Experiencias de Implementación de Despliegue Continuo con una Infraestructura "Conteinerizada"

Carlos S. Moreno, Heraldo R. Valenti, Diego M. Rubio, Claudio J. Gonzalez
Laboratorio de Investigación y Desarrollo en Ingeniería y Calidad del Software, Universidad Tecnológica Nacional – Facultad Regional Córdoba,
Ciudad Universitaria, Maestro M. Lopez esq. Cruz Roja s/n, 5000Córdoba, Argentina
{ingcsmoreno, heraldovalenti, rubiodiego, claudiojgonzalez}@gmail.com

Abstract

En los últimos años cada vez más organizaciones de desarrollo de software reconocen la importancia de desplegar rápida y continuamente software. Esto ha llevado a la unión de áreas de desarrollo y operaciones para la automatización de dichas entregas y las pruebas a través de los distintos ambientes. En este trabajo se presenta un ejemplo de implementación de un ambiente completo de despliegue continuo utilizando para ello las ventajas y portabilidad otorgadas por ambientes "conteinerizados". Junto con la implementación piloto se presentan las lecciones aprendidas de dicho ambiente y las conclusiones de la experiencia.

1. Introducción

En los últimos tiempos, cada vez más organizaciones de desarrollo de software reconocen la importancia de entregar rápidamente software fiable y de alta calidad [1]. Sin embargo, tienen dificultades para implementar enfoques y prácticas del proceso de entrega de software adecuados [1]. Por ello se preocupan por establecer un proceso de desarrollo y entrega de productos de software, confiable y estable; y por desarrollar un plan personalizado que permita mejorar tanto el proceso de entrega, como el flujo de información entre las personas involucradas, a través de prácticas y herramientas para realizar, automatizar y adaptar dicho plan [1].

Para llegar a completar este objetivo es que surgen tendencias como las llamadas "operaciones ágiles", que basan su aplicación en enfoques ágiles de trabajo en el área de operaciones; o la de generar una comprensión más amplia del valor de la colaboración entre los equipos de desarrollo y el personal de operaciones a lo largo de todas las etapas del ciclo de vida del desarrollo de software, haciendo hincapié en la importancia que el área de operaciones ha ido adquiriendo en nuestro mundo cada vez más orientada a los servicios. De la unión de estas dos tendencias, es que surge el nuevo término que denomina a este grupo emergente de prácticas de ingeniería como DevOps [2], y con este, un modelo de Agentes de DevOps [3].

El presente trabajo describe una implementación práctica de este modelo y su implementación en un caso de prueba a modo de validación. Para ello el trabajo se estructura de la siguiente manera:

- La sección 2 realiza una breve introducción a los conceptos utilizados (DevOps y Docker)
- La sección 3 presenta un breve resumen del estado del arte en la implementación de dichos sistemas.
- La sección 4 presenta la experiencia realizada y su implementación.
- La sección 5 presenta las principales lecciones aprendidas.
- Por último se presentan en la sección 6 las conclusiones y futuros trabajo relacionados.

2. DevOps

El término DevOps, corresponde a las siglas de "Development & Operations". Fue usado originalmente por Patrick Debois y Andrew Shafer en 2008 [4], luego, pasó a ser de un uso más común en 2009, gracias a la presentación "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr" de John Allspaw y Paul Hammond, durante la Velocity Conference [5]. Finalmente, se terminó de volver popular al ser usado como hashtag de Twitter durante la devopsday del mismo año, la cual recibió grandes influencias de varias charlas y papers [1].

