



Eficiencia con Entrega Continua y Transformación Digital

Tutor: Nicolás Battaglia

Emanuel Nicolás Derito

Marzo, 2021

Universidad Abierta Interamericana

Trabajo Final de carrera presentado para obtener el título de

Licenciatura en Gestión de Tecnología Informática

Resumen

Los componentes tecnológicos, desde hace unos años, pasaron a ser parte indispensable de nuestra actividad cotidiana. Generaciones como Baby Boomers o X fueron incorporando gradualmente la utilización de estos dispositivos, mientras que las nuevas generaciones como Millennials y Centennials, nacieron con toda esta tecnología disponible para su uso, lo que les permitió la naturalización en su aprendizaje y el aprovechamiento de todos sus beneficios.

Por un lado, estas nuevas herramientas tecnológicas no solo modificaron la forma de realizar las tareas diarias, sino que desafiaron a las empresas a cambiar sus procesos y la forma tradicional de efectuar negocios. Por este motivo las organizaciones comenzaron una profunda transformación digital (TD), las nuevas generaciones impulsan la necesidad de contar con soluciones sencillas y rápidas desde distintos sitios y en diversos periodos.

Por otro lado, ya no es suficiente contar con una página web ni facilitar una aplicación para teléfonos móviles. Los usuarios son más exigentes, no solo con el producto que se le ofrece, sino también con el tiempo en el que este se encuentra disponible. Las empresas deben conocer y mejorar constantemente la calidad y disponibilidad de sus aplicaciones si desean mantenerse competitivas en el mercado.

Para ello, en este trabajo se investigaron un conjunto de técnicas y metodologías para desarrollar software, que utiliza ciclos cortos de interacción y un proceso de implementación automático.

Estas técnicas y metodologías permiten crear una aplicación en pocas semanas, resolver errores y/o agregar funcionalidades de forma rápida.

Así, una empresa que utiliza metodologías de trabajo tradicionales, puede mejorar el tiempo de salida al mercado de un nuevo producto (aproximadamente un 75%) aplicando este conjunto de herramientas y técnicas.

Se realizó una revisión sistemática para identificar las características distintivas de las metodologías ágiles y técnicas desarrolladas, como así también un análisis de campo en una reconocida empresa que representa un caso de éxito nacional.

Palabras clave: desarrollo de software ágil, Devops, scrum, integración continua, despliegue continuo, software aplicativo

Abstract

In recent years, technological components have become an indispensable part of our daily lives. While generations like Baby Boomers or X learned little by little to use these devices in their daily activities, new generations like Millennial or and Centennial, were born with all this technology available for their usage, naturalizing their learning from childhood and taking advantage of its benefits.

These new technological tools not only modified the way we carry out our daily activities, but also challenged companies to change their processes and the traditional way of doing business. This is why organizations are going through a deep digital transformation (TD), the new generations drive the need for quick and easy solutions at any time and from anywhere.

It is just not enough anymore to have a website or provide an application for mobile phones. Users are more demanding, not only about the product it is offered, but also about how fast it has to become available to them. Companies must constantly know and improve the quality and availability of their applications if they want to stay competitive in the market.

We researched a set of techniques and methodologies to develop software, which use short cycles of interaction and an automatic implementation process.

This techniques and methodologies allow you to create an application in a few weeks, solve errors and/or add new functionalities quickly.

Thus, a company that uses traditional work methodologies can improve the time to market of a new product (approximately 75%) by applying this set of tools and techniques.

A systematic review was conducted to identify the distinctive characteristics of the agile methodologies and techniques developed, as well as a field analysis in a renowned company that represents a national success story.

Keywords: agile software development, Devops, scrum, continuous integration, continuous delivery, application software

Dedicatoria

A mi hija Juliana y a mi mujer Florencia, las amo.

A mi Mamá fuiste la mejor del mundo, te voy a estar agradecido toda mi vida.

A mi Papá gracias por todo lo que me enseñaste.

A mi hermano Luciano.

Reconocimientos

A mi tutor y profesor Nicolás Battaglia, muchas gracias por todo.

A mi profesora Marcela Samela.

Tabla de Contenidos

Capítulo 1	1
Introducción	1
Hipótesis	2
Objetivo General	2
Objetivos Específicos	3
Tipo de Investigación	3
Capítulo 2	4
Marco Teórico	4
Metodología Ágil	4
Arquitectura de Microservicios	8
¿Qué es DevOps?	9
Cloud (Nube)	11
Contenedores de software	12
Kubernetes	15
Ambientes	16
Integración Continua	17
Git	19
Entrega Continua	20
Jenkins X	22
Capítulo 3	23
Ejemplo de Integración y Entrega Continua	23
Investigación	39

Capítulo 4	43
Caso de Éxito Banco Galicia	43
Introducción	43
¿Por qué Transformación Digital?	43
Desarrollo Ágil	48
Tiempos de despliegue	52
Conclusiones	55
Líneas Futuras de Investigación	57
Bibliografía Complementaria	58
Acrónimos	59
Referencias	60

Índice de Gráficos

Ejemplo tablero Kanban	6
Gráfico Tabla de Comparación de Arquitecturas	9
Diferencias entre Arquitecturas VM y Docker	13
Tiempo Promedio de Inicio	14
CloudShell GCP	25
Aplicación de Ejemplo	35
Muestra estadística recolectada en el mes agosto 2019	52

Capítulo 1

Introducción

Las generaciones actuales, Millennials y Centennials, no rechazan la tecnología y mucho menos realizar operaciones de forma online (compra/venta de cualquier tipo, transferencias bancarias, consultas medicas, etc) como Aguaded-Gomez (2011) afirma:

En estas nuevas generaciones, el ordenador ya compite plenamente con el televisor, con un nivel de penetración del 95% entre niños de 6 a 9 años y del 97% en los mayores —16 a 18 años—. Las conexiones a Internet se encuentran también en los hogares de niños y adolescentes, alcanzando el 82% en los mayores de 10 años. Algo parecido ocurre con el tercer dispositivo definidor de esta nueva generación —y quizás el más revolucionario por su accesibilidad, omnipresencia y universalidad—, el terminal móvil, que alcanza porcentajes del 83% en niños de 10 a 18 años y el 100% a partir de los 17 años. (p.7)

El “Time to Market”, tiempo transcurrido entre el momento en que un producto es concebido hasta llegar a estar disponible en el mercado, deberá ser bajo a fin de lograr características novedosas.

La transformación digital [TD], es un cambio de paradigma que engloba todos los procesos de negocio de una organización, por ende, nos enfocaremos sobre el desarrollo de aplicaciones. Pensar de forma totalmente digital requiere hacer una reingeniería de los procesos, reemplazar procesos manuales por software moderno y rápido.

Con la Metodología Tradicional Cascada [MTC] crear una aplicación, es parte de un proceso rígido con etapas definidas y un cambio en mitad de proyecto puede afectar toda la creación.

El problema de esta metodología es la rigidez y el aburrimiento. Todo tiene que estar por escrito, con supuestos casi inmodificables y planes que deben seguir estrictamente. ¿Alguna vez ha observado que cuando estima el tiempo de duración de alguna actividad, el resultado es el doble, el triple y hasta el cuádruple de lo estimado? Si extrapolamos este fenómeno a un proyecto de gran envergadura, no sería extraño enfrentarnos a una enorme catástrofe. (Dimes, 2015, p. 6)

Sería muy dificultoso dar respuestas rápidas a estas generaciones o innovar en el mercado, si no se puede cambiar de forma ágil o dinámica.

Hipótesis

La combinación correcta de TD, Metodología Ágil Scrum [MAS], Microservicios [MS] e Integración/Entrega Continua [CICD] permite afrontar un proceso de desarrollo de software de manera dinámica, respondiendo a cambios de requerimientos de manera ágil y utilizando una estrategia de instalación de software con automatizaciones programables evitando intervenciones manuales del desarrollador.

Objetivo General

Proponer un conjunto de herramientas y técnicas que responda de manera efectiva ante cambios en requerimientos del cliente, permita agilizar el proceso de desarrollo de software y su implementación en producción sea rápida y controlada.

Objetivos Específicos

- Exponer las razones por las cuales estas herramientas permiten generar software de forma más rápida.
- Explicar la forma correcta de aplicar estas herramientas en una organización de forma productiva.
- Detallar un listado de buenas prácticas recomendadas.
- Realizar una implementación aplicativa como demostración de despliegue continuo.
- Exponer un caso de éxito.

Tipo de Investigación

Se trata de un trabajo de investigación en el cual se realizó un análisis de tipo cualitativo sobre desarrollo de software y la manera de mejorar los tiempos de resolución aplicando metodologías y herramientas de forma innovadora.

Capítulo 2

Marco Teórico

Metodología Ágil

Existen varios tipos de enfoques, pero todos respetan la entrega de valor iterativa e incremental a diferencia de la metodología en cascada, en la que se debe esperar a la finalización de una etapa para iniciar la siguiente.

Scrum

Esta metodología de trabajo posee una serie de buenas prácticas para afrontar proyectos complejos o dinámicos (con requerimientos cambiantes). Se basa en la entrega parcial y regular del producto final: lo que genera como beneficio que el cliente siempre puede probar el prototipo y dar su opinión.

Poppendieck y Anderson (2010) nos dicen:

- Divide tu organización: Equipos pequeños e interdisciplinarios.
- Divide el trabajo: Crear una lista de entregables pequeños y alcanzables, ordenar esa lista por prioridad y estimar el esfuerzo relativo.
- Divide el tiempo: Interacciones con días fijos.
- Optimiza las entregas y los procesos: Inspeccionar el entregable con el cliente para actualizar las prioridades y también realizar una retrospectiva después de cada interacción

(agendar una reunión con el equipo para saber cuáles fueron los puntos fuertes y débiles en el sprint).

Un grupo Scrum está conformado por 3 roles: Dueño del Producto (PO), Equipo y Scrum Master.

Es iterativo, los ciclos o sprints tiene un tiempo de duración fijo, todos los días se reúne el equipo de trabajo y sincronizan sus tareas, para que todos sepan en qué se encuentran trabajando.

Kanban

Es una herramienta de proceso, así como Scrum, pero busca exhibir la cantidad, gestión y flujo de tareas. Las únicas normas en que se basa son:

- Visualiza el flujo de trabajo: Dividir el trabajo en bloques y crear una tarjeta por cada elemento.
- Limita el “Work in Progress”: En el bloque de “en progreso” deben existir un número concreto y alcanzable de elementos a realizar.
- Conoce el “lead time”: El lead time es el “tiempo de ciclo” que debe medirse para, con la experiencia, ir disminuyendo.

