica porque son los dolores cuyo impacto es más fácil de percibir tras la prueba de concepto. La implantación es automática y toma muy poco tiempo, lo que contrasta fuertemente con la realidad actual, que requiere escribir un documento y coordinar con días de anticipación la instalación. En la prueba de concepto se pudo apreciar que instalar es tan fácil como apretar un botón y puede ser ejecutada a demanda. Del mismo modo, los cambios en el proceso también impactan en el tiempo ocupado para corregir un problema.

Luego, en tercer y cuarto lugar, con un 75% de repuestas entre 4 y 5 tenemos los dolores 1 (“El equipo de Data Center tiene problemas de prolijidad al aplicar los cambios”) y 10 (“La duración de las etapas de paso a producción toman mucho tiempo”). Con la implantación de OpenShift se percibe que hay una labor en la que ya no se requieren los servicios del Data Center, lo que elimina los problemas de prolijidad. Por otro lado, la automatización del paso a producción elimina algunas etapas de coordinación entre los distintos actores involucrados, con lo que estas etapas reducen considerablemente su tiempo de ejecución.

El dolor 2 (“queremos disminuir daños colaterales después de cada cambio y paso a producción”) recibió un 58% de respuestas positivas, y un 25% de respuestas que consideran que la situación queda igual. Esto es así porque, si bien las técnicas DevOps ayudan en esto (a través del testing automatizado, por ejemplo), se percibe que esto depende de la calidad en el análisis y diagnóstico adecuado del problema, algo que depende de las capacidades de análisis del equipo de desarrollo, más que de herramientas.

El dolor 4 (“los distintos ambientes no están homologados”) tuvo una respuesta sorprendente para el autor, pues se esperaba que fuera más alta. Es probable que no esté claro que los contenedores aseguren que esto se cumpla. Otra posible razón es que durante la prueba de concepto hubo una falla en la configuración de parámetros de los ambientes de Producción, lo que pudo haber impactado negativamente en esta nota.

El dolor 5 (“los costos del proceso de desarrollo se deben mantener dentro de un margen razonable”) y 8 (“el esfuerzo de validación de los cambios es excesivo”) tuvo 50% de respuestas entre 4 y 5. Este dolor es el único que tuvo una respuesta indicando que puede ser peor. No hay claridad por parte de los entrevistados que estas técnicas impacten en la disminución de los costos de desarrollo. La opinión general es que estos dependen del costo de personal y de la adquisición de licencias y aumento en el uso de infraestructura, valores que históricamente sólo han ido en alza.

El dolor 8 (“El esfuerzo de validación de los cambios es excesivo”) también tuvo un 50% de respuestas entre 4 y 5, y un 33% de los entrevistados consideraron que se puede mantener igual. Esto tiene que ver con que los entrevistados perciben que hay un problema de cobertura de las pruebas que aún no es bien resuelto ni por el proceso ni las herramientas que se usan.

Los dolores que tienen la menor cantidad de evaluaciones positivas son el número 6 (“El diagnóstico de las fallas en producción toma mucho tiempo y el primer diagnóstico no siempre es el adecuado”) y el número 7 (“Es esencial cumplir los plazos comprometidos por desarrollo”). El dolor 6 tiene un 33% de respuestas que indican que la situación se mantiene igual, y de acuerdo con los entrevistados es así porque depende del conocimiento de negocios y de las aplicaciones que tengan los equipos y eso se resuelve con otras técnicas no abordadas en la prueba de concepto. Con respecto al dolor 7, se cree que se mantendrá igual porque este es un factor externo impuesto muchas veces, entonces es difícil cambiarlo con el proceso o con automatización.

6. Conclusiones

Sobre el trabajo realizado

El trabajo expuesto en esta tesis se ejecutó en tres fases: Descubrimiento, Construcción y Validación.