Como se ha visto en el párrafo anterior, el término existe desde hace algunos años, y si bien su popularidad sigue aumentando, aún no es fácil encontrar una definición clara y consensuada que dé identidad al concepto de DevOps [6]. The Agile Admin lo define como: "La práctica en la que ingenieros de desarrollo y operaciones participan conjuntamente en el ciclo de vida completo del software, desde su diseño, pasando por el desarrollo hasta llegar al soporte de producción." [2]. Andrej Dyck y Ralf Penners enunciaron que DevOps "es un enfoque organizacional que busca la empatía y colaboración inter-funcional entre equipos - especialmente de desarrollo y operaciones TI - en el desarrollo de software, para lograr operar sistemas adaptables a los cambios y acelerar el flujo de cambios" [1]. Gene Kim habla de DevOps como un "movimiento emergente de profesionales que evoca una relación de trabajo colaborativa entre el área de Desarrollo y de Operaciones, resultando en un flujo rápido del trabajo planificado, a la vez que, simultáneamente, se incrementan la confianza, la estabilidad, la adaptabilidad a los cambios y la seguridad del ambiente de producción" [7]. Constantine Aaron Cois y Joseph Yankel adoptan el concepto de "un movimiento dentro de la ingeniería de software que busca alinear el esfuerzo de los ingenieros de software y el staff de operaciones (infraestructura, control de calidad, empaquetado, entrega de productos), para facilitar la transición de los artefactos del proyecto a través de los procesos y herramientas implicadas" [3].

A pesar de esta diversidad, es claro que lo principal de DevOps son sus aspectos culturales como el mantener un buen flujo de información, colaboración inter-funcional, responsabilidades compartidas, y fomentar nuevas ideas. En otras palabras, DevOps busca establecer un enfoque en la colaboración entre equipos con un objetivo común, el de desarrollar software de alta calidad [1].

.

2.1. Componentes Principales del Sistema de DevOps:

Siendo la búsqueda del desarrollo de software de calidad, y la agilización de los periodos de entrega dos de los principales aspectos en los que DevOps hace hincapié [7] [3] [1], la adición de los sistemas automatizados al proyecto permite que ciertas tareas de comunicación y administración de información sean delegadas a herramientas o componentes del sistema de DevOps, dejando a los actores humanos tareas especializadas que una computadora sea incapaz de realizar, como el diseño e implementación del software. Estos actores del sistema deben realizar sus tareas asignadas de la manera más autónoma posible, de modo que la intervención de actores humanos se mantenga al mínimo [3].

A continuación se listan los principales actores del sistema [3], y en la ilustración 1 puede verse de forma gráfica el modo en que se relacionan entre sí:

- **Sistemas de Control de Versionado**: Sistemas de almacenamiento y administración del historial de versiones tanto del código fuente como de cualquier otro artefacto necesario para el sistema de software. *Herramientas comúnmente usadas*: GIT [8], Mercurial [9], SVN [10], RTC SCM [11].
- **Administración de Tareas**: Sistemas para llevar control sobre las tareas a realizar y sus estados. *Herramientas comúnmente usadas*: JIRA [12], IBM Rational Team Concert [13], Redmine [14].
- **Sistemas de Integración Continua**: Sistemas para compilar y probar el software produciendo como salida una aplicación funcional. *Herramientas comúnmente usadas*: Jenkins [15], TeamCity [16], GitlabCI [17], Bamboo [18].
- **Sistemas de Documentación**: Sistemas para crear, almacenar, distribuir y mostrar documentación del proyecto de software. *Herramientas comúnmente usadas*: Google Drive [19], SharePoint [20], Confluence [21].
- **Sistemas de Revisión de Código**: Sistemas para realizar revisiones técnicas de código sobre los cambios a ser introducidos en el código fuente del proyecto de software. *Herramientas comúnmente usadas*: ReviewBoard [22], Gerrit [23].
- **Sistemas de Monitoreo**: Sistemas para controlar el estado y funcionalidad de todos los demás sistemas. *Herramientas comúnmente usadas*: Hobbit [24], Zabbix [25], Sensu [26].
- **Sistemas de Comunicación**: Sistemas responsables de comunicar a los agentes humanos la información generada, así provenga de un agente de sistema, u otro agente humano. *Herramientas comúnmente usadas*: SMTP Servers, Skype [27], Google Hangouts [28], ICQ [29], HipChat [30].