Kanban no establece roles, pero eso no significa que no existan dentro del grupo de trabajo. Para ponerlo en práctica se utiliza un tablero como la imagen a continuación, donde las 3 normas están explícitas, en cada hoja se encuentra una tarea y en las columnas el estado en la que la tarea se encuentra.

Ejemplo tablero Kanban

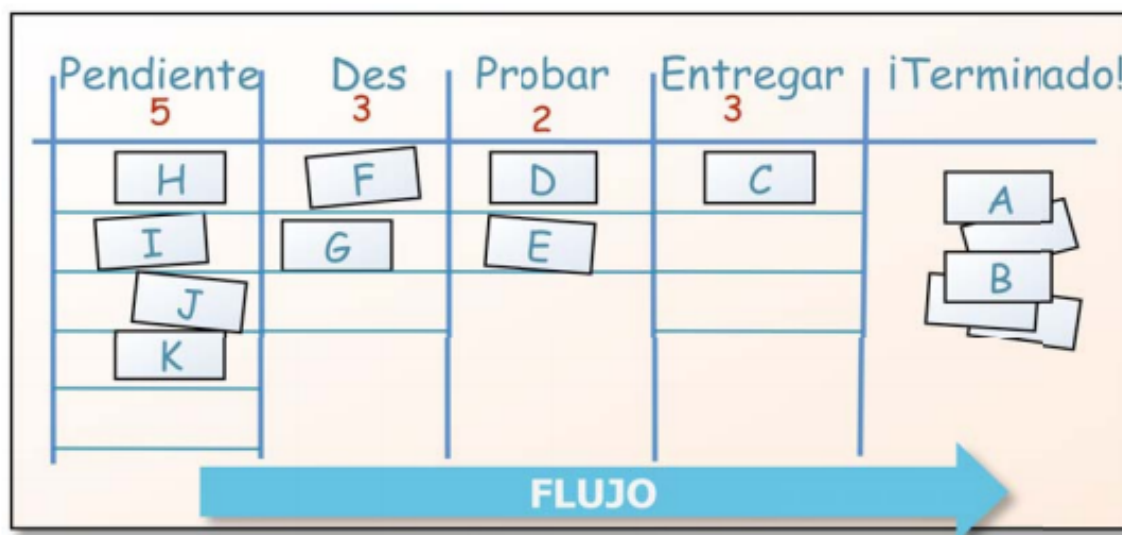


Figura 1. Flujo del tablero Kanban donde cada letra es una historia de usuario. M. Poppendieck – D. Anderson 2010

En este trabajo se va a referir al concepto Scrumban que reúne la naturaleza preceptiva (precepto – orden o mandato impuesto, ordenar) de Scrum y la capacidad de control sobre el trabajo en curso del proceso Kanban. Los dos se encuentran alineados con el Pensamiento LEAN y el Manifiesto Ágil, lo que permite mejorar su eficiencia y eficacia.

Puntos fuertes:

- El equipo elige cuándo y cuánto trabajar – principio “Just in time”.
- Procesos de optimización continuos y empíricos – principio Kaizen
- Fomentan la visibilidad del proceso como base de mejora.
- Todo se visibiliza en el tablero Kanban.

- Entrega temprana y frecuente.
- Equipos auto organizados.
- División del trabajo en partes.
- Scrum tiene roles (PO, SM, etc.)
- Kanban no tiene roles.

La forma de trabajo respeta a Scrum como organización del equipo y a Kanban para dar visibilidad a la célula de trabajo.

Para esta organización de trabajo se realizan 4 ceremonias (reuniones) fijas:

- Todos los días al inicio de la jornada laboral, se lleva a cabo la ceremonia “Daily standup” la cual se utiliza para exponer en qué se va a ocupar el tiempo en el día, que trabas existen y de qué forma se pueden resolver.
- Antes del inicio del sprint se realiza la ceremonia “Planning”, donde el PO selecciona qué requerimientos se van a abordar.
- Al finalizar el sprint se realiza la ceremonia “Review”, donde se muestra a todo el equipo el desarrollo logrado.
- Luego de la finalización del sprint se genera la ceremonia de “Retrospective”, en la cual se revisan las acciones del equipo y se busca ser más eficientes en el futuro.

Ejemplo de Planificación de Sprint:

1. El PO (Dueño del Producto) toma las historias de usuario y selecciona cuales se van a trabajar en este sprint (iteración).

2. El equipo realiza una puntuación sobre esa historia. (La puntuación con base en compromiso, tiempo y costo del trabajo a realizar)
3. Dependiendo del valor de cada historia se calcula cuántas van a entrar en el sprint.
4. Cada historia se divide en tareas y se exponen en el tablero Kanban, para luego cambiarla de columna a medida se vayan realizando.

Arquitectura de Microservicios

El objetivo principal de este diseño es que la aplicación general se base en un conjunto de unidades muy diminutas, ligeras y sencillas llamadas microservicios. Cada uno de estos posee una funcionalidad o tarea única, se conectan entre ellos mediante operaciones http. Las actualizaciones de desarrollo se realizan sobre el microservicio puntual, lo que genera tolerancia a fallos y si uno de estos es interrumpido, la aplicación general sigue disponible. La escalabilidad es muy fácil de resolver ya que se puede crecer en cantidad de microservicios o en la instancia de uno de ellos.

Tiene 4 características principales en la que se basa:

- Enfoque en las capacidades de negocio
- Independencia entre servicios
- Descentralización de datos
- Tolerancia a fallos

Entre las desventajas, encontramos que el mantenimiento al alcanzar un nivel alto de Microservicios, su administración se torna dificultosa.

Gráfico Tabla de Comparación de Arquitecturas

Categoría	Arquitectura Monolítica	Arquitectura de microservicios
Código	Una base de código única para toda la aplicación.	Múltiples bases de código. Cada microservicio tiene su propia base de código.
Comprensibilidad	A menudo confuso y difícil de mantener.	Mayor facilidad de lectura y mucho más fácil de mantener.
Despliegue	Implementaciones complejas con ventanas de mantenimiento y paradas programadas.	Despliegue sencillo ya que cada microservicio se puede implementar de forma individual, con un tiempo de inactividad mínimo, si no es cero.
Lenguaje	Típicamente totalmente desarrollado en un lenguaje de programación.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.
Escalamiento	Requiere escalar la aplicación entera, aunque los cuellos de botella estén localizados.	Cada microservicio puede desarrollarse en un lenguaje de programación diferente.

Figura 2. Tabla comparativa entre Arq. Monolítica y Microservicios. López & Maya (2017).

¿Qué es DevOps?

En 2009 Patrick Debois organiza el primer DevOpsDay, accidentalmente acuñando el término Devops.

Representa un cambio en la cultura IT, enfocado en la rapidez del servicio de delivery con la adopción de agilidad. Busca improvisar la colaboración entre equipos de operaciones y

desarrollo, especialmente con el uso de herramientas programables que sirvan para aumentar y mejorar el ciclo de vida de aplicaciones. (Gartner Glosario, 2015)

Partiendo desde el concepto de que es una cultura de trabajo sobre el desarrollo de software y entre varios puntos busca:

- Evitar los bloqueos entre distintos sectores tecnológicos
- Fomentar la agilidad, las personas están sobre los procesos y herramientas.
- Automatizar todo lo posible para reducir tiempos.
- Reducir los tiempos de implementaciones en Producción.
- Buscar transparencia y visibilidad en el trabajo.

Lo que se busca es que la comunicación y la cooperación entre las personas involucradas sea más empática, ya que al fin y al cabo todos tienen un objetivo en común.

Este pensamiento desencadena que el administrador y el desarrollador tengan otro tipo de relación, la cual anteriormente era estar aislados unos de otros.

En la actualidad nos encontramos que el administrador, el cual fomenta este tipo de cultura, es llamado Devops, ya que su rol está enfocado en compartir este pensamiento y buscar las automatizaciones en todos los procesos a través del desarrollo de scripts (pequeños programas).

Existen varios mitos acerca de Devops, el libro “The Devops Handbook” de G. Kim, J. Humble, P. Debois y J. Willis (2016) concluye:

- DevOps solo sirve en Startups: Google, Amazon y Netflix son ejemplos de que también se puede implementar esta práctica en grandes empresas.
- Devops reemplaza Agile: No solo son compatibles sino que Devops continua la lógica expuesta por el Manifiesto Ágil.
- DevOps es incompatible con Seguridad Informática: Ante la ausencia de controles tradicionales debido a las automatizaciones se piensa que los procesos DevOps son inseguros, pero en realidad dichos controles se encuentran dentro de las automatizaciones.
- Devops es solamente “Infraestructura as a Code”: Es un punto fuerte de esta práctica, pero también se incluye un cambio cultural en normas y arquitectura que permita objetivos compartidos entre Desarrollo de Software y Operaciones IT.

Cloud (Nube)

Resumiendo y sin extendernos sobre el concepto, son aplicaciones listas para usarse como un servicio a clientes a través de internet. Existen Nubes Públicas, Híbridas y Privadas. Su diferencia es quien da el servicio, si un proveedor o la misma empresa. Dentro del servicio a proveer nos encontramos con:

SAAS: Software as a Service su diferencia radica es que es una funcionalidad la que está disponible como servicio, ejemplo: Gmail (correo electrónico), Github (repositorio y control de versiones), Dropbox (alojamiento de archivos), etc.

PAAS: Platform as a Service, es una capa de servicios entre IAAS y SAAS, presta un servicio pero mucho más complejo, como la creación y disponibilidad de aplicaciones. Ejemplo: Red Hat Openshift.

IAAS: Sobre esta categoría se basa la misma “nube”, ya que como una infraestructura tecnológica se da servicio a los clientes, brindando acceso a recursos tecnológicos como contenedores, kubernetes, Machine Learning, IA, etc. Ejemplo: AWS, Google Cloud, Microsoft Azure, etc.

Contenedores de software

Su comportamiento se podría comparar con una máquina virtual (VM), pero la realidad muestra que son bastante diferentes. Un hipervisor o vm comparte recursos como CPU, memoria RAM, espacio de disco y hasta Sistema Operativo con el servidor físico donde está montado y el contenedor es un software de virtualización pero a nivel aplicativo.

El contenedor es un punto clave en “Cloud Computing” ya que de forma dinámica, se pueden crear, administrar y destruir servidores aplicativos. Brinda escalabilidad y flexibilidad a los recursos informáticos que la nube utiliza para dar servicio a los clientes.

