UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TITULACIÓN DE INGENIERO EN INFORMÁTICA

Definición de un ambiente de construcción de aplicaciones empresariales a través de Devops, Microservicios y Contenedores

TRABAJO DE FIN DE TITULACIÓN

AUTORA: Farías Alejandro, Ivonne Karina

DIRECTOR: Cabrera Silva, Armando Augusto, Ing.

CENTRO UNIVERSITARIO SALINAS

2017

Septiembre, 2017

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

"Yo Farías Alejandro Ivonne Karina declaro ser autora del presente trabajo de fin de titulación: Definición de un ambiente de construcción de aplicaciones empresariales a través de Devops, Microservicios y Contenedores, de la Titulación de Ingeniero en Informática, siendo el Ing. Armando Augusto Cabrera Silva director del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad".

f.
Autor Ivonne Karina Farías Alejandro
Cédula 0918914169

DEDICATORIA

Dedico este trabajo de tesis a Dios, el principal motor de mi vida y mi ayuda en todo momento.

A mi familia quienes me han apoyado incondicionalmente, cada uno ha representado un pilar fundamental para poder avanzar esta travesía.

AGRADECIMIENTO

Agradezco a Dios en primer lugar por ser mi guía y mi fortaleza, por renovar mis fuerzas cada día.

Agradezco a mi familia y amigos, a cada miembro de ella, mis padres, tíos, abuelitos, primos, por estar a mi lado, por su apoyo y comprensión.

Agradezco a mis maestros y compañeros de la Universidad Técnica Particular de Loja por compartir sus conocimientos y a la distancia hemos podido apoyarnos para poder cumplir una nueva meta.

Un agradecimiento especial a mi director de tesis el Ingeniero Armando Cabrera por sus consejos y la dirección respectiva para poder culminar este trabajo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

El presente trabajo de titulación presenta las definiciones para la construcción de un ambiente de aplicaciones adoptando una cultura Devops basados en una arquitectura de microservicios desplegados en contenedores.

El documento se enfoca en la aplicación de las prácticas de las metodologías ágiles y su integración con DevOps. Mediante la muestra de los puntos de enlaces entre sus fases y las prácticas continuas se expone como su empleo adecuado dentro de las organizaciones contribuye a tener mejores resultados para las empresas.

Se presenta el movimiento DevOps como una cultura de automatización de procesos dentro de las organizaciones, sus actores principales definidos bajo los conceptos de la metodología ágil Scrum y sus respectivas etapas.

En el trabajo también se muestran las herramientas que intervienen en cada una de las fases ciclo de vida de DevOps y aquellas que colaboran en la aplicación de las prácticas continuas con las que se integran. Con esto se pretende fortalecer los procesos que forman parte de una cultura DevOps usando microservicios desplegados en contenedores como Docker.

Palabras Clave: Arquitectura, DevOps, microservicios, contenedores, marco de trabajo.

ABSTRACT

This document of thesis presents the definitions for the construction of an environment of applications adopting a culture Devops based on an architecture of microservices deployed in containers.

The document focuses on the application of agile methodologies practices and their integration with DevOps. By showing the points of links between their phases and continuous practices it is exposed how their proper use within organizations contributes to have better results for companies.

The DevOps movement is presented as a process automation culture within the organizations, its main actors defined under the concepts of agile Scrum methodology and their respective stages.

The work also shows the tools involved in each of the life cycle phases of DevOps and those that collaborate in the application of continuous practices with which they are integrated. This is intended to strengthen processes that are part of a DevOps culture using microservices deployed in containers such as Docker.

Keywords: Architecture, DevOps, microservices, containers, framework.

INTRODUCCIÓN

No solo la tecnología avanza a grandes pasos sino que la forma en que las empresas se organizan también ha evolucionado, exigiendo con ello nuevas formas de comunicación entre el personal de los diferentes departamentos. Independientemente de las denominaciones de las empresas, siempre su objetivo principal será brindar un servicio o producto de calidad.

El ofrecer servicios tecnológicos que faciliten los procesos del negocio para los clientes se ha convertido en uno de los nuevos objetivos de algunas organizaciones que buscan la optimización en sus tareas.

Una mala estimación y demora en los tiempos de entrega de las aplicaciones representa la insatisfacción de los clientes y esto puede representar también graves pérdidas de ingresos para las empresas.

La adopción de una cultura DevOps contribuye no solo para el desarrollo de aplicaciones de software más rápidas, flexibles y ágiles sino que mantiene comunicados a los departamentos de las organizaciones. Hablar de DevOps es hacer énfasis en la automatización de procesos y mantener una comunicación directa con el área de operaciones se puede cumplir con los objetivos de la empresa con la mayor precisión.

El documento se enfoca siguiendo la metodología ágil Scrum que permite coordinar las tareas y actividades que intervienen en las fases de DevOps bajo sus componentes desde la planificación inicial, la construcción del backlog, el backlog del producto, la planeación del sprint.

Mediante las reuniones diarias se mantiene información actual sobre los avances del proyecto; por medio de las revisiones del sprint se pueden comprobar a tiempo las novedades presentadas y aprovechar el sprint retrospectivo para retroalimentar el backlog de producto y empezar un nuevo ciclo.

Siguiendo las prácticas de integración, entrega y despliegue continuos se espera que cada Sprint contenga las fases desde la planificación hasta el monitoreo.

La aplicación de una arquitectura de microservicios no solo contribuye con estos conceptos sino que facilita las prácticas continuas. Con el respectivo despliegue en contenedores se reducen los niveles de riesgos en producción, se pueden aislar los fallos y no afectar a toda la aplicación. Con este enfoque orientado a contenedores se puede eliminar la mayoría de los problemas que surgen al tener configuraciones de entorno incoherentes y los problemas que se derivan de ellos.

Para una automatización eficaz que genere valor a las organizaciones es necesario conocer las herramientas adecuadas para cada fase, ya sea desde la planificación, construcción, desarrollo, despliegue, pruebas o monitoreo.

La presentación de las herramientas que abarcan las fases del ciclo de vida de DevOps y las que intervienen en las prácticas continuas tanto de entrega, integración y despliegue se suman en este documento.

Objetivos

General

Definiciones de ambiente de construcción de aplicaciones empresariales a través de DevOps, microservicios y contenedores.

Específicos

1. Analizar DevOps e identificar su importancia en el desarrollo de aplicaciones.

2. Analizar la importancia e impacto de microservicios y contenedores en el despliegue de aplicaciones.

3. Definir un esquema de interacción e integración entre DevOps, Microservicios y Contenedores.

4. Presentar las herramientas requeridas para la construcción del ambiente de desarrollo.

CAPÍTULO I

MARCO TEÓRICO

1.1 Arquitectura de Software

La Arquitectura de Software representa el modelo a seguir para definir la estructura de la construcción de un sistema. Contiene los diseños, patrones, reglas y lineamientos que se aplican en el diseño e implementación de un software. Todo esto considerando los atributos de calidad como la seguridad, el rendimiento, confiabilidad y usabilidad.

"*La arquitectura comprende los conceptos y atributos esenciales en el ámbito de un sistema, representados en cada una de sus partes, vínculos y en los fundamentos que determinan su diseño y avance."* (ISO/IEC/IEEE 42010, 2011)

Con esto podemos decir que nos ayuda a definir la estructura del sistema mediante uno o varios esquemas que sirven de guía para la construcción de un software.