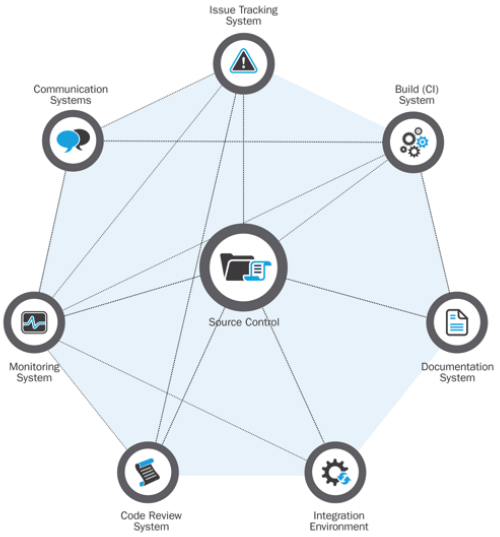

Ilustración 1: Actores principales del modelo de DevOps (3)

Con este modelo, el concepto de DevOps se enuncia como un movimiento emergente que engloba actividades transversales a todo el ciclo de vida del proceso de software, desde el desarrollo de código, hasta tareas administrativas y operacionales. Para finalizar y como se mencionó con anterioridad, DevOps promueve que todas las tareas que no requieran exclusivamente de la intervención humana pueden ser catalogadas y delegadas a actores del sistema preparados realizarlas de manera automática [31].

2.2. Docker

Siendo un producto relativamente nuevo construido sobre la tecnología de contenedores de Linux que rápidamente se ha consolidado desde su lanzamiento en el año 2013, Docker es una plataforma que permite la creación, ejecución y gestión de ambientes aislados para las aplicaciones de software (conocido generalmente como "conteneirización") mediante un conjunto de herramientas integradas que brindan solución algunas de las problemáticas principales que se presentan cuando se debe implementar la gestión de ambientes de ejecución [32].

Su principal funcionalidad consiste en la generación de ambientes de ejecución que permite el empaquetado en imágenes livianas de todas las dependencias requeridas por las aplicaciones que vivirán en él [32]. Estas imágenes luego pueden ser distribuidas, y, al contener su entorno de ejecución completo, pueden ser ejecutadas en contenedores, lo que permite asegurar que las aplicaciones se comportarán de la misma manera en todas las ejecuciones [32] [33].

Docker surge como alternativa frente a la clásica técnica de virtualización de servidores con un enfoque arquitectónico distinto que permite la reutilización de ambientes de ejecución con un mejor uso de los recursos de hardware, principalmente de espacio en disco [33, 32].

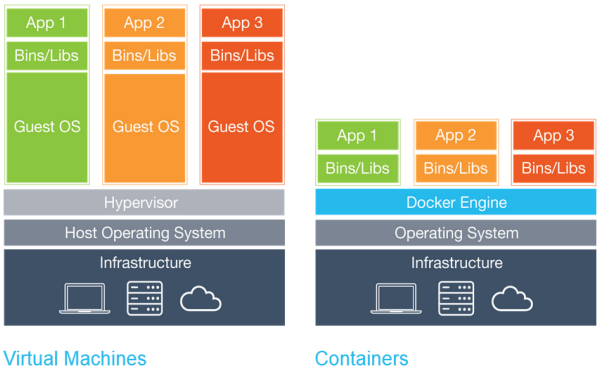

Ilustración 2: Arquitectura de VM vs Docker (9)

Como se puede observar, el componente Docker Engine brinda las funciones que en una máquina virtual serían provistas por el sistema operativo invitado, reduciendo en el orden de los gigabytes el uso del espacio en disco por cada imagen generada [33].

Entre las principales problemáticas que surgen cuando se requiere la gestión de ambientes de ejecución se pueden mencionar el control del ciclo de vida de los ambientes, su distribución, orquestación, escalado de los mismos y comunicación entre nodos distribuidos, para los cuales Docker provee herramientas que brindan una solución lista para ser configurada y utilizada [34] [33]. A continuación se listan las herramientas utilizadas durante el desarrollo del presente trabajo:

- Docker Machine: provee las funcionalidades para realizar la conexión a un nodo o un clúster de los mismos y controlar el ciclo de vida de los contenedores que se ejecutan en ellos [33].
- Docker Registry: es una herramienta que permite la distribución de imágenes de una manera automática [33].
- Docker Compose: resulta útil para llevar a cabo la orquestación de varios contenedores que componen un ambiente de ejecución mediante archivos declarativos [33].