Diferencias entre Arquitecturas VM y Docker

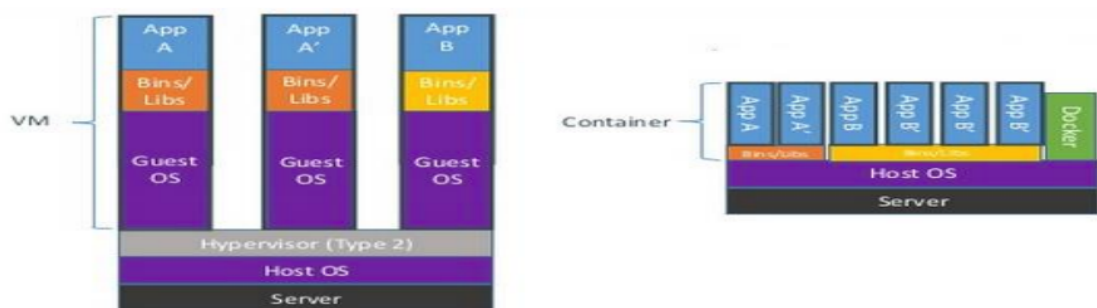


Figura 3. Diferencias entre VM y Docker. Seo, K. T., Hwang, H. S., Moon, I. Y., Kwon, O. Y., & Kim, B. J. (2014).

Como se visualiza en la imagen el uso de la tecnología de virtualización de servidores por aplicación al abstraer capas, tiene a la velocidad como principal beneficio.

Al momento una de las aplicaciones más utilizadas para la creación de contenedores es Docker.

Para crear un contenedor lo primero que se necesita es escribir un Dockerfile, en el mismo se incluye lo mínimo necesario para que el contenedor se pueda ejecutar.

Ejemplo Dockerfile en el cual se va a crear una imagen desde un SO centos 7 con un servidor nginx:

```
FROM centos:centos7
```

```
MAINTAINER emanuelderito@gmail.com
```

```
RUN yum -y install rh-nginc18 && yum clean all && mkdir -p /usr/share/nginx/html
```

RUN echo "imagen de prueba nginx-centos7 tesis-derito" > /usr/share/nginx/html/index.html

EXPOSE 80

USER nginx

ENTRYPOINT ["container-entrypoint"]

CMD ["nginx18"]

Con este archivo y los dos comandos: docker build y docker run, ya se ejecuta un contenedor y se deja disponible para su utilización.

Tiempo Promedio de Inicio

Según un análisis efectuado en el paper de Seo, K. T., Hwang, H. S., Moon, I. Y., Kwon, O. Y., & Kim, B. J. (2014) afirman:

Un contenedor con Docker demora aproximadamente 1,53 segundos en iniciar contra 11,48 segundos en una VM tradicional. El test se realizó con 20 imágenes de cada servidor con las mismas condiciones de Hardware y Software. Una imagen docker demora 10 segundos menos en levantar un contenedor.

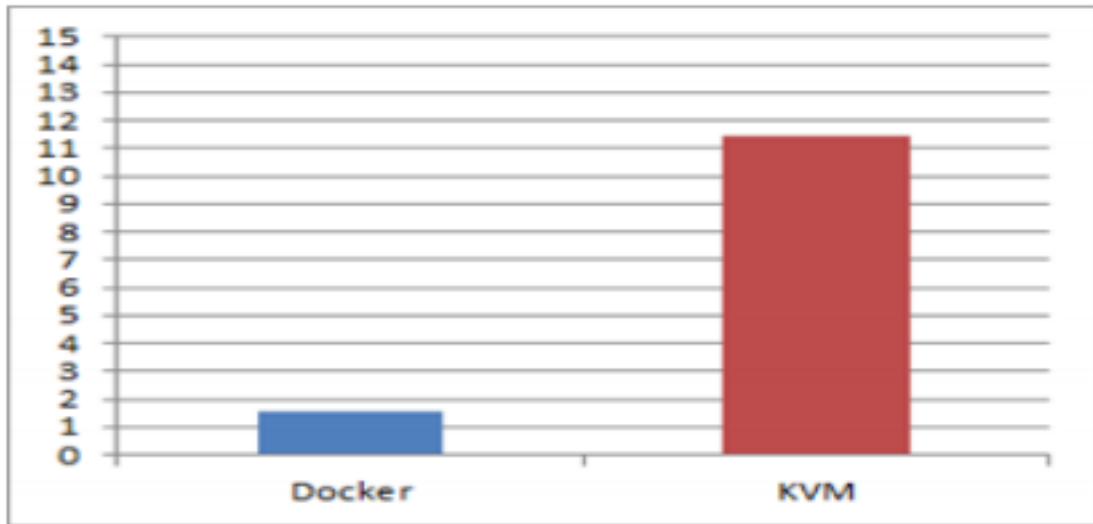


Figura 4. Velocidad de inicio Docker y KVM. Seo, K. T., Hwang, H. S., Moon, I. Y., Kwon, O. Y., & Kim, B. J. (2014).

Comandos básicos:

docker build -t "nombre de la imagen"

Crea una imagen docker a partir de un Dockerfile

docker run "nombre de la imagen" (incluye varios parámetros)

Crea un contenedor con la imagen especificada

De forma simple se puede generar un clúster en segundos, las imágenes se guardan en un repositorio de imágenes y se pueden reutilizar cuantas veces se deseen.

Kubernetes

Es una aplicación Open Source creada por Google, la cual se utiliza para administrar y organizar grupos de contenedores, en 2015 liberaron el código fuente y su utilización. No vamos a adentrarnos en detalle en esta herramienta, solo necesitamos saber que hace uso de un cluster de servidores y proporciona una orquestación a contenedores. Esta aplicación incluye una herramienta para línea de comandos, kubectl.

Tareas básicas:

- Optimizar el uso de hardware del clúster.
- Organizar contenedores en múltiples nodos(servidores).
- Escalamiento, adaptabilidad y organización de contenedores.
- Mantenibilidad en la nube y también de forma híbrida.

Comandos básicos:

kubectl get services *# Genera un listado de servicios en el proyecto.*

kubectl get pods *# Genera un listado de pods (grupo de contenedores)*

kubectl get deployment *# Genera un listado de despliegues*

Ambientes

El objetivo del uso de ambientes es tener escenarios (servidores, clusters, containers, etc) de prueba donde cada uno tiene una característica esencial.

La cantidad clásica de ambientes es: Desarrollo, Homologación y Producción.

El ambiente de Desarrollo sirve para que el equipo realice pruebas, además pueden modificar cualquier tipo de configuración sobre el mismo; el escenario es para el uso del equipo de Desarrollo. Cuando las pruebas realizadas por el desarrollador estén verificadas se puede derivar el código a Homologación.

El ambiente de Homologación sirve para hacer pruebas de integración de la aplicación, donde una persona o sector especializado en calidad pruebe íntegramente las funcionalidades. Cuando la calidad esté asegurada, la integración se encuentre verificada y el responsable de la aplicación dé su aprobación, se va a poder desplegar en Producción.

Producción en cambio, es el ambiente más crítico, es el que usa el cliente.

Los 3 escenarios deben relativamente tener configuración similar en cuanto a hardware y software, siempre teniendo en cuenta que el más crítico es Producción.

Integración Continua

Esta práctica de desarrollo de software nace como respuesta a lo difícil que era anteriormente realizar una integración de funcionalidades de un proyecto.

Cada desarrollador trabajaba sobre su código pero al momento de unificarlo se generaban errores, conflictos en funcionalidades y problemas de coordinación.

El objetivo es que todos los desarrolladores unifiquen sus códigos de forma fácil y rápida, ya que hace un fuerte hincapié en que el proceso de integración sea automático.

Build es crear una aplicación desde su código fuente, en él se encuentran pasos como la compilación y distintos tipos de pruebas automáticas.

Algunas premisas de la CI:

- Build automatizado (mediante scripts o programas).
- Dentro de esta automatización deben ejecutarse los tests(pruebas).
- Dichas pruebas también deben ser automáticas.
- Todos deben actualizar su código cada día.
- Mantener el build rápido.
- Probar en ambientes clonados de Producción.
- Todos deben ver que está sucediendo.
- Utilizar despliegues automáticos en todos los ambientes.

Tiene como beneficio fundamental la reducción de riesgos de integración y de errores al realizar automatizaciones para el proceso de actualización de código.

Una de las herramientas más utilizadas en este momento para la realización de Integración Continua es Git.

Git

Es un software para el control de versiones diseñado por Linus Torvalds, el mismo se compone por organizaciones las cuales poseen repositorios, cada uno de estos últimos cuenta con una parte remota y una parte local.

En los repositorios se encuentran los archivos con el código fuente desarrollado.

El desarrollador trabaja sobre su repositorio local y cuando termina su trabajo sube al repositorio remoto las actualizaciones.

Cuando otro desarrollador quiere subir su código al repositorio remoto le avisa que hay cambios nuevos y debe actualizar su repositorio local, si los dos desarrolladores modificaron el mismo archivo se produce un conflicto que debe ser solucionado para continuar.

A esto se le suma que cada repositorio puede usar ramas, que son clones del contenido del repositorio, para desarrollar una funcionalidad nueva sin alterar el contenido de la rama original, y también permite la operabilidad con versiones que congelan el contenido de una rama en el momento de su creación.

Conociendo esto nos adentramos en los llamados flows que son operatorias de trabajo para el desarrollo, el más conocido y utilizado es git-flow.

Git-flow: Es recomendado para ciclos de trabajo ya estipulados, posee dos ramas bien definidas y eternas, master y develop, luego cada rama que se crea parte desde alguna de estas dos dependiendo la necesidad.

Son utilizadas para funcionalidades nuevas (features), solución de errores (hotfix) y versiones con cambios varios (release) para luego integrarlas a master o develop.

La rama master es la responsable de llevar su contenido al ambiente de Producción.

Comandos básicos:

git init # Creación de un repositorio de forma local, se ejecuta desde la carpeta donde esta ubicado el proyecto.

git clone # Descarga un repositorio ya creado en el servidor local.

git add # Agrega archivos al repositorio.

git commit -m "mensaje" # Guarda las modificaciones con un mensaje descriptivo.

git branch # Nos muestra un listado de las ramas que posee el repositorio.

git checkout -b "nombre de la rama" # Crea una rama con el nombre especificado, copiando el contenido de la carpeta donde se encuentra ubicado.

git push # Luego del git commit, sube los cambios generados al repositorio remoto.

git pull # Actualiza el repositorio local con el contenido del repositorio remoto.

En el ejemplo se realizará un cambio en un repositorio git, con el SAAS GitHub.

Entrega Continua

Esta parte del proceso de actualización es la encargada de tomar el entregable generado en IC (Integración Continua) y de forma automática que se implemente en el ambiente destino.

En detalle el proceso completo de actualización se llama CICD (Continuous Integration Continuous Delivery) se efectúa a través de una serie de pasos que se agrupan en lo que se llama Pipeline (tubería)

Un pipeline contiene todos los pasos que se necesitan para generar desde el código fuente una aplicación ya disponible para su uso.

Ventajas

- Cualquiera puede disparar la actualización.
- Automatización total del proceso.
- El desarrollador solo se ocupa de mejorar la aplicación.
- Solución rápida de errores.