Hoy en día se pueden observar diversos conceptos de arquitectura de software; no obstante, es importante mencionar el SEI[1], donde existe una compilación de definiciones de varios autores expertos en el tema que expresa:

"Arquitectura de Software es un vehículo conceptual, por lo general representado en forma de diferentes puntos de vistas (lógica, capas, componentes, despliegues, sistemas) que guían y gobiernan el diseño, desarrollo, implementación y mantenimiento de las Aplicaciones de Software que permitan cumplir con los objetivos planteados, los requisitos (funcionales, no funcionales) en virtud de un determinado conjunto de restricciones (tales como el costo, el tiempo, el sistema, la plataforma y los usuarios)" (Software Engineering Institute - University Carnegie Mellon, 2016)

Según el estándar (ISO/IEC/IEEE 42010, 2011), la arquitectura comprende los conceptos y propiedades fundamentales de un sistema en su entorno plasmados en sus elementos, relaciones, y en los principios que rigen su diseño y evolución. La descripción de la arquitectura (AD) es un producto de trabajo utilizado para expresar la arquitectura. Ver Figura 1.

[1]Software Engineering Institute (Instituto de Ingeniería de Software). Fundado por el Congreso de los Estados Unidos en 1984 para administrar modelos de evaluación y mejora en el desarrollo de Software, administrado por la Universidad de Carnegie Mellon.

Un punto de vista arquitectónico es un producto de trabajo que establece los convenios para la construcción, interpretación y uso de vistas de arquitectura. Una vista expone la arquitectura de un sistema desde el punto de vista de los requerimientos específicos de la aplicación.

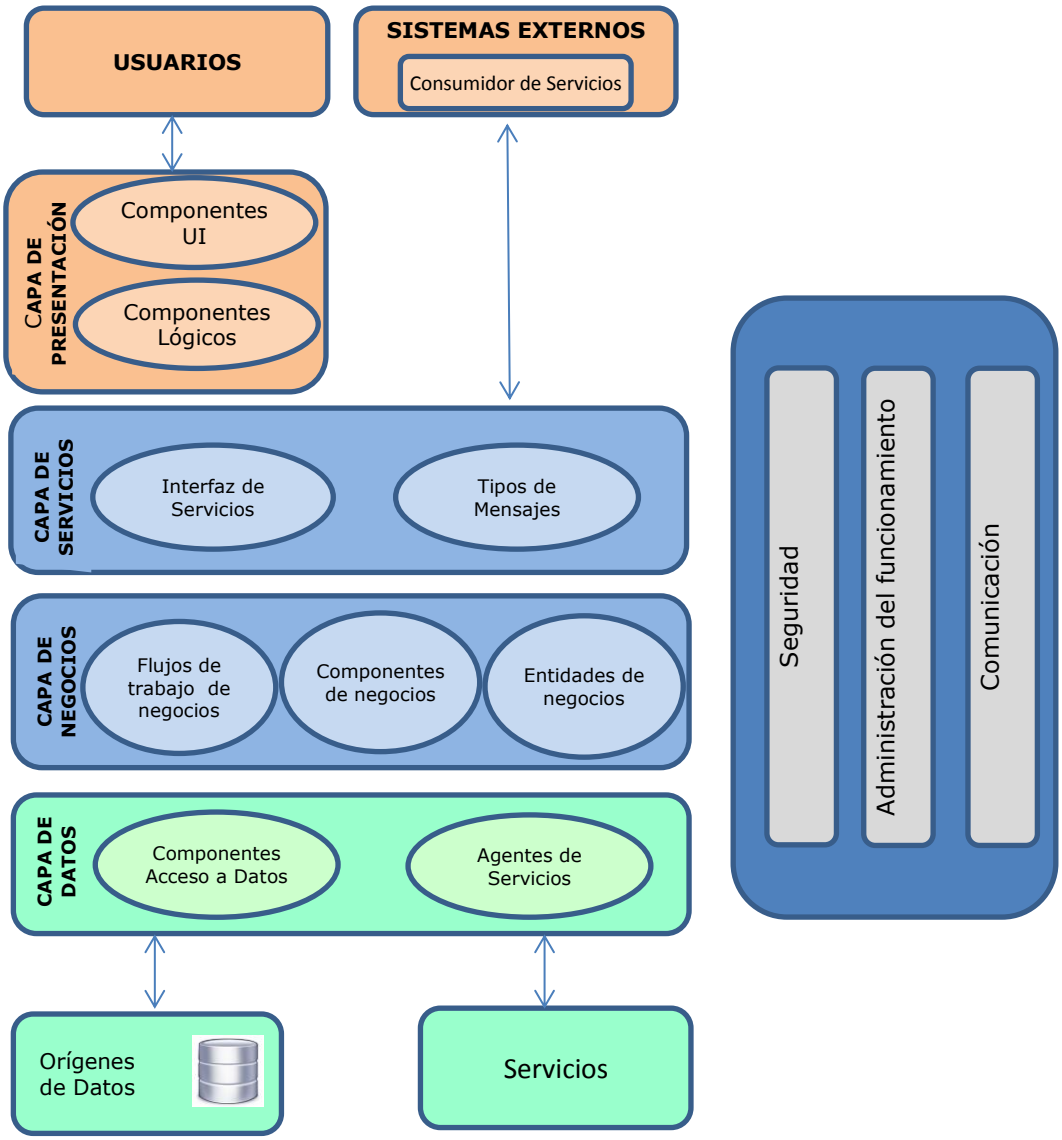

Figura 1: Descripción de Arquitectura de un estilo específico
Fuente: Adaptado de (Microsoft Corporation, 2009)

1.2 Diseño de la Arquitectura

Una de las decisiones más críticas de la Arquitectura de Software es elegir los estilos y patrones apropiados que den el soporte requerido para cumplir con los atributos de calidad deseados.

Al trabajar con microservicios y DevOps se debe tener claro el concepto de los estilos y patrones a usar que contribuyan a su correcta y ágil implementación.

1.2.1 Estilos de Arquitectura

En la actualidad existen una serie de definiciones para los estilos arquitectónicos de software; según una definición dada por (Campos, 2008), la cual parece ser la más acertada, donde indica que un estilo arquitectónico es "*una familia de sistemas en términos de patrón de organización estructural. Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que puede ser usado así como un conjunto de restricciones de cómo pueden ser combinados*".

Cabe recalcar que (Gutierrez, 2013) señala que cada estilo describe una clase de sistemas que comprende:

1. Un conjunto de componentes, como: BD y módulos computacionales; los cuales desempeñan una función solicitada por parte del sistema.
2. Dichos componentes necesitan un conjunto de conectores, los cuales permitirán que exista comunicación, coherencia y colaboración entre ellos.
3. Limitaciones que definen cómo se forman los componentes para crear el sistema.
4. Modelos semánticos que permiten al diseñador comprender las características generales de un sistema, mediante el análisis de las características de los elementos que lo componen.

En la Tabla 1 se citan las principales áreas de enfoque y los estilos arquitectónicos correspondientes:

Tabla 1: Estilos de Arquitectura

Categoría	Estilos de Arquitectura
Comunicación	Arquitectura orientada a servicios (SOA), Bus de mensajes
Despliegue	Cliente / Servidor, N-Tier, 3-Tier
Dominio	Diseño impulsado por dominio
Estructura	Arquitectura basada en componentes, orientada a objetos y en capas

Fuente: (Microsoft Corporation, 2009). Elaborado por Ivonne Karina Farías Alejandro

Es importante mencionar que los objetivos o metas que una organización se haya trazado a largo plazo, su capacidad para diseñar e implementar, su infraestructura o el conocimiento de los desarrolladores, pueden constituirse en factores de gran influencia al momento de seleccionar el estilo arquitectónico ideal.

1.2.2 Arquitectura en capas

En este tipo de arquitectura el sistema se organiza en capas, cada una de las cuales proporciona un conjunto de servicios a las capas superiores y demanda servicios de las inferiores. Ver Figura 2.