En la fase de Descubrimiento se invirtió gran parte del esfuerzo en determinar los dolores de la organización con respecto al proceso de desarrollo y a la entrega de software. Gracias a la participación de diversos actores se pudo identificar un conjunto de dolores relevantes que sirvieron de guía para las etapas siguientes.

En la fase de Construcción se definieron los objetivos y principios que conformaron el marco de referencia para el diseño de una arquitectura que solucionara los problemas detectados en la fase de Descubrimiento. Antes de establecer la solución tecnológica, se revisó el proceso de desarrollo y se identificaron las actividades que se verían impactadas por la arquitectura. Teniendo claro los cambios necesarios en el proceso, la arquitectura se alineó a los principios establecidos, de modo que se lograra que los cambios tuvieran un impacto positivo en el proceso. Durante este tiempo, además determinamos que dada la realidad cultural de Previred, junto con las necesidades del proceso, la técnica DevOps más adecuada para la organización corresponde al proceso de Entrega Continua.

Una vez definida la arquitectura deseada, se realizó un estudio para evaluar las mejores herramientas que permitirían implantar esta solución. Realizada la selección de herramientas, se diseñó y ejecutó una prueba de concepto (PoC, por su sigla en inglés, Proof of Concept) que permitió validar la factibilidad técnica de la solución, y dar viabilidad al piloto planeado como objetivo de la tesis.

Finalmente, durante la etapa de Validación, se ejecutó un piloto con un proyecto actual de Previred, y que se encuentra ya operando en producción. Además, se realizó una evaluación cualitativa que permitió validar si la solución presentada era percibida como una solución adecuada a los problemas detectados en la fase de Descubrimiento.

Evaluación general

Tal como se indicó en la Sección 5.3.3, si convertimos los resultados de la evaluación de la solución, a una escala con nota de 1 a 5, el valor obtenido por la solución, según la opinión de los principales interesados en la misma, fue de un 4.1, lo que se traduce en una percepción de que esta solución mejorará la situación actual. Por otro lado, ocho de diez de las respuestas tienen un porcentaje de respuestas positivas mayores al 50%, con lo que podemos concluir que hay una percepción de los evaluadores de que la solución es de buena calidad, y que ayuda a reducir los principales dolores detectados en la fase de Descubrimiento. Por otro lado, la reducción de los tiempos de implantación, la reducción de errores, y la automatización del proceso le da más seguridad al equipo y a los usuarios, lo que ha redundado en una percepción de mayor agilidad por parte de los interesados en el proceso.

Los resultados obtenidos muestran que se valora positivamente la solución propuesta, lo que se considera satisfactorio para el cumplimiento de los objetivos y alcance de esta tesis.

Logros

El objetivo general de implementar un proceso de integración y entrega continua para la organización se logró en gran medida ya que la prueba de concepto demostró que era factible técnica y económicamente, y la posterior validación con el piloto y la evaluación por parte de los interesados demostró que estos perciben que el proceso y la plataforma resuelven los principales dolores.

Con respecto al logro de los objetivos específicos, podemos destacar que:

- Fue posible implantar el proceso de integración continua para el desarrollo de software, lo cual requirió modificar el proceso de desarrollo. En este trabajo fuimos aún más allá, logrando implantar un proceso de entrega continua.
- Con respecto al objetivo de implantar la creación y despliegue de los ambientes mediante contenedores orquestados y el despliegue de las aplicaciones en estos ambientes, el logro fue absoluto, lo que queda demostrado con el piloto.
- Por último, este trabajo permitió agregar pruebas automatizadas y de regresión al proceso de desarrollo de software.

Impacto

Debido al éxito del piloto, junto a la buena recepción que los principales interesados mostraron acerca de la plataforma propuesta, Previred decidió adquirir la suite completa del producto OpenShift e implantar la plataforma para destinarla a servicios en modo de producción. La empresa definió una hoja de ruta para la adopción de la plataforma, donde primero se utilizará para la construcción e implantación de un nuevo servicio y, posteriormente, en la conversión y migración paulatina de los servicios existentes. Luego de la adquisición y configuración de la plataforma, Previred ha implantado exitosamente dos servicios adicionales usando el proceso y las herramientas definidas en este trabajo de tesis.