Desventajas

- La cantidad de Microservicios puede afectar la organización de la aplicación.
- Instalar de forma rápida puede generar que un error se propague más rápido.

Un pipeline posee pasos como se detallan a continuación:

1. Descargar el código, generalmente hace un clone del repositorio necesitado.
2. Reconocer en qué tecnología está desarrollado el código fuente, para esto busca qué archivos se acaban de descargar. Si es java debería encontrar un pom.xml, si es netcore debería encontrar un archivo .sln , si es python un requirements.txt , etc.

3. Compilar el código fuente descargado con la herramienta definida dependiente de la tecnología.
4. Realiza pruebas unitarias de forma automática.
5. Verifica la calidad del código con pruebas de cobertura.
6. Buscar vulnerabilidades con análisis de seguridad.
7. Genera la imagen docker.
8. Ejecutar es imagen para que se cree el contenedor (docker run).

Todos estos pasos se deben ejecutar de forma automática, evitando intervenciones manuales. Los tiempos de despliegue deben ser cortos.

Jenkins X

Para brindar una aproximación al concepto de referencia, vamos a utilizar una aplicación llamada Jenkins X, la cual es, dentro del mercado una novedad que parte de Jenkins como una versión ligera y fácil de instalar.

Con esta aplicación podemos generar un cluster de kubernetes para cualquier proveedor como Microsoft, Google o Amazon, generar aplicaciones java y administrar nuestros despliegues de forma ágil y fácil.

Dentro de sus funcionalidades se encuentra la de la creación y organización de jobs los cuales ejecutan pipelines que poseen el paso a paso para la integración y ejecución continua en el

desarrollo de software. Tiene su interfaz gráfica como su cliente para línea de comandos. En este trabajo vamos a realizar un ejemplo con el intérprete para línea de comandos.

Comandos básicos:

<i>jx create cluster</i>	<i>#Crear un cluster Kubernetes de forma automática.</i>
<i>jx import</i>	<i>#Importa un proyecto local nuevo.</i>
<i>jx create micro</i>	<i>#Crear un microservicio nuevo.</i>
<i>jx get applications</i>	<i># Muestra un listado de aplicaciones disponibles.</i>
<i>jx promote</i>	<i># Promueve una aplicación al ambiente requerido.</i>
<i>jx start pipeline</i>	<i># Inicia un job</i>
<i>jx get activity</i>	<i># Muestra en qué estado se encuentra un job</i>

Capítulo 3

Ejemplo de Integración y Entrega Continua

El ejemplo será demostrado en la “nube” de Google, GCP (Google Cloud Platform).

- 1) Creación del proyecto en GCP y acceso a la consola virtual.

Para usar GCP se requiere dar de alta una cuenta de Google. (Todo este proceso fue realizado utilizando los recursos gratis pero podrían incurrir en cobros por parte de GCP)

Todos las funcionalidades a utilizar en GCP deben pertenecer a un Proyecto, que es la entidad organizativa de la plataforma, en ellos se encuentran los permisos, metadatos y configuraciones que las aplicaciones necesitan para ejecutarse.

Las aplicaciones dentro de un Proyecto pueden comunicarse como si estuviesen en la misma red pero no pueden utilizarse desde otro Proyecto, a no ser que cuenten con una configuración especial.

GCP cuenta con una terminal de linux que se llama CloudShell desde donde vamos a ejecutar los comandos. Esta herramienta es una interfaz visual que incluye un programa que se llama “gcloud”, el cual sirve para ejecutar operaciones en la Plataforma.

CloudShell GCP

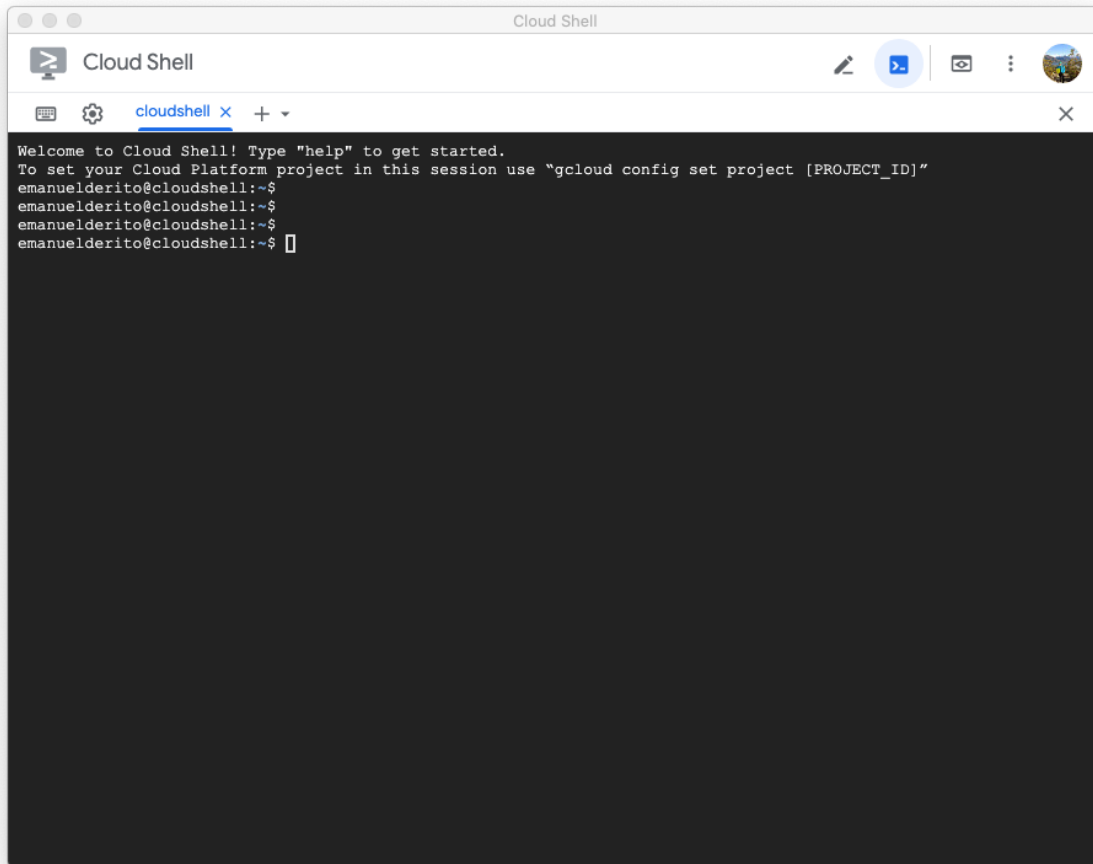


Figura 5. Consola CloudShell CGP. Fuente Propia 2020

CloudShell proporciona en el navegador: Una instancia de máquina virtual, un editor de código, 5gb de almacenamiento y compatibilidad de lenguajes (java, nodejs, go, python,etc) entre varias funcionalidades más.

La url de CloudShell es: <https://ssh.cloud.google.com/cloudshell/editor>

Lo primero a ejecutar es la creación de un Proyecto nuevo

```
emanuelderito@cloudshell:~$ gcloud projects create tesis-271417 --name= "tesis-271417"
```

Y la activación de ese Proyecto como predeterminado.

```
emanuelderito@cloudshell:~$ gcloud config set project tesis-27141
```

2) Instalación de Jenkins X

Descargamos la aplicación jx desde internet, precisamente desde el repositorio Github, con el siguiente comando:

```
emanuelderito@cloudshell:~(tesis-271417)$ curl -L "https://github.com/jenkins-x/jx/releases/download/$(curl --silent "https://github.com/jenkins-x/jx/releases/latest" | sed 's#.*/tag/^(.*)\'.*/I#')/jx-linux-amd64.tar.gz" | tar xzv "jx"
```

Luego movemos la aplicación al directorio /usr/local/bin para su ejecución.

```
emanuelderito@cloudshell:~(tesis-271417)$ sudo mv jx /usr/local/bin
```

El primer comando a ejecutar con la aplicación jx es la creación del cluster kubernetes. Se le pasa como parámetro gke para que la aplicación sepa que el cluster se va a crear en Google Cloud Platform, más precisamente en la aplicación Google Kubernetes Engine.

Dentro de este instalador tenemos que seleccionar, ingresar y aceptar los siguientes parámetros:

- Zona de instalación del cluster.
- Usuario Github.
- Dominio a utilizar para nuestras aplicaciones.

- Creación de tokens automática para Github.

La aplicación creará automáticamente dos ambientes: Staging y Production. También creará una docker registry donde guardará las imágenes, un repositorio de librerías nexus y un servidor web chartmuseum.

Es importante guardar el password de Admin que la aplicación va a informar.

```
emanuelderito@cloudshell:~ (tesis-271417)$ jx create cluster gke --skip-login
```

--El siguiente texto se encuentra resumido para su correcta lectura--

Using the only Google Cloud Project tesis-271417 to create the cluster

Updated property [core/project].

? No cluster name provided so using a generated one: hornetsand

? Defaulting to cluster type: Zonal

? Google Cloud Zone: southamerica-east1-a

Creating cluster hornetsand in southamerica-east1-a...

.....done.

Created [https://container.googleapis.com/v1/projects/tesis-271417/zones/southamerica-east1-a/clusters/hornetsand].

? Select Jenkins installation type: Static Jenkins Server and Jenkinsfiles

Setting the docker registry organisation to tesis-271417 in the TeamSettings

Git configured for user: emalpb and email emanuelderito@gmail.com

? Domain [? for help] (34.95.207.196.nip.io)

Set up a Git username and API token to be able to perform CI/CD

Creating a local Git user for github.com server

? github.com username: emalpb

? Select the organization where you want to create the environment repository: emalpb

Using Git provider github.com at https://github.com

? Using Git user name: emalpb

? Using organisation: emalpb

Creating repository emalpb/environment-hornetsand-staging

Creating Git repository emalpb/environment-hornetsand-staging

Pushed Git repository to https://github.com/emalpb/environment-hornetsand-staging

Jenkins X installation completed successfully

NOTE: Your admin password is: 9lj3c#ytQABl76K%zJHT

Your Kubernetes context is now set to the namespace: jx

To switch back to your original namespace use: jx namespace default

Or to use this context/namespace in just one terminal use: jx shell

For help on switching contexts see: <https://jenkins-x.io/developing/kube-context/>

To import existing projects into Jenkins X: jx import

To create a new Spring Boot microservice: jx create spring -d web -d actuator

To create a new microservice from a quickstart: jx create quickstart

Fetching cluster endpoint and auth data.

kubeconfig entry generated for hornetsand.

Context "gke_tesis-271417_southamerica-east1-a_hornetsand" modified.