"*La arquitectura basada en capas se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de responsabilidades. El rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada*." (Microsoft Corporation, 2009)

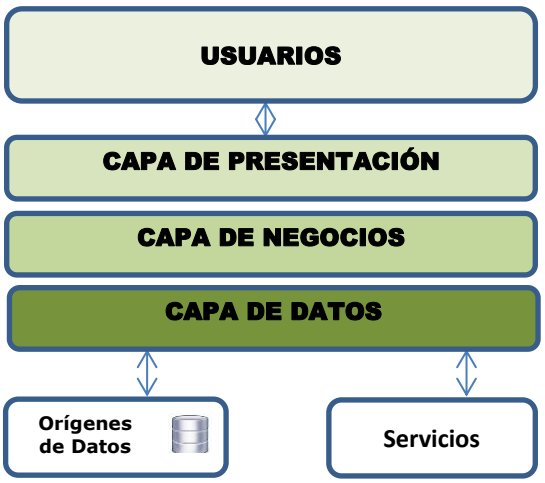

Figura 2: Tipos de componentes de la arquitectura en capas
Fuente: Adaptado de (Microsoft Corporation, 2009)

De acuerdo a la Figura 2 presentada, se torna preciso mencionar que la arquitectura en capas se encuentra integrada por algunos componentes (capas) como son:

- Componentes de interfaz de usuario, que constituye la capa de presentación.
- Componentes de lógica de negocio, lo cual hace referencia a las interfaces de servicios, las mismas que involucran a las entidades empresariales y sus respectivos flujos de trabajo.
- Componentes de acceso a datos, dicha capa incluye el origen de los datos.

Adicionalmente, trabajar con una arquitectura en capas tiene las siguientes ventajas:

- Admite la estandarización de servicios.
- Facilita el mantenimiento, ya que los cambios tan solo afectan a las capas contiguas.
- Proporciona una amplia reutilización, ya que las capas pueden intercambiarse debido a que cada interface es lo suficientemente clara.
- Existe poca dependencia entre una capa y otra.

1.2.3 Arquitectura Orientada a Servicios

Se puede definir una Arquitectura Orientada a Servicios como un enfoque de diseño que sirve para construir aplicaciones de negocio cuyas funciones son expuestas como servicios a los cuales se puede acceder en forma independiente usando interfaces estándares. Es un estilo que pertenece a la Arquitectura de Software para lo cual incluimos conceptos citados por Microsoft.

"La arquitectura orientada a servicios, conocida por sus siglas SOA, hace referencia a una perspectiva de mejora de las aplicaciones de software empresarial, donde los procesos del software se dividen en servicios, los cuales pueden ser visualizados y estar disponibles en una red. El nivel de desarticulación e interoperabilidad que proporciona admite un alto grado de reutilización (interno y externo) y de configuración". (Microsoft Corporation, 2016)

Según (Microsoft Corporation, 2009), "La arquitectura orientada a servicios permite que la funcionalidad de la aplicación se exponga y consuma como un conjunto de servicios. Los servicios usan una forma estándar de interacción que les permiten ser invocados, publicados y consumidos. Los servicios SOA están enfocados en proveer un esquema (schema) y una interacción basada en mensajes con una aplicación."

1.2.3.1 Vista Lógica de la Arquitectura SOA

El libro Executing SOA de (Bieberstein, Laird, Jones, & Mitra, 2008) presenta una vista lógica de referencia de esta arquitectura donde las instancias pueden ser desarrolladas para una plataforma y una tecnología específicas. En la Figura 3 se presenta esta vista lógica.

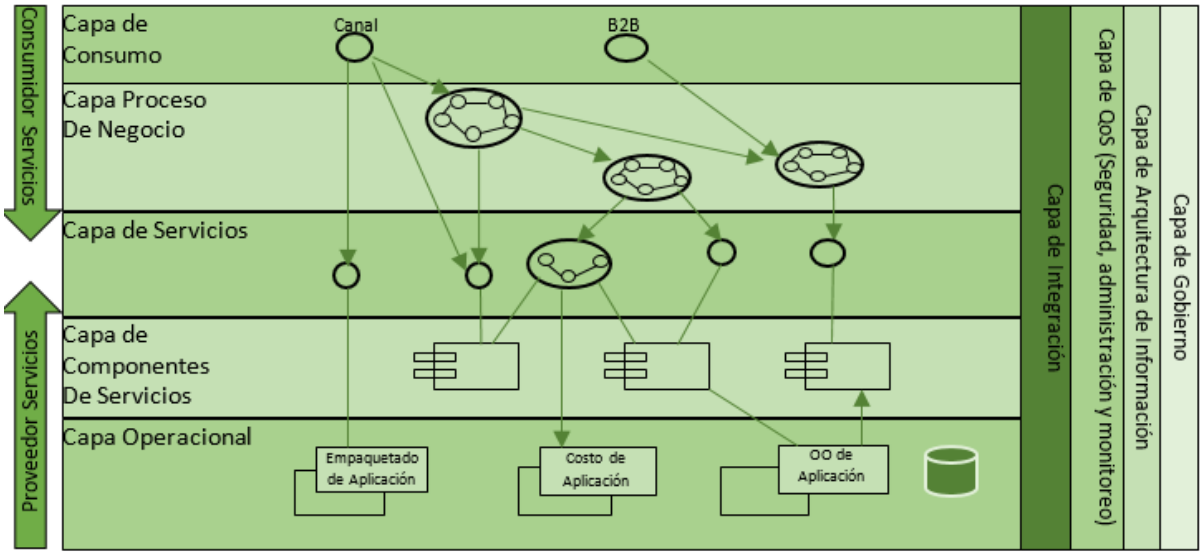

Figura 3: Vista Lógica de la Arquitectura SOA
Fuente: Adaptado de (Bieberstein et al., 2008)

A continuación se presentan las definiciones de cada una de las capas de esta vista lógica:

Capa 1: Sistemas Operacionales. Esta capa incluye los sistemas operativos que existen en el actual entorno de TI de la empresa, incluyen todas las aplicaciones personalizadas, aplicaciones de empaquetado, sistemas de procesamiento de transacciones y las diversas bases de datos.

Capa 2: Componentes de servicios. Los componentes de esta capa se ajustan a los contratos definidos por la capa de servicios para garantizar la calidad de servicio (QoS) y el cumplimiento de los acuerdos.

Capa 3: Capa de servicios. Incluye todos los servicios definidos en la empresa constituye tanto su carácter sintáctico (operaciones de servicios, la entrada y salida de mensajes, definición de fallos) y la información semántica (políticas de servicio, decisiones de gestión de servicios, requisitos de acceso a servicios).

Capa 4: Capa de procesos de Negocio. Los procesos de negocio describen cómo se ejecuta el negocio.

Capa 5: Capa de Consumo. En esta capa se encuentran los diversos canales a través de los cuales las funciones de TI se entregan. Esos canales pueden ser los consumidores externos e internos que acceden a la funcionalidad de la aplicación a través de mecanismos de acceso como sistemas B2B o portales.

Capa 6: Capa de Integración. Proporciona la capacidad para que los consumidores de servicios puedan localizar proveedores de servicios. A través de tres capacidades básicas de mediación, enrutamiento y transformación de datos y protocolos.

Capa 7: Capa QoS.: Se centra en la implementación y gestión de la capa no funcional (NFR) que los servicios necesitan implementar.

Capa 8: Capa Arquitectura de Información. Garantiza una representación adecuada de los datos e información que se requiere en una SOA junto con las consideraciones claves y las directrices para su diseño y uso.

Capa 9: Capa de Gobierno. Asegura la gestión adecuada en todo el ciclo de vida de los servicios. Esta capa es responsable de priorizar los servicios para cada una de las capas con el fin de cumplir una meta de negocio o TI.