3. Estado del Arte

Recientes estadísticas sobre la adopción de DevOps en las organizaciones, sugieren que ya no sea considerado solo un movimiento y un conjunto de herramientas, sino un área de TI bien establecida. En muchos casos, el área de DevOps, ya cuenta con un presupuesto y un equipo dedicado de soporte [6]. Estas iniciativas son a menudo llevadas adelante por pequeños equipos de expertos en despliegues automatizados,

administración de datos e integración continua. En encuestas realizadas [6], un 58% de las organizaciones encuestadas respondieron que tienen menos de 50 activos dedicados a dar soporte a DevOps en su organización, mientras otras llegan a superar los 500. Además, más del 80% de las organizaciones que implementan prácticas de DevOps, consolidan tal responsabilidad en un único equipo dedicado [6].

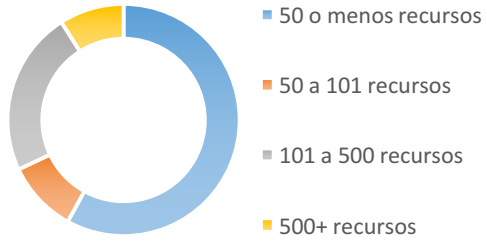

Ilustración 3: Cantidad de recursos dedicados a DevOps en las organizaciones (6)

Mientras DevOps crece más y más en popularidad, y si bien se puede considerar que ya ingresado al mundo empresarial, el movimiento en si aún carece de una definición que haya sido fuertemente aceptada en la industria. Dos organizaciones separadas bien podrían implementar un conjunto de técnicas y herramientas comunes, y sin embargo seguir prácticas diferentes. Algunas organizaciones integran DevOps a las actividades del día a día del departamento de IT, mientras que otras, con una definición más afianzada o madura, de DevOps, tienden a crear equipos dedicados para que sean los responsables exclusivos de llevar adelantes las iniciativas del área, y estas últimas son las que logran obtener el mayor provecho de DevOps [6].

La vasta mayoría de las organizaciones creen tener

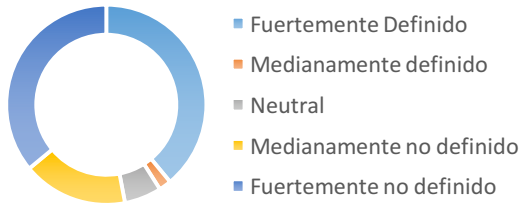

Ilustración 4: ¿Que tan bien definido está DevOps en la organización? (6)

bien definido el concepto de DevOps. Sin embargo, el mismo estudio sugiere que las tácticas y prácticas varían dramáticamente entre todas las organizaciones relevadas. Las organizaciones cuyas iniciativas hayan partido de equipos de desarrollo, tienden a enfocarse más en integración continua, mientras que las que surgieron de equipos de operaciones, se inclinan a hacer foco en despliegue automático en ambientes Cloud privados [6].

Es por eso que se han explorado y analizado las prácticas, métodos y herramientas de los líderes de hoy en día que han logrado llevar DevOps al éxito [6].

En el ambiente de la implementación práctica del entorno de herramientas de DevOps a realizar, se optó por la tecnología de contenedores por su rápida proliferación y la tendencia, por parte de los proveedores de la nube a ofrecer servicios basados en ellos, como así también el nacimiento de nuevos proveedores alrededor de este servicio. En este contexto, y dado que Docker ha sido adoptado como el producto de facto cuando se trata de tecnologías de contenedores, es que se hizo uso de este producto para diseñar y ejecutar nuestros ambientes y herramientas [35] [36].