El mayor impacto percibido ha sido en el proceso. La incorporación de un servidor de integración continua, junto con las pruebas automatizadas, de manera gradual en algunos proyectos, se ha evaluado como positiva por parte de los equipos de desarrollo y testing. Se percibe mayor confianza en la entrega y algunas métricas de calidad han mejorado.

Trabajo futuro

Una línea de trabajo en el mediano plazo es la inclusión, en el proceso y la plataforma, de técnicas que permitan integrar el monitoreo de los servicios a la gestión de incidencias y a la planificación de las mismas. De esta forma, al acercarse al monitoreo continuo, la Gerencia de TI sería más proactiva en la generación de valor atendiendo a las exigencias y uso de los usuarios, y sería preventiva en lo relacionado a la resolución de problemas en el ambiente de producción.

Por último, se propone también definir un proceso de mejora continua, en la que la Gerencia de TI evalúe periódicamente sus procesos y su plataforma tecnológica, y realice

las mejoras necesarias de forma de mantener alineada la operación con los objetivos estratégicos de la Gerencia, buscando reducir los dolores organizacionales.

Lecciones aprendidas

El principal aprendizaje obtenido es que es necesario validar las percepciones propias, como responsable de un área de desarrollo, comparándolas de una manera sistematizada y ordenada con las percepciones de los ejecutores del proceso. Si bien el autor de esta tesis tenía una idea a priori de los dolores, el estudio detallado de los mismos, mediante entrevistas a todos los involucrados en el proceso, permitió ampliar la visión. Esto además permitió priorizar ciertos dolores, y descubrir algunos que el autor desconocía. Por ejemplo, no tenía una percepción de que el diagnóstico de fallas fuera percibido como un dolor importante, lo mismo que el exceso de papeleo fuera un dolor tan relevante para el equipo.

La importancia de la cultura organizacional

Otro descubrimiento importante, que el autor quiere destacar, es que DevOps no es una metodología, o receta que deba implantarse directamente en cualquier organización. Las técnicas y herramientas de DevOps deben ser adaptadas a la realidad y cultura de cada organización.

Por ejemplo, el autor aspiraba a preparar el camino para la adopción de Implantación Continua por parte de Previred. Pero un análisis detallado del proceso y las prácticas, mostraron que esta aspiración chocaba con la cultura y las necesidades de control de la organización. No se trata de un problema de resistencia al cambio, sino que es parte el proceso y una necesidad cultural el que exista una instancia de inspección del usuario y del área que opera el sistema, antes de pasar a producción. Eliminar esta instancia, si bien puede agilizar aún más la entrega de software, choca con otras necesidades de control establecidas, definidas por necesidades de continuidad de negocios, auditorías, entre otras causas.

DevOps requiere una cultura madura y colaborativa

Implantar DevOps es complejo técnicamente, y por ello, requiere un grado de colaboración entre dos áreas que normalmente operan de forma contrapuesta.

Es por esto que en Previred junto con trabajar en los aspectos más duros y técnicos, nos hemos preocupado de que se perciba este trabajo como algo colaborativo entre los equipos de desarrollo y sistemas. En esto es esencial obtener la colaboración de los líderes de equipo, que se alinearon para el logro de este objetivo. Un aspecto clave para poder lograr este alineamiento fue descubrir los dolores y socializarlos entre los líderes y los ejecutores del proceso. Al tener claro que con este trabajo podíamos abordar dolores concretos, el esfuerzo de alinear a los equipos para la obtención de resultados fue menor. Tanto usuarios como programadores, testers e ingenieros de sistemas, sintieron que estas técnicas y herramientas les ayudarían a realizar un mejor trabajo.