1.2.3.2 Ciclo de Vida de un Servicio

Dentro del gobierno de SOA, los ciclos de vida ayudan para el trabajo con los procesos de la gestión de proyectos en colaboración con los arquitectos y el equipo de IT ajustándose al cumplimiento de las políticas.

En la Figura 4 podemos ver las fases del ciclo de vida de un Servicio dentro de una Arquitectura SOA.

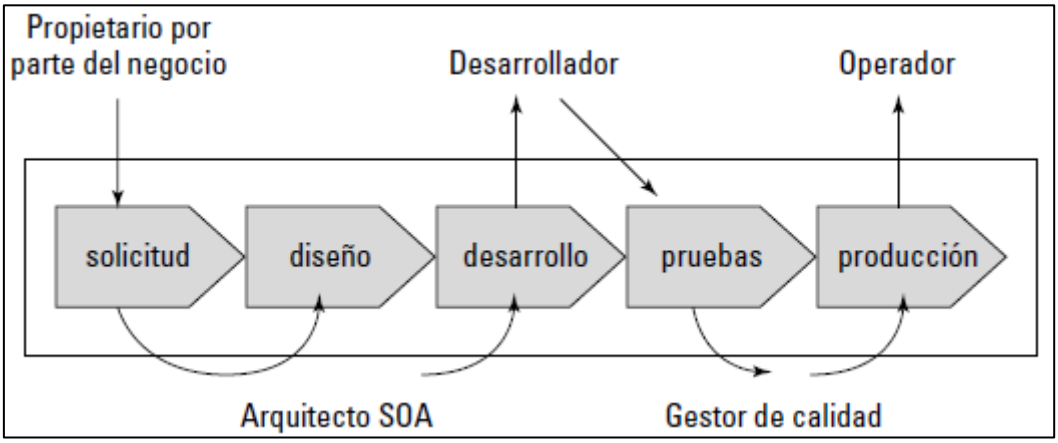

Figura 4: Ciclo de vida de un Servicio
Fuente: Adaptado de (Matsumura, Brauel, & Shah, 2009)

1.2.3.3 Principios SOA

Podemos definir a los principios como un conjunto de reglas que se deben aplicar dentro de los equipos de trabajo para implementar o mantener una Arquitectura Orientada a Servicios.

En el libro Principios de Diseño de Servicios por (Erl, 2009) se mencionan los siguientes principios:

- Contrato estandarizado
- Bajo Acoplamiento
- Abstracción
- Reusabilidad
- Autonomía
- Ausencia de estado
- Descubrimiento
- Combinación

1.2.4 Arquitectura de Microservicios

Una arquitectura de microservicios es un marco de desarrollo de aplicaciones basado en pequeños servicios que corresponden un área de negocio, los cuales son autónomos e independientes y tienen una intercomunicación entre ellos mediante peticiones HTTP.

Cada microservicio es desplegado sin causar afectaciones a los demás ya que solo exponen la API al resto de microservicios. En la Figura 5 se presenta una arquitectura de microservicios con múltiples lenguajes y tecnologías de almacenamiento de datos.

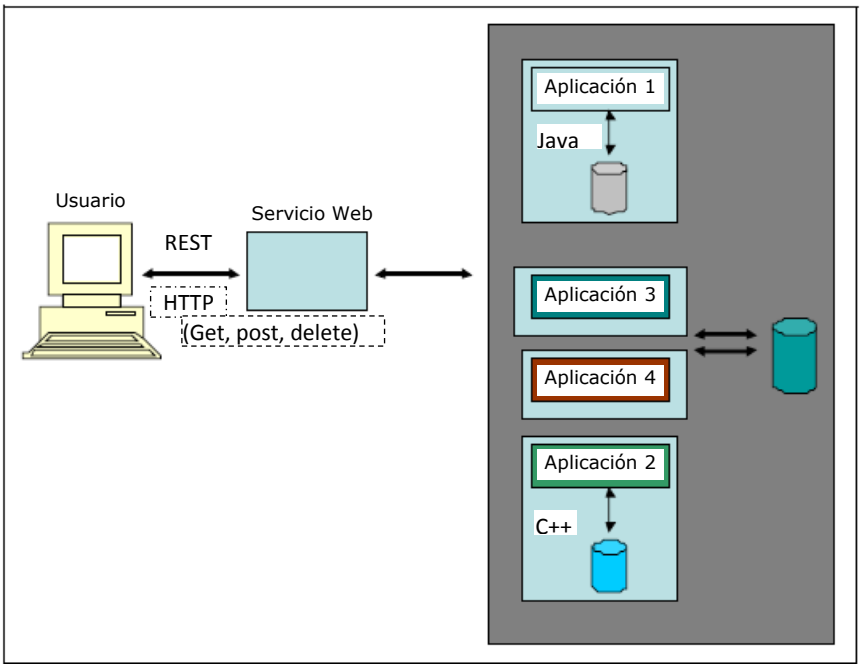

Figura 5: Arquitectura de Microservicios con múltiples lenguajes y tecnologías
Fuente: Adaptado de (IBM Corp., 2015)

Según (Fowler, 2014) *"El estilo arquitectónico de un microservicio, es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos ligeros, a menudo una API de recursos HTTP. Estos servicios se basan en capacidades empresariales y se pueden implementar de forma independiente mediante maquinaria de despliegue completamente automatizada. Existe un mínimo de gestión centralizada de estos servicios, que puede escribirse en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos."*

Entre los beneficios que podemos mencionar tenemos:

- Tienen un despliegue independiente.
- Son fáciles de entender debido a que la lógica del negocio se encuentra bien separada.
- Son multifuncionales.
- El nivel de escalabilidad en más fácil.

1.2.4.1 Despliegue de Microservicios

En (IBM Corp., 2015), se menciona que diseñar una arquitectura de microservicios abarca también en pensar en un modelo de despliegue, la forma en que se va a organizar y gestionar el despliegue de cada microservicio, de igual manera las herramientas tecnológicas que se van a usar. Los despliegues se pueden realizar a través máquinas virtuales o de contenedores.

Máquinas Virtuales

En esta opción el código se ejecuta en su propia máquina virtual con su propio sistema operativo (SO). Es responsable de proporcionar los servidores de aplicaciones y los tiempos de ejecución, por lo que usa más recursos en la construcción y gestión de imágenes.

Contenedores

Aquí se ejecuta el código de la aplicación en un contenedor (Ver Figura 6). Solo se debe proporcionar la aplicación ya que los servidores de aplicaciones y el tiempo de ejecución están incluidos en los contenedores, lo que facilita el despliegue.

En (de la Torre, 2016) menciona que un contenedor permite hacer las pruebas en entornos cerrados ya que se si presenta un fallo solo se volverá a desplegar en el contenedor afectado.

Contenerizar es una parte del desarrollo de software en donde las aplicaciones y sus versiones, dependencias, configuraciones de entorno son envasados dentro de un conjunto (imagen del contenedor), probados como unidad y desplegados en el Sistema Operativo.

En este enfoque orientado a contenedores se puede eliminar la mayoría de los problemas que surgen al tener configuraciones de entorno incoherentes y los problemas que se derivan de ellos.