NAME	HOSTS	ADDRESS	PORTS	AGE
chartmuseum	chartmuseum.jx.34.95.207.196.nip.io	34.95.207.196	80	2m28s
docker-registry	docker-registry.jx.34.95.207.196.nip.io	34.95.207.196	80	2m28s
jenkins	jenkins.jx.34.95.207.196.nip.io	34.95.207.196	80	2m27s
nexus	nexus.jx.34.95.207.196.nip.io	34.95.207.196	80	2m28s

Como ven, se informan las aplicaciones instaladas y las urls para su acceso ya disponibles en internet. Dichas aplicaciones están ejecutándose en contenedores docker dentro de kubernetes en GKE.

Finalizamos la instalación de Jenkins X.

3) Desplegar Aplicación.

Para el ejemplo contamos con un repositorio en Github <https://github.com/emalpb/tesis-derito.git>, es una aplicación que muestra un gráfico del valor del Bitcoin con respecto de la moneda de un país a elección.

La aplicación es una copia (fork) de <https://github.com/MarkFChavez/bitcoin-price-index>

Realizada en react – javascript y compilada con nodejs.

Primeramente se descarga el repositorio con un git clone.

```
emanuelderito@cloudshell:~ (tesis-271417)$ git clone https://github.com/emalpb/tesis-derito.git
```

Cloning into 'tesis-derito'...

```
remote: Enumerating objects: 129, done.
remote: Counting objects: 100% (129/129), done.
remote: Compressing objects: 100% (81/81), done.
remote: Total 129 (delta 52), reused 122 (delta 45), pack-reused 0
Receiving objects: 100% (129/129), 215.00 KiB | 0 bytes/s, done.
Resolving deltas: 100% (52/52), done.
```

Se accede al directorio de la aplicación.

```
emanuelderito@cloudshell:~ (tesis-271417)$ cd tesis-derito/
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$
```

Se ejecuta el comando “*jx import*”, él mismo va a detectar que la aplicación es javascript, crear y ejecutar un pipeline con el paso a paso necesario para generar la imagen docker y desplegar la aplicación en un contenedor dentro de kubernetes.

Para ello va a solicitar las credenciales del usuario Github.

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx import

WARNING: No username defined for the current Git server!
? Do you wish to use emailpb as the Git user name: Yes
performing pack detection in folder /home/emanuelderito/tesis-derito
--> Draft detected JSON (94.856513%)
--> Could not find a pack for JSON. Trying to find the next likely language match...
--> Draft detected YAML (1.884994%)
--> Could not find a pack for YAML. Trying to find the next likely language match...
--> Draft detected JavaScript (1.469259%)
selected pack: /home/emanuelderito/.jx/draft/packs/github.com/jenkins-x-buildpacks/jenkins-x-kubernetes/packs/javascript
```

*WARNING: existing Dockerfile, Jenkinsfile and charts folder found so skipping 'draft create' step
replacing placeholders in directory /home/emanuelderito/tesis-derito*

app name: tesis-derito, git server: github.com, org: emalpb, Docker registry org: tesis-271417

skipping directory "/home/emanuelderito/tesis-derito/.git"

Username for 'https://github.com': emalpb

Password for 'https://emalpb@github.com':

Created Jenkins Project: <http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/tesis-derito/>

Watch pipeline activity via: `jx get activity -f tesis-derito -w`

Browse the pipeline log via: `jx get build logs emalpb/tesis-derito/master`

You can list the pipelines via: `jx get pipelines`

When the pipeline is complete: `jx get applications`

For more help on available commands see: <https://jenkins-x.io/developing/browsing/>

Note that your first pipeline may take a few minutes to start while the necessary images get downloaded!

Triggered Jenkins job: <http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/tesis-derito/>

Con el comando “`jx get activity`” nos muestra en qué estado se encuentra la ejecución del pipeline, si se le agrega “`-w`” va a continuar mostrando la ejecución hasta que termine de correr.

emanuelderito@cloudshell:~/tesis-derito (tesis-271417)\$ jx get activity

<i>STEP</i>	<i>STARTED</i>	<i>AGO</i>	<i>DURATION</i>	<i>STATUS</i>
<i>emalpb/environment-beetlebold-production/master #1</i>	<i>7m38s</i>	<i>3m48s</i>	<i>Succeeded</i>	
<i>Checkout Source</i>	<i>5m43s</i>	<i>11s</i>	<i>Succeeded</i>	
<i>Validate Environment</i>	<i>5m32s</i>	<i>50s</i>	<i>Succeeded</i>	
<i>Update Environment</i>	<i>4m41s</i>	<i>51s</i>	<i>Succeeded</i>	
<i>emalpb/environment-beetlebold-staging/master #1</i>	<i>7m45s</i>	<i>3m25s</i>	<i>Succeeded</i>	
<i>Checkout Source</i>	<i>6m31s</i>	<i>15s</i>	<i>Succeeded</i>	

```

Validate Environment          6m16s   57s Succeeded
Update Environment           5m19s   59s Succeeded
emalpb/tesis-derito/master #1

```

```

emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx get activity

```

```

STEP                               STARTED AGO DURATION STATUS
emalpb/environment-beetlebold-production/master #1   9m17s   3m48s Succeeded
Checkout Source                               7m22s   11s Succeeded
Validate Environment                           7m11s   50s Succeeded
Update Environment                           6m20s   51s Succeeded
emalpb/environment-beetlebold-staging/master #1     9m24s   3m25s Succeeded
Checkout Source                               8m10s   15s Succeeded
Validate Environment                           7m55s   57s Succeeded
Update Environment                           6m58s   59s Succeeded
emalpb/tesis-derito/master #1                   1m58s    Running
Checkout Source                               1m7s    7s Succeeded
CI Build and push snapshot                     1m0s   NotExecuted
Build Release                                 59s    Pending

```

El siguiente comando nos informa los pipelines creados por la herramienta.

```

emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx get pipelines

```

```

Name                               URL                               LAST_BUILD STATUS
DURATION
emalpb/environment-beetlebold-production/master http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/environment-beetlebold-
production/job/master/ #1   SUCCESS 228.059µs

```

emalpb/environment-beetlebold-staging/master <http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/environment-beetlebold-staging/job/master/> #1 SUCCESS 205.358µs

emalpb/tesis-derito/master <http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/tesis-derito/job/master/>
#1 SUCCESS 120.346µs

Para ver la ejecución de un pipeline específico el comando a utilizar es:

emanuelderito@cloudshell:~/tesis-derito (tesis-271417)\$ jx get act emalpb/tesis-derito/master #3

```

emalpb/tesis-derito/master #3          5m58s    Running Version: 0.0.13
Checkout Source                        5m36s    7s Succeeded
CI Build and push snapshot             5m29s    NotExecuted
Build Release                          5m29s    2m25s Succeeded
Promote to Environments                3m4s     Pending
Promote: staging                       2m37s    Running
PullRequest                           2m37s    2m8s Succeeded PullRequest: https://github.com/emalpb/environment-
beetlebold-staging/pull/1 Merge SHA: bd9cf3409f6473d5c1d0fcf46910e3a24f9d2656
Update                                29s      Running Status: Pending at:
http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/environment-beetlebold-staging/job/master/2/display/redirect

```

```

emalpb/tesis-derito/master #3          8m22s    7m58s Succeeded Version: 0.0.13
Checkout Source                        8m0s     7s Succeeded
CI Build and push snapshot             7m53s    NotExecuted
Build Release                          7m53s    2m25s Succeeded
Promote to Environments                5m28s    5m4s Succeeded
Promote: staging                       5m1s     4m33s Succeeded
PullRequest                           5m1s     2m8s Succeeded PullRequest: https://github.com/emalpb/environment-
beetlebold-staging/pull/1 Merge SHA: bd9cf3409f6473d5c1d0fcf46910e3a24f9d2656

```

```

Update                2m53s  2m25s Succeeded Status: Success at:
http://jenkins.jx.34.95.207.196.nip.io/job/emaplb/job/environment-beetlebold-staging/job/master/2/display/redirect

Promoted              2m53s  2m25s Succeeded Application is at: http://tesis-derito.jx-
staging.34.95.207.196.nip.io

Clean up              24s    0s Succeeded

```

Una vez finalizados todos los pasos. Verificar que la aplicación se encuentra desplegada y lista para utilizar con el comando:

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx get applications
```

APPLICATION	STAGING	PODS	URL
tesis-derito	0.0.1	1/1	http://tesis-derito.jx-staging.34.95.207.196.nip.io

Recordemos que pods es un conjunto de contenedores, en este caso el pod va a contener sólo una instancia, lo que indica es que 1 pod está disponible y levantó de forma exitosa. Si se visualiza 0/1 en la columna pods, la aplicación falló en su inicio y se debe acceder al pod a través de kubernetes para verificar cuál fue el error.

Aplicación de Ejemplo

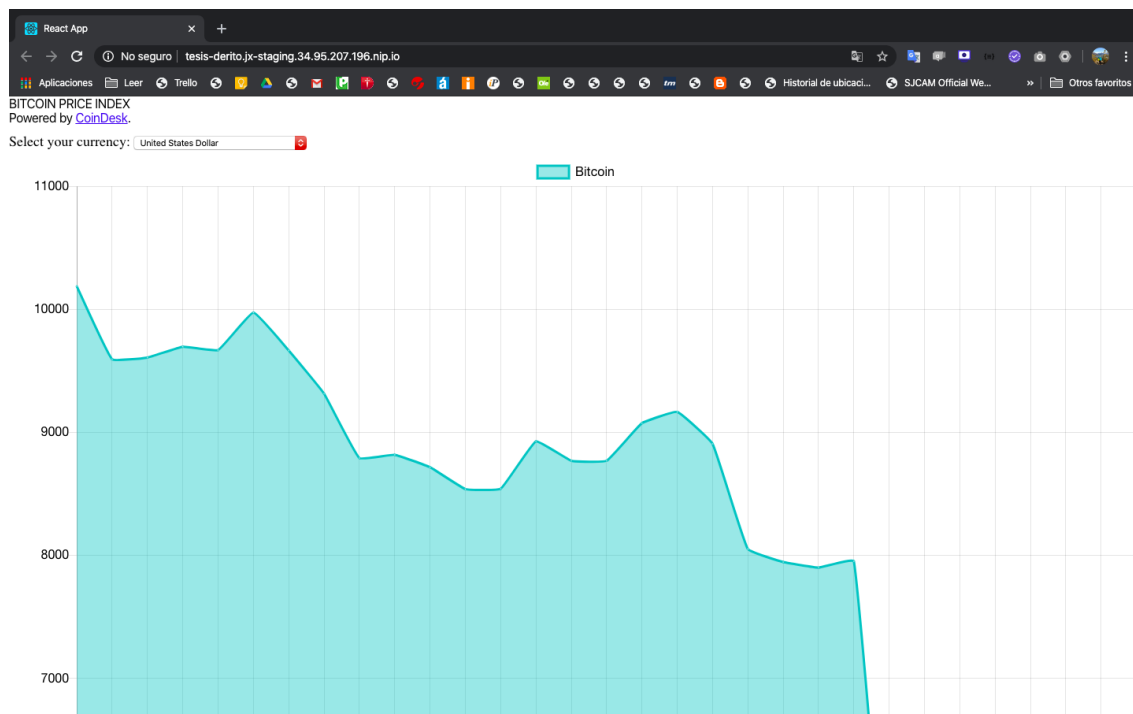


Figura 6. Aplicación funcionando. Fuente propia

4) Realizar un cambio en la Aplicación.