4. Experiencia e Implementación piloto

Con el fin de probar si la implementación funcional de un entorno de DevOps con ambientes "conteinerizados" es posible siguiendo los conceptos del modelo, se comenzó a desarrollar los ambientes descriptos. Se utilizaron contenedores de Docker para orquestar la infraestructura necesaria [33], y un set de herramientas de código abierto para los agentes. Dada la variedad de herramientas disponibles en el mercado, se optó por este conjunto por su facilidad de licencias y la elevada cantidad de información disponible en internet brindada por sus respectivas comunidades [37].

La primer herramienta a resolver es el sistema de gestión de versionado en donde alojar y controlar los cambios en el código fuente. Hoy en día existen muchas opciones alojadas en la nube que pueden consumirse como servicio, entre las más populares se encuentran GitHub [38], y GitLab [39], que además de proveer repositorios, brindan además funcionalidades como autenticación y autorización de usuarios, y revisiones de código [40]. La segunda opción, cuenta además, con la posibilidad de ejecutarse en un servidor local para una instalación en sitio [39]. Al utilizar Docker para orquestar el ambiente de desarrollo, basta con iniciar un contenedor con una imagen de GitLab como servicio, y otros dos contenedores con una base de datos Postgres [39], para persistir los datos de configuración del servicio, y uno con un servicio de Redis [39] [41], para mejorar las llamadas al servicio a través del uso de caché. Para ejecutar estos contenedores de manera correcta, creamos un archivo docker-compose donde se especifican cómo deberán interactuar los contenedores.

```
postgresql:
  image: sameersbn/postgresql:9.4
  .
gitlab:
  image: sameersbn/gitlab:7.12.2
  links:
    - redis:redisio
    - postgresql:postgresql
  ports:
    - "80:80"
  .
  redis:
  image: sameersbn/redis:latest
```

Lo siguiente a cubrir es la herramienta que vamos a usar para automatizar el proceso de Integración Continua [3], lo que nos permitirá, partiendo del código fuente alojado en GitLab, obtener un artefacto de software funcional listo para su distribución. La opción seleccionada para dar respuesta a esta necesidad es Jenkins [42], que nos permite generar tareas para ejecutar la compilación de código, generación de reportes de análisis estático de código, y del resultado de las suites de Pruebas Unitarias, y prepara un artefacto listo para distribuir. Para las tareas de generación de reportes, vamos a delegar su ejecución a SonarQube, la plataforma de administración de calidad, que nos va a permitir, además, llevar un historial de la evolución del proyecto a través de los builds que se vayan ejecutando a medida que pasa el tiempo [43]. Además, el almacenamiento de los artefactos generados va a ser delgado a Nexus, el repositorio de artefactos universal [44]. Añadimos estos tres servicios como contenedores Docker, y los comunicamos entre sí a través de una composición realizada con docker-compose.

```
nexus:
  image: "sonatype/nexus:2.11.4-01"
  ports:
    - "18081:8081"
jenkins:
  image: "lidicalso/jenkins:1.642.1"
  ports:
    - "18080:8080"
  links:
    - nexus:nexus
    - gitlab:gitlab
    - sonar:sonar
sonar:
  image: sonarqube:4.5.6
  ports:
    - "19000:9000"
    - "19092:9092"
  links:
    - sonardb:sonardb
```

La última herramienta que vamos a implementar es Redmine, la encargada de administrar las tareas del proyecto y de llevar la trazabilidad desde los requerimientos al código fuente [45]. Agregamos esta herramienta a la composición de Docker para que se ejecute junto con los demás servicios y pueda interactuar con los mismos.

```
postgresql:
  image: sameersbn/postgresql:9.4
  .
redmine:
  image: sameersbn/redmine:3.0.3-1
  links:
    - postgresql:postgresql
  ports:
    - "81:80"
  .
```

De este modo, la distribución e interacción entre las herramientas se lleva a cabo de la siguiente manera.