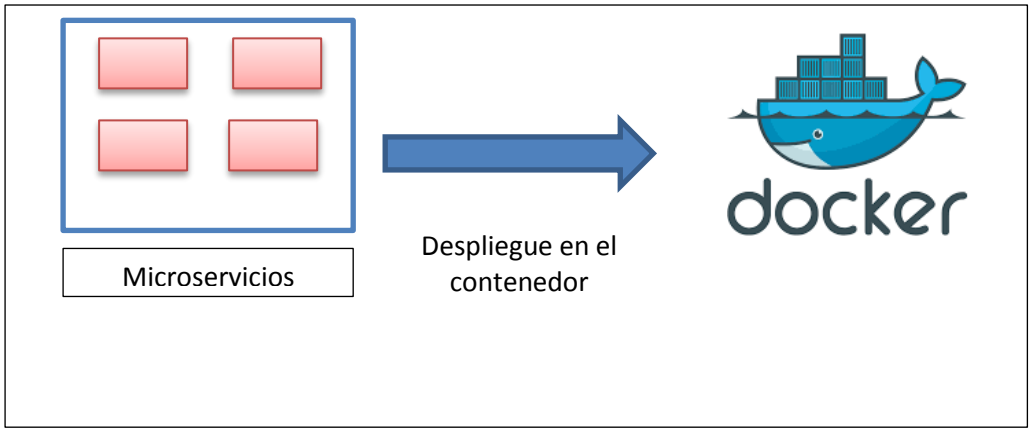

Figura 6: Despliegue en el contenedor
Fuente: Elaborado por el autor

Los contenedores separan las aplicaciones en un sistema operativo compartido. Esta práctica estandariza la entrega del programa de aplicación, permitiendo que las aplicaciones se ejecuten como contenedores de Linux o Windows. Dado que los contenedores comparten el mismo núcleo del sistema operativo (Linux o Windows), son mucho más ligeros que las imágenes de máquinas virtuales (VM) como nos indica (de la Torre, 2016).

Otro beneficio de usar contenedores es la capacidad de instanciar rápidamente. Por ejemplo, esto permite escalar más rápido al instante de instanciar una tarea específica a corto plazo en forma de un contenedor.

Resumiendo podemos mencionar que los principales beneficios proporcionados por los contenedores son aislamiento, agilidad, escalabilidad, portabilidad y control en todo el flujo de trabajo del ciclo de vida de la aplicación. Pero el beneficio más importante es el aislamiento proporcionado entre DevOps.

1.2.4.2 Docker

Docker es un proyecto open source creado para automatizar el despliegue de aplicaciones en contenedores portátiles que pueden ejecutarse ya sea en la nube o localmente. Docker también es una empresa que promueve y desarrolla esta tecnología en colaboración con proveedores como Linux y Microsoft.

Se está convirtiendo en un estándar de implementación ya que está siendo adoptada por la mayoría de proveedores de plataformas: Microsoft Azure, Amazon AWS, Google, etc.

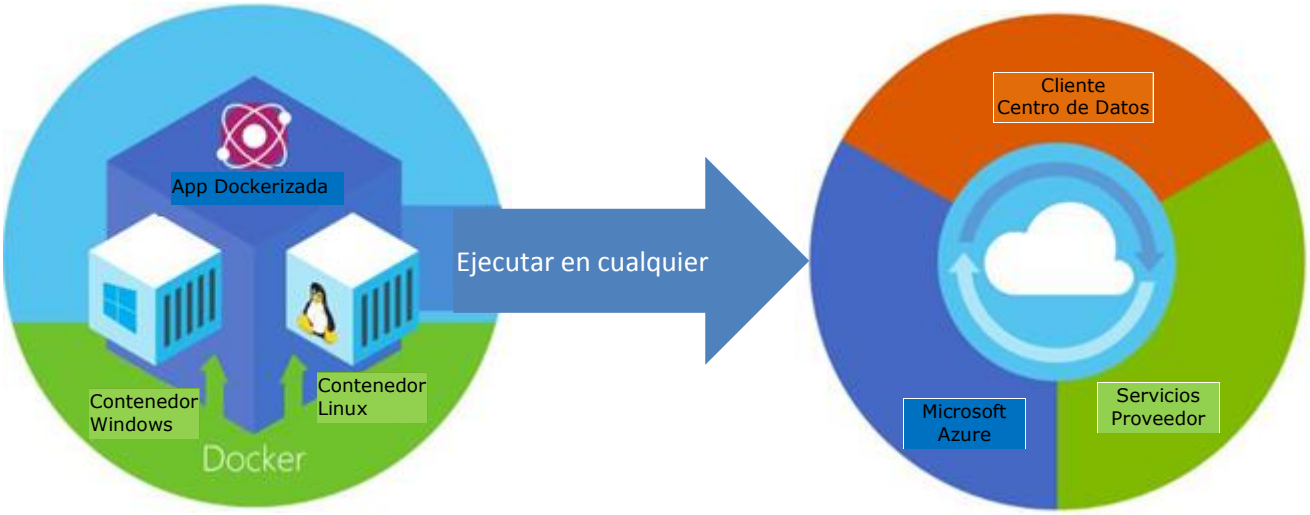

Figura 7: Despliegue en contenedores Docker
Fuente: Adaptado de (de la Torre, 2016)

Los contenedores Docker pueden ejecutarse en sistemas operativos de Linux y Windows como se muestra en la Figura 7. A su vez provee las herramientas de desarrollo para las diferentes plataformas.

En relación con los contenedores de Windows existen dos tipos:

- *Contenedores de Windows Server*
- *Contenedores Hyper-V*

Componentes de Docker

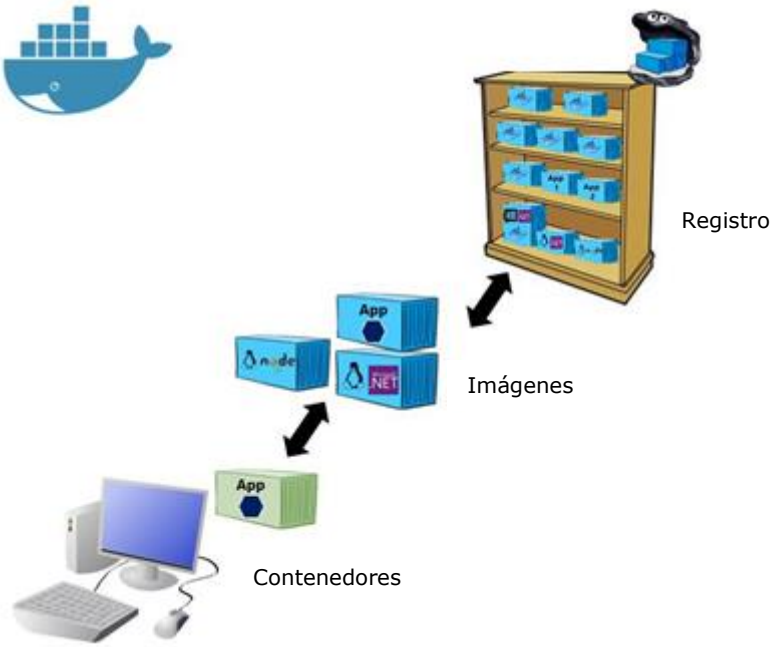

Figura 8. Componentes de Docker
Fuente: Adaptado de (de la Torre, 2016)

En la Figura 8 se presentan los componentes de Docker, los cuales se explican a continuación:

Contenedor: Es una o más instancias en tiempo de ejecución de una imagen de Docker que normalmente contendrá una sola aplicación o servicio.

Imagen: Es una colección ordenada de cambios en el sistema de archivos raíz y los parámetros correspondientes para su uso en tiempo de ejecución en un contenedor. No tiene estado y nunca cambia.

Registro: Es un servicio que contiene repositorios de imágenes de uno o más equipos de desarrollo. El registro predeterminado para Docker es el "Docker Hub".

1.2.4.3 Diseño de comunicación entre servicios

Para la comunicación unidireccional entre servicios podemos decir que las colas de mensajes son suficientes, pero para las comunicaciones bidireccionales podríamos optar por dos enfoques los cuales son mencionados por (Newman, 2015):

1. *Los mecanismos HTTP sincrónicos*, como REST (Transferencia de Estado Representacional), SOAP o WebSockets, los cuales permiten mantener un canal de comunicación abierto entre el navegador y servidor para manejar los eventos request/ response (solicitud/respuesta).