En el repositorio tesis-derito, ya se encuentra creada la rama “cambio-1”, lo primero que se debe hacer es descargar esa rama del repositorio remoto al repositorio local, con el comando:

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ git checkout cambio-1
```

Luego actualizar el contenido:

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ git pull
```

Volver a la rama master:

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ git checkout master
```

Y realizar una operación automática que se llama *merge*, que unifica el contenido de la rama cambio-1 a master. El parámetro -m es para grabar un mensaje en la operación.

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ git merge cambio-1 -m "cambio azul y rojo"
```

El cambio que está impactando en master, es la modificación de colores del gráfico por azul y rojo.

Una vez realizado esto se debe subir al repositorio remoto.

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ git push origin master
```

Se vuelve a ejecutar el pipeline de actualización de aplicación.

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx start pipeline emalpb/tesis-derito/master
```

Started build of emalpb/tesis-derito/master at <http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/tesis-derito/job/master/7/>

view the log at: <http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/tesis-derito/job/master/7/console>

Y se verifica la ejecución del mismo.

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx get act emalpb/tesis-derito/master
```

```
emalpb/tesis-derito/master #7          10m57s   8m5s Succeeded Version: 0.0.2
Checkout Source                        10m35s    8s Succeeded
CI Build and push snapshot             10m26s    NotExecuted
Build Release                          10m26s    2m0s Succeeded
```

```

Promote to Environments          8m26s   5m33s Succeeded
Promote: staging                 8m0s    4m53s Succeeded
PullRequest                     8m0s    2m9s Succeeded PullRequest: https://github.com/emalpb/environment-
beetlebold-staging/pull/3 Merge SHA: 83d8a9dd517e8461215938cfd422e6e1cac803bb
Update                          5m51s   2m44s Succeeded Status: Success at:
http://jenkins.jx.34.95.207.196.nip.io/job/emalpb/job/environment-beetlebold-staging/job/master/4/display/redirect
Promoted                       5m51s   2m44s Succeeded Application is at: http://tesis-derito.jx-
staging.34.95.207.196.nip.io
Clean up                       2m52s    0s Succeeded

```

Volvemos a ver las aplicaciones en ejecución y encontramos disponible nuestra web en versión 0.0.0.

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx get applications
```

APPLICATION	STAGING	PODS	URL
tesis-derito	0.0.2	1/1	http://tesis-derito.jx-staging.34.95.207.196.nip.io

Accedemos a la url:

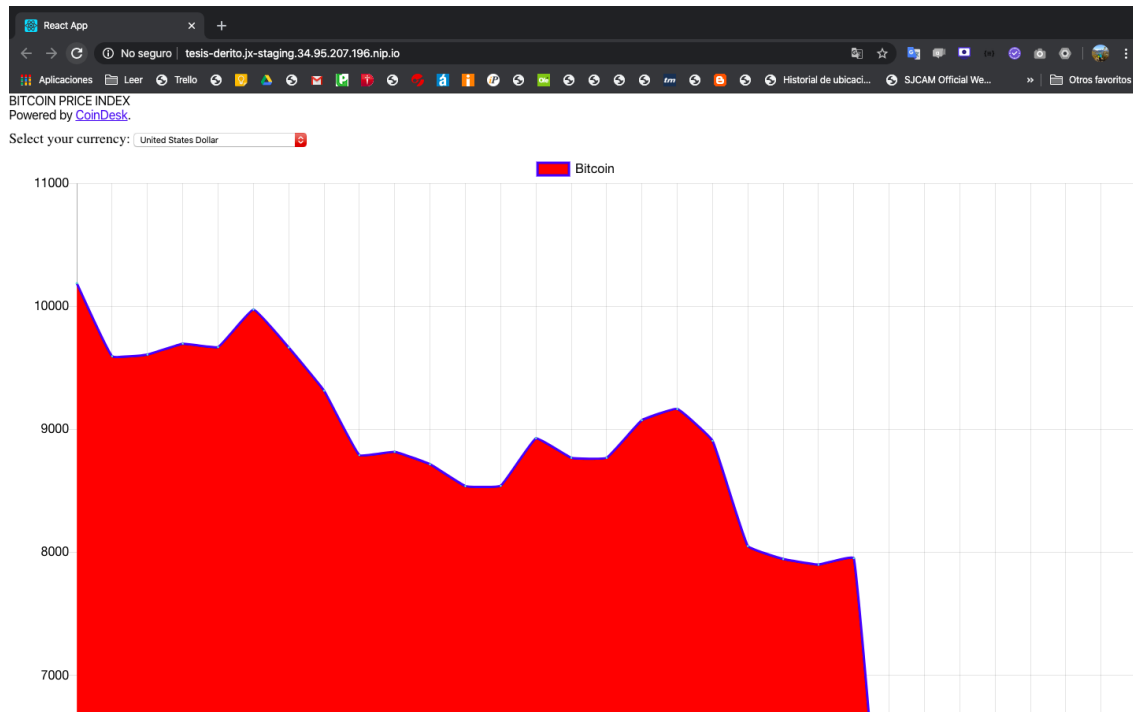


Figura 7. Cambio aplicativo desde la ejecución de un pipeline. Fuente propia

Recordemos que este ambiente es Staging y es un ambiente de pruebas similar a Homologación.

Para implementar nuestra aplicación en el ambiente Productivo el comando es:

```
emanuelderito@cloudshell:~/tesis-derito (tesis-271417)$ jx promote --app tesis-derito --version 0.0.2 --env production
```

Este comando va a ejecutar el pipeline que se encarga de crear o actualizar la aplicación en el ambiente productivo. Como parámetro necesita el número de versión.

Investigación

El objetivo era proponer un conjunto de herramientas y técnicas para dar respuesta de forma rápida y ágil a los cambios en los requerimientos.

Ese conjunto compuesto por:

- Scrumban para la gestión de las tareas del equipo y el manejo de los requerimientos.
- Arquitectura de Microservicios para la independencia entre funcionalidades que juntas generan una aplicación.
- Integración Continua para el desarrollo del software.
- Despliegue Continuo para la puesta en funcionamiento del software generado.

Se encontró en Scrumban una forma óptima de ordenar y administrar las tareas a realizar dentro del sprint (iteración de tiempo limitado), con roles definidos como Dueño del Producto (PO), Scrum-Master y el Equipo.

Donde el Scrum-Master se encarga de armar y ser anfitrión de las 4 ceremonias fijas (planning, daily, review y retrosprective).

El PO prioriza los requerimientos a afrontar en el sprint y valida una vez finalizado que se hayan cumplido.

El equipo toma esos requerimientos y los plasma en tareas en el tablero Kanban, para luego cambiarlos de estado hasta su finalización.

En Arq. de Microservicios una aplicación está generada por un conjunto de mini aplicaciones independientes, debe estar correspondida por un repositorio de código fuente (git). Los desarrolladores crean su código en su pc local y luego lo suben al repositorio. El objetivo es que el código fuente sea colaborativo, para que todo el equipo pueda acceder y entonces mejorar o solucionar partes del código. La interfaz gráfica es también una miniaplicación independiente parte de esta Arquitectura, la cual va a comunicarse con la parte operativa donde cada funcionalidad es un microservicio.

Cada aplicación – repositorio se convierte en una imagen que luego se debe ejecutar en un contenedor, recomendamos docker que es open-source.

Cada imagen docker se guarda en una biblioteca y tiene versiones evolutivas de sí misma. Ejemplo: app1:1.0, app1:1.1, app1:2.0 si una actualización tiene un error y daña la funcionalidad (microservicio), la idea es volver a la versión anterior cuanto antes para no afectar el funcionamiento en general.

El proceso encargado de realizar la implementación del código fuente debe ser automático, para eso existen gestores de automatización, programas que se encargan de realizar CICD de forma automática.

Recordemos CI (Integración Continua) es el proceso en el cual se toma el código fuente de git, se compila dando por resultado la aplicación lista para su utilización (binarios), se realizan pruebas sobre el mismo y deja disponible como entregable una imagen docker en la biblioteca.

Luego la otra parte del proceso CD (Despliegue Continuo) es la que toma esa imagen desde la biblioteca y ejecuta un proceso el cual transforma la imagen docker en un contenedor.

El contenedor expone el microservicio para que pueda ser utilizado en la red. Existen varios gestores de automatización, pero seleccionamos Jenkins X para el ejemplo. El mismo contiene Jobs (Pipelines) dentro de ellos se encuentra el paso a paso para realizar la operación de implementación (CICD) de forma automática.

Pasos del Pipeline:

Paso 1: Tomar el código fuente del repositorio

Paso 2: Compilar el código fuente.

Paso 3: Realizar pruebas unitarias.

Paso 4: Realizar pruebas de cobertura.

Paso 5: Realizar pruebas de seguridad.

Paso 6: Crear imagen docker.

Paso 7: Ejecutar docker run de esa imagen, para dejar disponible la aplicación.

Recomendaciones al definir estrategias de CICD:

1. Definir un git-flow y utilizar tags.
2. Definir un estándar de nombres antes de crear los repositorios.
3. Ejecutar test en forma paralela.
4. No trabajar por tecnología, buscar estadios estándares.
5. Utilizar en lo posible técnicas de despliegue avanzado.
6. Desarrollar pensando en la observabilidad (monitoreo).

7. Mínimamente 3 ambientes (Desarrollo, Homologación y Producción)

Capítulo 4

Caso de Éxito Banco Galicia

Introducción

En este capítulo se desarrolla como Banco Galicia decide cambiar de paradigma organizacional y mejorar la entrega de valor a sus clientes considerando la tecnología como un pilar estratégico. Este proceso se llama transformación digital [TD] y uno de los puntos fuertes a tratar, debido al consumo de internet desde dispositivos, fue el desarrollo de aplicaciones.

¿Por qué Transformación Digital?

Si miramos las tendencias en el mercado actual vemos cómo la tecnología es protagonista y omnipresente ya que un tercio de las 100 empresas más caras son tecnológicas.

Se busca cambiar, fundamentalmente, la forma en que el cliente se relaciona con el Banco por medio de tres premisas claves: la experiencia, el crecimiento y la eficiencia.