- **Sistemas de Control de Versionado**: GIT [8]
- **Sistema de Integración Continua**: Jenkins [15] (Con Sonar y Nexus para herramientas de soporte)
- **Sistemas de Administración de Tareas**: Redmine [14]
- **Ambiente de Integración**: Docker [46]

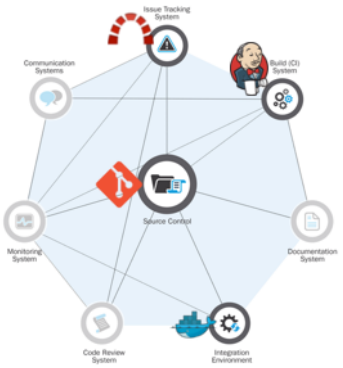

Ilustración 5: Herramientas implementadas como agentes de DevOps

Los mismos se comunican de la siguiente manera:

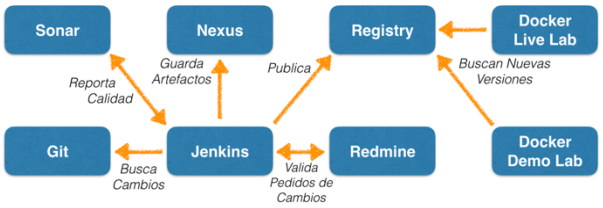

Ilustración 6: Diagrama relacional de las herramientas implementadas

El código fuente del proyecto, alojado en un repositorio de GIT, es descargado por Jenkins. Luego se ejecutarán los procesos correspondientes para compilar, y, a través de SonarQube, analizar el código determinando su calidad respecto de a ciertas métricas previamente definidas, como pruebas unitarias, o porcentaje de pruebas exitosas. Si el análisis de Sonar tiene un resultado exitoso, Jenkins sube los paquetes generados durante la compilación a un repositorio de artefactos (Nexus) para que estén disponibles para su uso en las tareas subsiguientes. Seguidamente, Jenkins ejecuta el comando para crear una nueva imagen de Docker, que contenga la versión de software compilada, y sube la misma a un repositorio privado con las etiquetas de la versión correspondiente, y la etiqueta "latest", para marcarla como la última disponible.

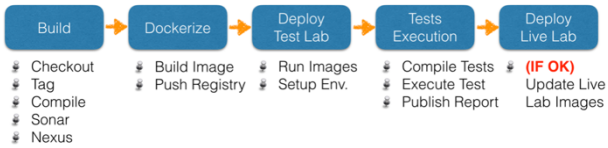

Ilustración 7: Diagrama de flujo de integración continua

Los ambientes de Docker "Test Lab" y "Live Lab" ejecutan contenedores de las imágenes creadas por Jenkins. Una vez que una nueva imagen es marcada como "latest", esta se descarga y ejecuta en el ambiente "Test Lab", y una vez en ejecución, Jenkins lanzará una suite de test para comprobar el estado y calidad de dicha imagen. Por último, si el porcentaje de pruebas exitosas

es aceptable, la imagen docker testeada será marcada (con un Tag) como "Live". Con esto, el ambiente "Live Lab" descargará la imagen para reemplazar los contenedores en ejecución con la nueva versión estable disponible.

De este modo, cualquier cambio introducido en el código fuente, puede, automáticamente, ser probado y llevado al ambiente de producción en cuestión de minutos.

5. Lecciones aprendidas

Durante la realización de la experiencia surgieron interrogantes con respecto a aspectos particulares de la implementación que consideramos pueden ser útiles para futuras implementaciones del modelo de DevOps.

Interacción entre herramientas

En la etapa de selección de las herramientas a utilizar, es importante tener en cuenta la capacidad, o posibilidad, de interactuar con las demás herramientas del conjunto, ya sea a través de una funcionalidad integrada, o bien la exposición de una interfaz pública de manera tal que la comunicación entre las herramientas se realice de forma simple y sencilla. Este criterio de selección de herramientas facilitó la automatización del ambiente de despliegue continuo.