2. *Mensajería asíncrona,* mediante un broker (intermediario) de mensajes

Una arquitectura basada en microservicios por lo general aplica el estilo REST con referencia a los estándares del protocolo HTTP.

1.3 Patrones de Arquitectura

Elegir el patrón arquitectónico que más se adapte a nuestra estructura debe ser una tarea primordial para las organizaciones en especial antes de empezar el diseño para la elaboración de las aplicaciones, en ocasiones hay veces que se necesita combinar dos patrones, todo varía dependiendo de las necesidades del negocio y los requerimientos que se pidan.

(Sommerville, 2011) define a un patrón arquitectónico como *"una descripción abstracta estilizada de buena práctica, que se ensayó y puso a prueba en diferentes sistemas y entornos. De este modo, un patrón arquitectónico debe describir una organización de sistema que ha tenido éxito en sistemas previos."*

Un patrón expresa una relación entre un contexto, un problema y una solución, tal como se detalla en la Tabla 2.

Tabla 2: Esquema de un Patrón

Contexto	Es una situación de diseño en la que aparece un problema de diseño.
Problema	Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
Solución	Es una configuración que equilibra estas fuerzas. Ésta abarca: • Estructura con componentes y relaciones • Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

Fuente: (Camacho, 2004). Elaborado por Ivonne Karina Farías Alejandro

Dentro de los patrones arquitectónicos se pueden mencionar: Capas (Layers), Tuberías y Filtros (Pipes and Filters), Pizarra (Blackboard), Broker, Modelo-Vista-Controlador (Model-View–Controller).

1.3.1 Patrón arquitectónico Modelo Vista Controlador (Model-View-Controler)

El MVC o más conocido como el patrón Modelo Vista Controlador está compuesto por tres estructuras como su nombre lo indica: Modelo-Vista-Controlador.

A continuación se describen cada uno de los componentes antes mencionados:

Modelo: Tiene los datos necesarios para el manejo del sistema, es una representación de lo que ejecutará el programa.

Vista: Es una representación del modelo y la interacción con el usuario. Generalmente es la interfaz del usuario.

Controlador: Es el intermediario entre el modelo y la vista. Gestiona la información y controla los datos entre ellos.

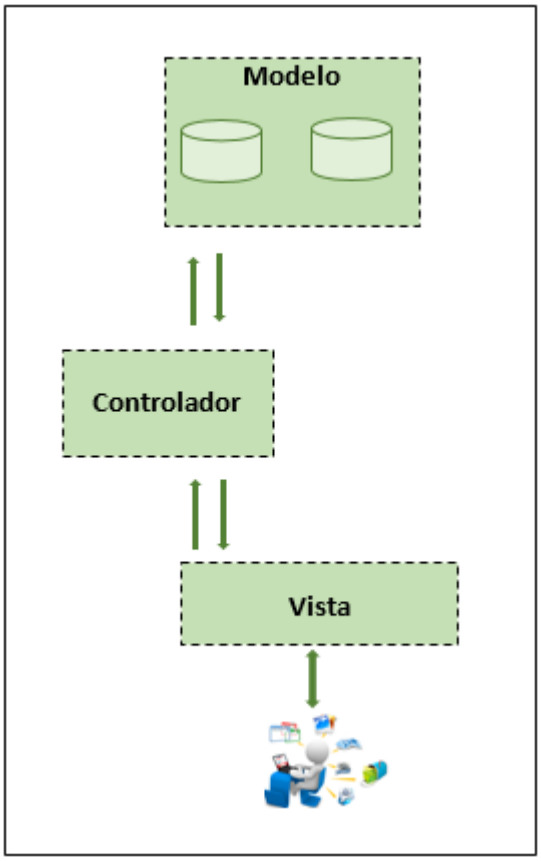

Figura 9: Modelo vista controlador (MVC)
Fuente: (Reinoso & Kicillof, 2004). Adaptado por Ivonne Karina Farías Alejandro

1.3.2 Diseño para una Estructura en Capas

En la guía de estudio de la Arquitectura de Software de (Camacho, 2004), se menciona que las capas representan conjuntos lógicos, los cuales sirven de apoyo para distinguir los diferentes tipos de tareas efectuadas por los componentes, esto facilita la implementación de un diseño que permite la reutilización de componentes. Se torna preciso destacar que estas capas no cuentan con la localización física de los componentes.

A continuación se detalla cada uno de los pasos que (Camacho, 2004) sugieren para su diseño:

1. Seleccionar la táctica de estratificación respectiva.
2. Establecer las capas que se requiere.
3. Resolver la forma en que se realizará la distribución de las capas y componentes.
4. Determinar si es preciso destruir capas.

5. Fijar reglas que sirvan para la intercomunicación entre capas.
6. Distinguir las expectaciones de corte transversal.
7. Delimitar las interfaces entre capas.
8. Seleccionar una estrategia de ejecución.
9. Elegir adecuados protocolos de comunicación.

1.3.2.1 Capa de Presentación

En la capa de presentación se maneja o expone todo lo que se refiere a la interfaz de usuario, aquí encontramos todos aquellos componentes que son necesarios para interactuar con el usuario.

- **Componentes de la Interfaz del Usuario:** son los elementos que conforman la interfaz.
- **Presentación de Componentes lógicos:** Todo lo referente al código de la aplicación.

Posteriormente se describe el proceso para el diseño de la capa de presentación, el cual consta exactamente de seis pasos, que según (Microsoft Corporation, 2016) son:

1. Identificación del tipo de cliente.
2. Selección de la tecnología adecuada.
3. Planteamiento de la interfaz del usuario.
4. Establecimiento de una estrategia que permita constatar la autenticidad de los datos.
5. Definición de la respectiva estrategia de lógica del negocio.
6. Determinación de la estrategia apropiada, mediante la cual se logre obtener comunicación con otras capas.

1.3.2.2 Capa de Negocio

En esta capa se emplean las pautas claves para llevar a cabo el diseño de la capa empresarial de una aplicación. Según (Microsoft Corporation, 2016), el propósito del arquitecto radica en mermar la complejidad al realizar la división de las tareas en diversas áreas de interés.

A continuación se detalla el proceso para realizar el diseño de esta capa, el mismo que consta tan solo de tres pasos según lo señala:

1. Construcción de un diseño de alto nivel para la capa de negocio.
2. Planteamiento de los respectivos componentes del negocio.
3. Diseño de los componentes del organismo institucional.

1.3.2.3 Capa de Datos

La capa de datos generalmente está integrada por: a) Componentes de acceso a datos y b) Agentes de servicio. Los pasos correspondientes para el diseño de esta capa se detallan a continuación:

1. Confeccione un diseño general que permita el acceso a datos.
2. Seleccione los tipos de organismos que requiere.
3. Seleccione el tipo de tecnología a utilizarse para poder acceder a los datos.
4. Plantee los componentes de acceso a datos.
5. Plantee los agentes de servicio.

1.3.2.4 Capa de Servicios

La capa de servicios es donde se determina y efectúa la interfaz de servicio y los contratos de datos. Básicamente contiene: a) Interfaces de servicio y b) Tipos de mensajes.

Para ejecutar la implementación de los servicios (Microsoft Corporation, 2016) sugiere dos modalidades distintas, que son: REST y SOAP; la primera admite un conjunto restringido de operaciones, las cuales se aplican a los recursos representados; en cambio la segunda consigue el desplazamiento por medio de diversos estados y la interrelación con un punto final . De acuerdo a lo que se ha mencionado, a continuación se detallará el proceso que debe seguirse para el diseño de esta capa:

1. Especificación de los contratos de datos y mensajes que personifican el esquema empleado para los mensajes.
2. Especificación de los contratos de servicio, los cuales se distinguen por las operaciones que se encuentran sustentadas por su servicio.
3. Especificación de los contratos con inconsistencias, que reintegren a los consumidores del servicio la respectiva información que posee falencias.
4. Planteamiento de objetos de innovación que traducen entre organismos institucionales y contratos de datos.