Experiencia: Nos referimos a cuál es la sensación que queda en el cliente luego de haber realizado cualquier operación con el Banco, ya que como sabemos, las nuevas generaciones son nativamente digitales, es decir que nacieron acostumbrados a relacionarse a través de dispositivos y a recibir respuestas automáticas y rápidas para cualquier tipo de operación. En ese sentido el Banco hasta hoy generaba una disrupción en esta manera de relacionarse ya que requería que la

persona llame por teléfono, se acerque a una sucursal, espere a fin de mes para ver los consumos de la tarjeta, etc. Si sumamos esto a que la industria bancaria está muy regulada e históricamente contaba con una “burocratización” muy grande, se ha logrado que las personas sientan un rechazo hacia los bancos y toda su gestión, lo que disminuye el uso de los servicios y reduce las ganancias.

Teniendo en cuenta lo anterior, la tecnología surge como una alternativa ideal para poder hacer todas las operaciones e interacciones más simples, menos burocráticas y poder generar una ventaja diferencial con la competencia.

Crecimiento: Poder llegar a todos los rincones del país, incluso en las cuales no se cuenta con una sucursal. Al hacer las transacciones más rápidas, fáciles y menos burocráticas, desde cualquier lugar y a cualquier hora se apunta a aumentar la cantidad de operaciones, lo que se traduce como mayor cantidad de comisiones.

Eficiencia: Se busca aumentar capacidades para realizar operaciones por canales automáticos sin la necesidad de tener personas e infraestructura localizada cerca de los clientes.

El Banco tiene costos fijos de personal e infraestructura, ya que cuenta con sucursales y cajeros automáticos en todo el país.

A diferencia de las empresas fintech (Finanzas y Tecnología), se llaman así debido a que fueron creadas con la última tecnología existente para ofrecer productos financieros, ejemplo: MercadoLibre, Naranja X, Todo Pago, etc.

Se busca aprovechar que esta generación poblacional (Y) es mucho más digital que las anteriores, lo que se interpreta como un beneficio y la posibilidad de ahorrar costos de infraestructura, pero también es totalmente necesario dar respuestas inmediatas y de calidad.

Si analizamos esta tendencia en Argentina podemos ver que en los últimos años Mercado Libre ha dejado de ser una plataforma de venta de productos y se ha sumado a la competencia en el mercado financiero, con productos como fondos de inversión, cuentas virtuales, medios de pago con QR, etc.

También tenemos el desembarco de bancos digitales como el BruBank y WiloBank que, si bien todavía no han llegado a capturar una parte significativa del mercado, son compañías que ya nacieron digitalmente, se adaptan mucho más rápido a los cambios y pueden ofrecer al cliente servicios superiores en calidad e información en tiempo real a menor costo.

La necesidad fue responder a estos nuevos competidores buscando una solución digital, por eso enfocaron la TD al desarrollo de aplicaciones de manera rápida y dinámica.

Anteriormente el Banco Galicia basaba la creación de aplicaciones en empresas tercerizadas [SF], los proyectos eran clásicos con MTC, las tareas eran definidas, los tiempos se encontraban comprometidos y el alcance era fijo. Se generaban monolíticos, lo que significa que toda la aplicación es un código único donde se encuentran todas las funcionalidades y a lo sumo se utilizaba una base de datos. Si el monolítico tenía un problema, toda la aplicación dejaba de funcionar.

La segregación de funciones estaba bien definida para la comunicación entre sectores se debía generar un ticket en una herramienta (Serena/Remedy).

El método de trabajo era el siguiente:

- El Manager del Proyecto generaba tickets para los sectores de Infraestructura, Seguridad Informática, Redes, etc. solicitando las configuraciones necesarias.
- La SF subía el código compilado al repositorio.

- Enviaba un ticket al sector de Implementaciones que instalaba el software en el ambiente indicado.
- El sector de Calidad a través de un ticket probaba ese software implementado y dependiendo del resultado, se continuaba al siguiente ambiente o en su defecto devolvía el ticket a la Software Factory.

La rigidez que existe en los requerimientos y en las etapas del proyecto producían que el cliente no esté a gusto con lo que recibe. El proyecto era afectado por los “tiempos de respuesta” de cada sector interviniente. Un ticket podía pasar más de una vez por un sector, lo que producía que las estadísticas generadas no sean del todo reales.

Todo este proceso fue reorganizado en la creación de aplicaciones “ágiles”.

A cada proyecto se le llama Journey (Viaje) el mismo consta de 12 personas: Product Owner, Navegador, DevOps, Arquitecto, 2 Desarrolladores FrontEnd, 2 Desarrolladores BackEnd, Scrum Master, Desarrollador QA, Diseñador UI, Diseñador UX.

Todos en el mismo lugar, trabajando en conjunto y con objetivos compartidos. Con respecto a la infraestructura de trabajo se dispuso de una herramienta PAAS (Platform as a Service) que se llama Openshift de la compañía Red Hat, la misma incorpora todas las tecnologías open source con las cuales se trabaja en la nube, como Docker y Kubernetes. A su vez el repositorio se montó sobre un git y la aplicación para CICD fue Jenkins.

El viaje comienza con 4 semanas de recopilación de información y técnicas de elicitación de requerimientos, los diseñadores junto al PO y Navegador se encargan de realizar entrevistas y

obtener información para luego definir qué puntos de dolor se van a atacar (problemas a solucionar).

Las 2 semanas que siguen son para definir qué tipo de Prototipo [MVP] se va a realizar y presentarlo para su aprobación a la gerencia. Luego de autorizado el journey cuenta con 8 semanas para desarrollar el MVP. En esta fase se generan los requerimientos (historias de usuario). El PO se encarga de administrar esas historias y de seleccionar cuáles se van a realizar en el sprint, que tiene por duración una semana. Cada desarrollador se encarga de su “historia de usuario” o tarea, las cuales para estar resueltas deben respetar “criterios de aceptación”, al cierre del sprint se hace una review y se muestra todo el trabajo realizado en ese sprint. Esto permite llevar la agilidad y el trabajo en equipo al máximo. En esta investigación se encontró que las 3 aplicaciones relevadas llegaron en término al lanzamiento del MVP. Luego del lanzamiento la labor diaria cambia a ir mejorando y agregando funcionalidad sobre la aplicación. El cliente tiene visibilidad sobre en qué etapa se encuentra la aplicación y aporta feedback constante, en cada sprint.

Cada instalación o actualización se realiza de la siguiente manera:

1. Los desarrolladores trabajan en su pc local sobre el código
2. Lo suben al repositorio git
3. Ejecutan un pipeline en DEV o Desarrollo el cual realiza los pasos de CICD de forma automática.
4. Se realizan pruebas de integración manuales.
5. Luego se ejecuta INT -> QAS -> PRD siguiendo el orden y con su respectivo pipeline.

6. El más importante es Producción o PRD, en el cual el desarrollo debe llegar totalmente validado para eso se solicita al PO que de su autorización.

Para cada operación a realizar en la aplicación se debe ejecutar un pipeline el cual deja un registro, con el que podemos tener visibilidad de instalaciones efectuadas y nos sirve para tener estadísticas. Entre los dos tipos de metodología encontramos que en MTC están muy segregadas las funciones, pero tienen el problema de los datos estadísticos reales a su vez que un error o cambio del usuario produce una demora, con este conjunto de herramientas el MVP tiene un tiempo estimado de puesta en producción de 3 meses, los datos estadísticos son reales ya que la contabilización se realiza por ejecución del pipeline.

El beneficio del uso en conjunto de MAS, Microservicios, DevOps y CICD produce eficiencia en tiempos, personal, estadísticas, etc.

Desarrollo Ágil

Desde la adopción de esta forma de trabajo las primeras aplicaciones generadas fueron:

Todas estas aplicaciones se crearon bajo metodología de trabajo ágil (Scrumban, Microservicios en Openshift, Devops y CICD) con entrega de MVP a las 14 semanas de iniciado el proyecto. Los nombres de las aplicaciones están ofuscados.

Venta Minorista:

Anteriormente el proceso de alta de un cliente requería que la persona se acercara a una sucursal y espere a ser atendido por un oficial, el cual lo guiaba para que complete varios formularios, presente unas fotos, lleve la fotocopia de su DNI y CUIL.

Una vez ingresado el trámite debía esperar entre 5 y 7 días para que su cuenta esté disponible. Todo este proceso fue reemplazado por una aplicación web que se puede utilizar desde el celular, pc o tablet.

Se le solicita al cliente que ingrese DNI y fecha de nacimiento, los otros datos necesarios los genera automáticamente utilizando microservicios propios o de terceros (ejemplo RENAPER o AFIP) , luego le pide una foto (selfie) para escanear datos biométricos, toma la dirección automáticamente utilizando Google Maps y por último solicita una foto del frente y contrafrente del DNI.

Eficiencia:

- Tiempo de operación (TO, demora entre que la operación ingresa y la misma es finalizada) reducido de 5 días a 24 hs (80% de reducción)
- El ahorro en costo operativo equivale a la suma anual de 4 millones de pesos.
- Con una liberación de 33 FTEs (Full-Time Equivalent, una forma de medir la cantidad de empleados a tiempo completo que se necesitan para llevar a cabo dicho trabajo) en Sucursales y 25 FTEs en Originación Minorista

Venta de Cheques:

El cliente debía ir a la ventanilla de la sucursal con todo sus cheques, presentarlos al empleado, luego llenar una planilla con todos los datos de los cheques y firmarla. El empleado debía escanear cada uno de los cheques y enviarlos a la casa matriz por correo interno.

El MVP desarrollado es una aplicación web para clientes y empresas donde solo se ingresa el numero de cheque y los datos del mismo. La aplicación se encarga de validar que el cheque pueda ser comprado, realiza varios análisis automáticos que el empleado anteriormente hacía de forma manual, para finalizar devuelve la planilla completa con todos los cheques a aceptar y los que deben ser rechazados.

Eficiencia:

- TO se redujo de 16 minutos para 5 cheques a 3 minutos para 50 cheques. (5200% mejor performance)
- La venta de un cheque paso de un costo operativo de 3,2 minutos a 0,06 minutos.
- Con una liberación de 47 FTEs en sucursales.
- Mejora de nivel de satisfacción de 2,6 a 4,6 sobre 5 puntos (77% de satisfacción mejorada)

Venta a Empresas:

Esta operación era realizada en su totalidad de forma manual. El objetivo es vincular una empresa para el uso de líneas de crédito del Banco Galicia. En la sucursal el empleado le solicitaba al cliente que complete una serie de formularios y adjunte toda la documentación referida a la

empresa, entre ellos el estatuto con la firma de todos los involucrados. Con este ingreso el trámite recién comenzaba y la empresa debía aguardar mínimo 5 días para saber si era aceptado o no.