Relación entre los agentes de DevOps

Los agentes de DevOps, definidos en la sección 2.1, no solo nos ayudan identificar los roles o tareas que podemos automatizar, sino también a diagramar los flujos de trabajo entre las herramientas. Pensar en la relación que las mismas tendrán, desde una visión del modelo de DevOps, influye positivamente en la generación de sinergia a través de su relación.

Agentes de DevOps como checklist

El contar con estos roles bien identificados, nos permitió también tener una visión de las tareas cubiertas, y definir los próximos pasos a seguir. Es decir, nos ayuda a modo de guía durante la identificación y planificación de tareas de implementación de prácticas de DevOps.

Infraestructura bajo control de configuración

Herramientas como Docker y Docker Compose permiten definir en archivos de texto plano los ambientes en donde se ejecutará cada servicio y la interacción entre ellos. De este modo, nos fue posible llevar un control estricto y regulado sobre los cambios efectuados en ellos y facilitar su reproducción.

Despliegue automático de cada build

Idealmente, luego de cada cambio realizado en el código fuente de la aplicación se dispara una ejecución de build que da lugar a una nueva versión de la aplicación. El uso de Docker Registry para la distribución provee la posibilidad de utilizar un modelo de "pulling" para realizar el despliegue descentralizado de cada build que creemos nos proporcionó algunas ventajas esenciales frente a un modelo de despliegue centralizado. En primer lugar, permite realizar el despliegue de un build en todos los ambientes (desarrollo, pruebas, producción) de la misma manera, resultando en un procedimiento unificado y flexible. Esto deriva en la posibilidad de reproducir nuevos ambientes de ejecución y permitiendo escalar los ambientes de forma simple y sencilla. Por último, nos parece interesante la posibilidad de poder extender este modelo a la provisión del ambiente de despliegue continuo implementado, pudiendo reproducir el ambiente de forma sencilla y escalable.

6. Conclusiones

El presente trabajo se inició con la premisa de validar la aplicabilidad y portabilidad de la implementación funcional de un entorno de DevOps con ambientes "conteinerizados". Para ello se trabajó en todo el ambiente, su "conteinerización" usando Docker y su prueba a través del uso de un software de referencia.

Como resultado, y luego de superar las limitaciones y dificultades de la configuración de los ambientes descriptos, se procedió a utilizar el ambiente en el entorno del laboratorio de investigación (LIDICALSO). Luego, con la finalidad de validar su portabilidad, se utilizó el mismo en el entorno de una empresa local siguiendo los mismos pasos y herramientas mencionadas en el presente trabajo.

Con esto se logró validar la posibilidad de automatización y despliegue, no sólo del software mencionado, sino además de la portabilidad y la capacidad de replicación de la instalación (y posterior actualización) de los entornos necesarios para realizar las prácticas de desarrollo y operación (DevOps).

Si bien en este trabajo se utilizaron herramientas de código abierto que simplificaron su despliegue en la empresa piloto, resta como futuro trabajo la evaluación de las distintas herramientas disponibles para realizar cada uno de los pasos descriptos como un análisis y estudio del uso de las mismas en el ambiente regional de desarrollo de software.

7. Referencias

[1] A. Dyck, *Towards Definitions for Release Engineering and DevOps,* Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop on, 2015, p. 3.

[2] "What is DevOps," [Online]. Available: http://theagileadmin.com/what-is-devops/. [Accessed 22 Noviembre 2015].

[3] C. A. C. y. J. Yankel, "Modern DevOps: Optimizing Software Development Through Effective System Interactions," IEEE, 2014.

[4] J. Willis, "The Convergence of DevOps," 2012. [Online]. Available: http://itrevolution.com/the-convergence-of-devops/. [Accessed Noviembre 24 2015].

[5] J. A. a. p. Hammond, "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr," 23 Junio 2009. [Online]. Available: http://conferences.oreilly.com/velocity/velocity2009/public/schedule/detail/7641. [Accessed 24 Noviembre 2015].

[6] "2015 Annual State of DevOps," Glenaster y Delphix, 2015.

[7] G. Kim, "The top 11 things you need to know about DevOps," IT Revolution Press, 2014.

[8] [Online]. Available: https://git-scm.com.