Cierre

La experiencia de ejecutar esta tesis ha sido enriquecedora para el autor. No sólo por el logro tecnológico, que es relevante y de gran impacto, sino porque con esta labor pudo trabajar coordinando y liderando un equipo multidisciplinario, que además incluyó proveedores externos. El hecho de que la solución fuera bien evaluada, junto con la confirmación por parte de la organización para continuar con el proyecto, constituyen una gran satisfacción laboral, profesional y personal.

7. Bibliografía

[1] G. Gruver. &. T. Mouser, "Leading The Transformation", Kindle Edition, 2015.

[2] V. Farcic, "The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Micro Services, Leanpub, 2016.

[3] G. Kim, "The Phoenix Project. A Novel About IT, DevOps, and Helping Your Bussiness Win", Kindle Edition, 2013.

[4] Gene Kim, Jez Humble, Patrick Debois, John Willis The DevOps HandBook, How to create worldclass agility, reliability & security in technology organizations, Kindle Edition, 2016.

[5] Martin Fowler, «"Continuous Delivery",» [En línea].: http://martinfowler.com/bliki/ContinuousDelivery.html. [Último acceso: diciembre 2018].

[6] «"What is a Container",» [En línea]. Available: https://www.docker.com/what-container. [Último acceso: diciembre 2018].

[7] Scott Ambler, Pramod J. Sadalage, "Refactoring Databases", Kindle Edition, 2006.

[8] «Learn About SCRUM,» [En línea]. Available:https://www.scrumalliance.org/learn-about-scrum . [Último acceso: diciembre 2018].

[9] «Introducing GItFlow,» [En línea].: https://datasift.github.io/gitflow/IntroducingGitFlow.html. [Último acceso: diciembre 2018].

[10] A. S.A., «Atentus, monitoreo y control de calidad web,» [En línea].: http://www.atentus.com/.[Último acceso: diciembre 2018]

[11] Nagios, «https://www.nagios.com/solutions/application-monitoring/,» [En línea] [Último acceso: diciembre 2018].

[12] D. Cohen y. B. Cabtree, ««Qualitative Research Guidelines Project»,» [En línea].: http://www.qualres.org/HomeInte-3595.html. [Último acceso: diciembre 2018]

[13] N. Rozanski. &. E. Woods, Software Systems Architecture, Addison-Wesley, 2005.

[14] «"Kubernetes Concepts, Pods",» [En línea].: https://kubernetes.io/docs/concepts/workloads/pods/pod/. [Último acceso: diciembre 2018].

[15] «Routing in RedHat OpenShift Container Platform with F5 Just Got Easier,» [En línea]. Available: https://devcentral.f5.com/articles/f5-friday-routing-redhat-openshift-container-platform-with-f5-just-got-easier-25683. [Último acceso: diciembre 2018].

[16] «"Guía de productos Atlassian",» [En línea].: https://es.atlassian.com/software. [Último acceso: diciembre 2018].

[17] «Wikipedia Artículo: Application Lifecycle Management (ALM),» [En línea]. Available: https://en.wikipedia.org/wiki/Application_lifecycle_management. [Último acceso: diciembre 2018].

[18] S. Sadeghianfar, «CI/CD with OpenShift,» [En línea].: https://blog.openshift.com/cicd-with-openshift/. [Último acceso: diciembre 2018].

[19] Wikipedia, «DevOps,». [En línea].: http://en.wikipedia.org/wiki/DevOps. [Último acceso: diciembre 2018]

[20] «Continuous Delivery vs Continuous Deployment: What's the Diff?,» [En línea].: https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff. [Último acceso: diciembre 2018].

[21] «Introducing RedHat OpenShift Platform,» [En línea].: https://blog.openshift.com/introducing-red-hat-openshift-container-platform/. [Último acceso: diciembre 2018].

[22] «Spring Cloud Config» [En línea].: https://spring.io/projects/spring-cloud-config. [Último acceso: diciembre 2018].