Diseño del enfoque de abstracción usado para interrelacionarse con la capa de negocio.

1.4 Vista Arquitectónica de Philippe Kruchten-Modelo 4+1

El autor (Kruchten, 2006) planteó el modelo 4+1, afín al Rational Unified Process (RUP). Dicho modelo establece la representación de la arquitectura de un software a través de la utilización de cinco vistas concurrentes, las mismas que están dirigidas a un conjunto determinado de conceptos.

Las primeras cuatro vistas son una representación de las decisiones diseño y la quinta se usa para proyectar o aprobar las decisiones. Ver Figura 10.

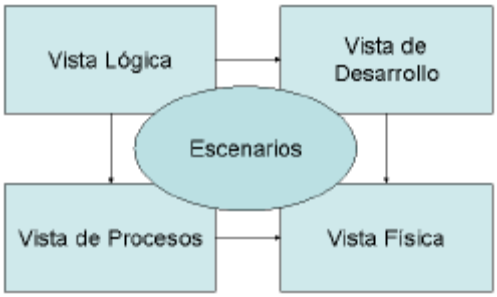

Figura 10: Modelo 4 + 1 Vistas
Fuente: Adaptado de (Kruchten, 2006)

A continuación se describen cada una de las vistas del modelo de representación arquitectónica 4 + 1 de (Kruchten, 2006):

Vista Lógica: Constituye primordialmente la funcionalidad que el sistema suministra a los usuarios finales; por lo tanto, describe las acciones que el sistema debe realizar, las funciones y por ende los servicios que otorga el mismo. Cabe resaltar que la abstracción, encapsulamiento y herencia, se consideran principios que deben emplearse con la finalidad de identificar mecanismos de diseño a varios elementos del sistema. Sirven de complemento para esta vista los diagramas de clases, de comunicación o de secuencia de UML.

Vista de proceso: Se encuentra orientada a la exposición de contenidos de concurrencia y distribución, ecuanimidad del sistema, y de tolerancia a posibles inconsistencias. En adición a esto, la vista de procesos además señala el hilo de control donde se lleva a cabo una operación de una clase identificada previamente en la vista lógica. Sirve de complemento para esta vista el diagrama de actividad de UML.

Vista Física: Aquí se exponen todos los componentes del sistema desde la óptica de un ingeniero en sistemas. Fundamentalmente este tipo de vista se utiliza para modelar el hardware empleado en la ejecución del sistema y las relaciones entre sus componentes. Sirve de complemento para esta vista el diagrama de despliegue de UML.

Vista de Desarrollo: También se conoce como vista de despliegue; se enfoca en la organización de los módulos del software que actualmente se está utilizando, pero desde el punto de vista de un programador, el cual se encargará de mostrar la forma en que está estructurado el sistema de software, sus componentes y respetivas dependencias. Sirven de complemento para esta vista los diagramas de componentes y de paquetes de UML.

Escenarios: Cumple con la función de articular y relacionar las cuatro vistas anteriormente descritas; es decir, los creadores del software organizan la representación de sus decisiones arquitecturales en torno a las cuatro vistas, y las ilustran mediante el escogimiento de casos de uso o espacios, estableciendo de esta manera la quinta vista. Sirve de complemento para esta vista el diagrama de casos de uso de UML.

1.5 Métodos Ágiles y DevOps

En los Principios del Manifiesto Ágil de (Manifesto for Agile Software, 2001) se hace un énfasis en la comunicación y la colaboración, el funcionamiento del software, la organización del equipo y la flexibilidad para adaptarse a las realidades de los negocios emergentes.

Desarrollar software ágil requiere colaborar activamente con el cliente, lo que pone en discusión los contratos, las ventas, el mantenimiento y las relaciones de comunicación; en otras palabras, toda la organización. Una empresa que adopta un modelo ágil deberá hacer cambios en todas estas áreas.

Los equipos ágiles son muy disciplinados, escriben y prueban casos antes de escribir un nuevo código. Comprueban e integran el código mostrando los resultados a los usuarios cada semana. Se debe considera que los métodos agiles son un punto primordial para trabajar con DevOps.

Conocer las metodologías como Scaled Agile (SAFe), Disciplined Agile Delivery (DAD) y Scrum servirán de apoyo para implementar una Cultura de DevOps. El presente trabajo basará sus procesos en Scrum, pero se considera necesario presentar globalmente la definición de SAFe el cual incluye obligatoriamente a los DevOps en su framework.

1.5.1 Scaled Agile (SAFe)

El marco de escala ágil, Scaled Agile Framework (SAFe) es una base de conocimiento de patrones probados e integrados para el desarrollo ágil de escala empresarial. Es escalable y modular, permitiendo que cada organización pueda aplicarlo.

Los roles de un equipo ágil incluyen:

- Scrum Master
- Dueño del Producto
- Equipo de Desarrollo

El sitio *scaledagileframework.com* presenta una guía completa para implementar este marco en cada nivel de una organización.

1.5.2 Scrum

(Rubin, 2013) lo define como una metodología para el desarrollo de software ágil, es un conjunto de roles y responsabilidades que permiten el descubrimiento y aprendizaje continuos. Scrum hace énfasis en la toma de decisiones a partir de los resultados obtenidos en el mundo real.

La correcta gestión del tiempo es considerada una de las partes más importantes para Scrum y para llevar esto a cabo ha fijado una cantidad de tiempo para realizar los procesos y actividades dentro de sus proyectos garantizando así que el equipo no sobrepase los límites establecidos para determinados trabajos y evitando los desperdicios de tiempo y energía de los miembros del equipo.

Sprints

Scrum divide el tiempo en periodos cortos de trabajo conocidos como Sprints, generalmente en una o dos semanas de duración.

1.5.2.1 Principios de Scrum

En (Satpathy, 2016) se presentan seis principios que se pueden aplicar en distintos tipos de proyectos Scrum para todas las organizaciones:

1. Control del proceso empírico.
2. Auto-organización.
3. Colaboración.
4. Priorización basada en el valor.
5. Asignación de un bloque de tiempo.
6. Desarrollo iterativo.

1.5.2.2 Fases y Procesos de Scrum

En la Tabla 3 se presentan las fases y procesos que se deben considerar al momento de la implementación de Scrum para los proyectos. En la columna de procesos se listan las actividades y el orden específico que se debe seguir en un proyecto.

Tabla 3: Fases y Procesos de Scrum

Fases	Procesos
Inicio	• Crear visión del proyecto • Identificar el Scrum Master • Formar equipos Scrum • Desarrollar épicas • Crear lista priorizada de pendientes • Realizar planificación del lanzamiento
Planificación y Estimación	• Crear historias de usuario • Aprobar, estimar y asignar historias de usuario • Crear tareas • Estimar tareas • Crear lista de pendientes del sprint
Implementación	• Crear entregables • Realizar reuniones diarias • Mantener lista priorizada de pendientes
Revisión y retrospectiva	• Convocar Scrums • Demostrar y validar el sprint • Retrospectiva del sprint
Lanzamiento	• Envío de entregables • Retrospectiva del proyecto

Fuente: Adaptado de (Satpathy, 2016)

1.5.2.3 Framework de Scrum

En este apartado se van a presentar los componentes del marco de trabajo de Scrum, los cuales serán aplicados para el desarrollo de los posteriores capítulos. Es importante entender y familiarizarse con estas definiciones ya que son la base del marco de trabajo de esta metodología.