[9] [Online]. Available: https://www.mercurial-scm.org.

[10] [Online]. Available: https://subversion.apache.org.

[11] [Online]. Available: https://jazz.net/products/rational-team-concert/features/scm.

[12] [Online]. Available: https://www.atlassian.com/software/jira.

[13] [Online]. Available: https://jazz.net/products/rational-team-concert/.

[14] [Online]. Available: http://www.redmine.org.

[15] [Online]. Available: https://jenkins.io.

[16] [Online]. Available: https://www.jetbrains.com/teamcity.

[17] [Online]. Available: https://about.gitlab.com/gitlab-ci/.

[18] [Online]. Available: https://www.atlassian.com/software/bamboo.

[19] [Online]. Available: gsuite.google.com/Drive.

[20] [Online]. Available: https://products.office.com/en-us/sharepoint/collaboration.

[21] [Online]. Available: https://www.atlassian.com/software/confluence.

[22] [Online]. Available: https://www.reviewboard.org.

[23] [Online]. Available: https://www.gerritcodereview.com.

[24] [Online]. Available: http://xymon.sourceforge.net.

[25] [Online]. Available: http://www.zabbix.com.

[26] [Online]. Available: https://sensuapp.org.

[27] [Online]. Available: https://www.skype.com.

[28] [Online]. Available: https://hangouts.google.com.

[29] [Online]. Available: https://icq.com.

[30] [Online]. Available: https://www.hipchat.com.

[31] H. V. y. C. B. Carlos Santiago Moreno, "Exploración de DevOps," LIDICALSO - UTN - FRC, Córdoba, 2015.

[32] D. Inc, "What is Docker?," May 2013. [Online]. Available: https://www.docker.com/what-docker. [Accessed Junuary 2016].

[33] D. Inc, "Understand the architecture," [Online]. Available: https://docs.docker.com/engine/understanding-docker/. [Accessed 10 April 2016].

[34] "What to consider before adopting Docker as part of your Enterprise DevOps Strategy," 29 March 2016. [Online]. Available: http://containerjournal.com/2016/03/29/consider-adopting-docker-part-enterprise-

devops-strategy-2/. [Accessed 16 April 2016].

[35] S. J. Vaughan-Nichols, "Docker brings container technology to the enterprise," 9 June 2014. [Online]. Available: http://www.zdnet.com/article/docker-1-0-brings-container-technology-to-the-enterprise/. [Accessed 14 May 2016].

[36] M. Crosby, "Open Container Format Progress Report," 22 June 2015. [Online]. Available: https://blog.docker.com/2015/07/open-container-format-progress-report/. [Accessed 14 May 2016].

[37] V. F. ,. R. T. Greg Madey, "THE OPEN SOURCE SOFTWARE DEVELOPMENT PHENOMENON: AN ANALYSIS BASED ON SOCIAL NETWORK THEORY," Eighth Americas Conference on Information Systems, 2002.

[38] [Online]. Available: www.github.com. [Accessed 16 April 2016].

[39] [Online]. Available: www.gitlab.com. [Accessed 16 April 2016].

[40] "Documentación," [Online]. Available: https://about.gitlab.com/features/. [Accessed 16 April 2016].

[41] M. Paksula, "Persisting Objects in Redis Key-Value Database," University of Helsinki, Department of Computer Science, Helsinki, Finland.

[42] Jenkins, "Jenkins Pipelines Architecture," [Online]. Available: https://jenkins.io/doc/pipeline/. [Accessed 10 April 2016].

[43] D. Rodriguez, "Measuring Code Quality with SonarQube," 7 August 2015. [Online]. Available: http://www.avioconsulting.com/blog/measuring-code-quality-sonarqube. [Accessed 10 April 2016].

[44] M. Moser, "Concepts & Benefits of Repository Management," Sonatype, Inc, 2015.

[45] R. McRee, "Implementing Redmine for Secure Project Management," GIAC GCPM Gold Certification, 2013.

[46] [Online]. Available: https://www.docker.com.