Este proceso no garantiza el alta de la empresa.

La aplicación creada como MVP brinda una simulación de líneas de crédito ofertadas por el Banco Galicia, para que el cliente verifique su interés. Una vez aceptada esta oferta se le solicita que llene un formulario online y que suba los documentos escaneados.

Con Inteligencia artificial se leen las letras del estatuto y se genera una solicitud al área de Bastanteo para que finalice el proceso.

Solo con el MVP se automatizó y organizó el 75% de la operación.

Eficiencia:

- TO reducido de 5 días a 48 hs. (60% de reducción)
- Se ahorró en papel la suma anual de 2 millones de pesos
- Con una liberación de 5 FTEs y una estimación de 30 FTEs en sucursales más 10 FTEs en Operaciones.

Tiempos de despliegue

La comparación se efectuó sobre seis aplicaciones, tres con la forma tradicional de desarrollo que a su vez son las más críticas del Banco y las mismas tres ágiles informadas en el texto anterior. Todas en estado productivo y dando servicio a clientes del Banco.

Tradicionales:

- HomeBanking: Servicios bancarios que se pueden acceder desde internet, a través de una aplicación web.
- Portal para empresas: Web con servicios bancarios para empresas.
- Portal Interno para empleados: Aplicación web en la red interna del Banco solamente para empleados, se publica contenido informativo.

Muestra estadística recolectada en el mes agosto 2019

Se procedió a recolectar información sobre implementaciones en el ambiente de Producción sobre cada una de estas aplicaciones:

Tickets de actualización de software: HomeBanking – Empresas – PortalInterno (Entrega Tradicional)

- Incluyen 4 cambios en 1 implementación.
- Usan un instalador compilado por el desarrollador fuera del proceso de forma manual.
- No tienen testeos automáticos, una vez finalizada la implementación las pruebas las realiza otro sector.

- Ejecuta la implementación de un servidor a la vez.
- Duración de ejecución total: 20 minutos.

Ejecución de Pipelines: Venta Minorista – Venta de Cheques – Venta Mayorista (Entrega Continua)

- Solo 1 cambio
- La ejecución contiene la compilación automática, prueba unitaria, prueba de cobertura de código, escaneo de vulnerabilidades y el despliegue en todos sus contenedores.
- El despliegue mantiene el contenedor con la versión vieja corriendo hasta que el contenedor con la versión nueva se encuentra operativo, lo que no produce interrupción de servicio.
- Duración de ejecución total: 4 minutos.

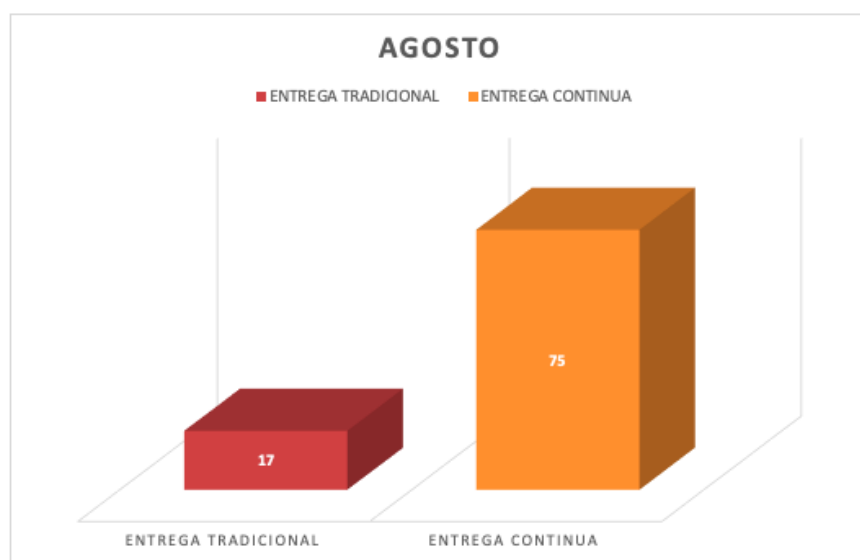


Figura 7. Diferencia entre tipos de implementaciones. Fuente propia

Se calculó que 17 implementaciones(tickets) con 4 cambios es lo mismo que 68 ejecuciones de pipelines.

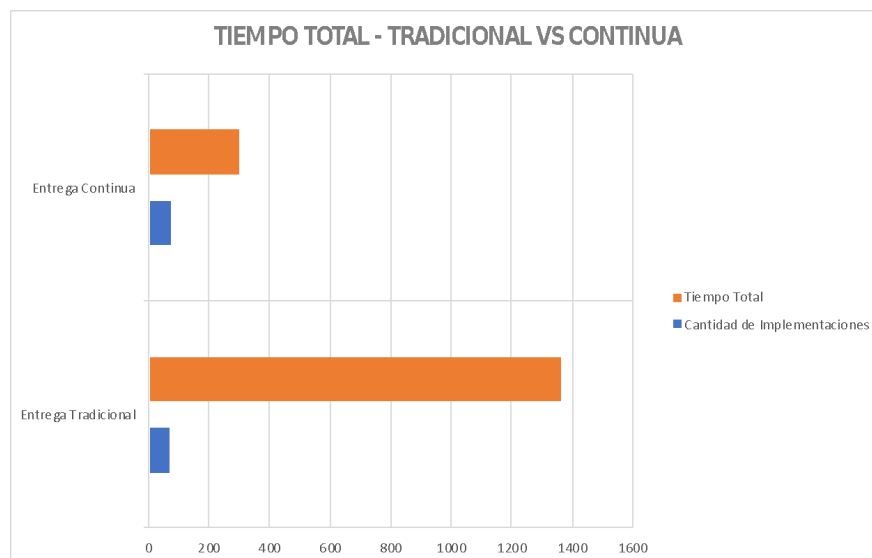


Figura 8. Velocidad entre distintas implementaciones. Fuente propia

Las implementaciones con Entrega Tradicional y sin testeos automáticos demoran un 350% más que la Entrega Continua.

Conclusiones

Pensar los procesos con un enfoque digital es un cambio fundamental que se debe dar en todas las áreas de una organización para que junto con la tecnología puedan crearse aplicaciones que generen valor al usuario y una diferencia competitiva para la empresa.

La unión de Scrumban, Microservicios y CICD en el desarrollo de aplicaciones brindan beneficios muy rápidamente:

- Disponer de un prototipo en un lapso de tiempo corto permite que el usuario brinde su opinión para saber hacia dónde dirigir los esfuerzos.
- Actualizar de forma automática permite más cambios en el desarrollo.
- La ejecución de pipelines que incluyen los pasos test unitarios, cobertura de código y análisis de seguridad es necesario para ahorrarnos tiempos de pruebas y garantizar calidad/seguridad.
- Al poder adjudicar valor y solucionar errores de forma rápida generamos una mejor experiencia al cliente final.
- La autorización del responsable (PO) es clave para tener control sobre qué llega a Producción.
- Fomentar siempre la búsqueda de automatizaciones que nos ahorran trabajo, como explica la cultura Devops.

En relación a lo expuesto, puede comprenderse cómo la TD generó un cambio de paradigma en la metodología de trabajo del desarrollo de aplicaciones, según el caso de éxito informado, en catorce semanas que contaron con tres aplicaciones nuevas funcionando en producción y facilitando la labor diaria de los empleados. Estas nuevas aplicaciones generaron ahorros en horas laborales de empleados, gastos básicos (papel, formularios, impresiones, etc.) y también optimización del tiempo utilizado en dar respuesta al cliente mejorando los índices de satisfacción en un 60%.

Líneas Futuras de Investigación

Se han descubierto temas de investigación con las que se puede mejorar y ampliar lo desarrollado en esta investigación:

Cuáles serían las ventajas y desventajas de reemplazar Arq. de Microservicios por otra arquitectura de software, con la intención de mejorar velocidad, dinámica y tolerancia a errores.

GitOps: Es un tema interesante el cual se basa en la creación de “ambientes como código”, consiste en describir lo necesario para el ambiente en un archivo, almacenarlo en un repositorio git y desde ese archivo generar en kubernetes él o los contenedores con la configuración necesaria.

Investigar sobre todos los tipos de prueba que pueden ser automatizados dentro del proceso de entrega continua. Es necesario para evitar implementar en el ambiente de producción código con errores.

Un tema interesante es Devops, el cual evoluciona de manera rápida y además se encuentra en la vanguardia, con el objetivo de normalizar, estandarizar y expandir la colaboración, automatización y el autoservicio.

Bibliografía Complementaria

CICD

Fowler, M., & Foemmel, M. (2006). Continuous integration.

Maggi, D. (2020). Diseño de una arquitectura basada en contenedores para la integración y el despliegue continuo (CI/CD).

Cloud

Kavis, M. J. (2014). Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS). John Wiley & Sons.

Docker

Combe, T., Martin, A., & Di Pietro, R. (2016). To docker or not to docker: A security perspective. IEEE Cloud Computing, 3(5), 54-62.

Kubernetes

Sayfan, G. (2017). Mastering kubernetes. Packt Publishing Ltd.

Hightower, K., Burns, B., & Beda, J. (2017). Kubernetes: up and running: dive into the future of infrastructure. " O'Reilly Media, Inc."

Metodología Ágil

Fuentes, J.R. L. (2015). Desarrollo de Software ÁGIL: Extreme Programming y Scrum. ITCampus Academy.

Acrónimos

MVP Minimum Viable Product: Prototipo de una aplicación.

FTE Full-Time Equivalent: Es una forma de medir la cantidad de empleados a tiempo completo que se necesitan para llevar a cabo dicho trabajo.

IAAS Infraestructura como servicio

PAAS Plataforma como servicio

SAAS Software como servicio

SO Sistema Operativo

TD Transformación Digital

Referencias

- Aguaded-Gómez, J. I. (2011). Niños y adolescentes: nuevas generaciones interactivas. *Comunicar*, 18(36), 7-8.
- Dimes, T. (2015). *Conceptos Básicos de Scrum: Desarrollo de software Agile y manejo de proyectos Agile*. Babelcube Inc..
- Gartner Glossary (2015). *Information Technology Glossary*. Recuperado de <https://www.gartner.com/en/information-technology/glossary/devops>
- de Mary Poppendieck, P., & Anderson, D. (2010). *Kanban y Scrum—obteniendo lo mejor de ambos*. Prólogo de Mary Poppendieck & David Anderson. ESTADOS UNIDOS DE AMÉRICA: C4Media Inc.
- Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution.
- López, D., & Maya, E. (2017). *Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web*.
- Seo, K. T., Hwang, H. S., Moon, I. Y., Kwon, O. Y., & Kim, B. J. (2014). Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters*, 66(105-111), 2.