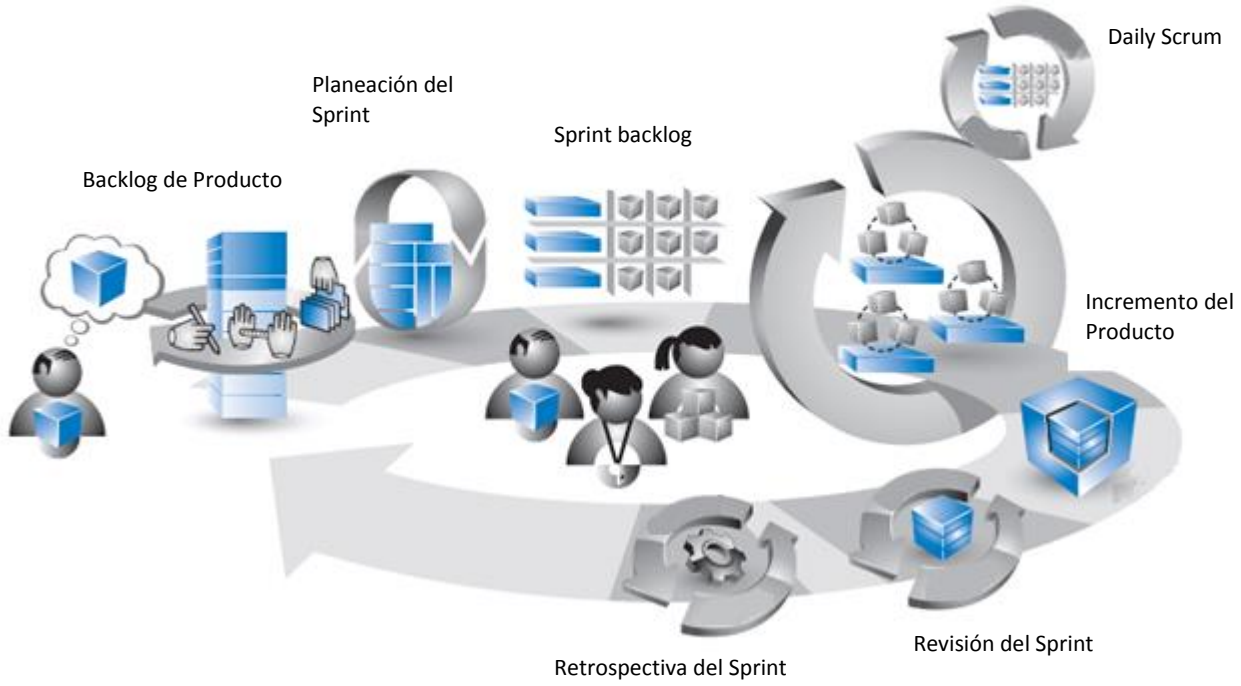

Figura 11: Framework de Scrum
Fuente: Adaptado de (Rubin, 2013)

En la Figura 11 se presenta el marco de trabajo de Scrum con sus principales componentes y el respectivo flujo, dando inicio con el backlog del producto, la planeación del sprint, definiendo el sprint backlog, los Daily Scrum, el Incremento del producto, la revisión del sprint y la retrospectiva del sprint, alimentando de esta forma todo un ciclo de trabajo.

A continuación se va a describir cada uno de los componentes de acuerdo a las definiciones mencionadas por (Satpathy, 2016) en la guía para el cuerpo de conocimiento de Scrum presentados en la Tabla 4.

Tabla 4: Componentes del Framework de Scrum

Componente	Descripción
Backlog de Producto	La cartera de productos es una lista priorizada con las funcionalidades del producto y ayuda a la comprensión de lo que se debe construir.
Planeación del Sprint	Está a cargo del Scrum Master y el equipo de desarrollo, aquí se planifica como se puede completar los trabajos en los Sprints y lo que se puede lograr.
Sprint Backlog	Describe a través de un conjunto de tareas como diseñar, construir, integrar y probar el conjunto de funciones de la cartera de productos de un sprint.
Daily Scrum	Es una actividad diaria de todos los miembros del equipo de desarrollo de máximo 15 minutos.
Incremento del Producto	El incremento del producto se refiere a los resultados del sprint.
Revisión del Sprint	Su objetivo es inspeccionar el producto que se está construyendo, aquí intervienen los clientes, los patrocinadores y los miembros del equipo de Scrum.
Retrospectiva del Sprint	Su objetivo es inspeccionar el proceso durante el sprint. El Scrum Master, el propietario del producto y el equipo de desarrollo se reúnen en esta actividad.

Fuente: Adaptado de (Satpathy, 2016)

1.6 Prácticas ágiles de Desarrollo de Software

Luego de citar los métodos agiles más utilizados ahora se van a presentar las prácticas ágiles que se utilizan en su desarrollo y las cuales son necesarias en una cultura DevOps.

Para este punto primero se mencionará los principios del Manifiesto Ágil que se toman como base para estas prácticas.

- Satisfacer al cliente mediante la entrega temprana y continua de software.

- Entregar el software frecuentemente, puede ser entre dos semanas y dos meses, preferible en el tiempo más corto posible.

- Trabajar en conjunto tanto los desarrolladores como los responsables del negocio durante todo el proyecto.

Siguiendo estos principios en este documento se va a trabajar con las prácticas conocidas como Entrega Continua, Despliegue Continuo e Integración Continua.

1.6.1 Entrega Continua

Es más que una nueva práctica de la metodología ágil, ha sido diseñada para brindar un mayor rendimiento y que los equipos se mantengan en constante retroalimentación sobre su aplicación. Las aplicaciones deben estar siempre en un estado disponible para el despliegue.

Tanto la empresa como el equipo de TI pueden probar la aplicación en cualquier momento. Se tiene un sistema de registro de las versiones de cada aplicación, mediante el uso de herramientas adecuadas que permiten llevar un control desde el primer momento de la entrega.

Según (Fowler, 2014) *"la entrega continua es una disciplina de desarrollo software en la que la aplicación se construye de una forma que puede ser liberada a producción en cualquier momento*".

.

Principios de la Entrega Continua

En (Humble & Farley, 2011) se mencionan ocho principios asociados a las entregas continuas:

1. Crear un proceso repetible y confiable para la liberación del software.
2. Automatizar todo lo que se pueda.
3. Mantener todo dentro de un control de versiones.
4. Si duele, hágalo con más frecuencia y traiga el dolor hacia adelante.
5. Construir con calidad.
6. Hecho significa liberado.
7. Todo el mundo es responsable del proceso de entrega.
8. Mejora Continua.

1.6.2 Despliegue Continuo

En esta práctica se sigue el pipeline trazado y se lo aplica en producción. De esta forma si el checkin pasa todas las pruebas requeridas se pasa al despliegue en producción directamente.

Pero para evitar posibles incidentes las pruebas deben ser exhaustivas, pruebas de componentes, pruebas de aceptación ya sean funcionales y no funcionales de la aplicación completa.

Debemos considerar que para tener Despliegue Continuo primero debe haber una Entrega Continua y en este caso también existir una Integración Continua.

Mediante el uso de diversas herramientas de automatización y de integración continua el proceso de entrega en producción de la aplicación se agiliza, garantizando también la calidad y minimizando los costos ocasionados por los incidentes.

1.6.3 Integración Continua

Se refiere a que cada vez que alguien realiza algún cambio, toda la aplicación se construye y se ejecutan un conjunto de pruebas. De esta manera si algo falla en el proceso de construcción o prueba, el equipo de desarrollo detiene lo que está haciendo y lo corrige inmediatamente.

Con la integración continua la aplicación está probada y lista para funcionar con cada cambio conociendo el momento en que podría fallar para poder arreglarlo de inmediato.

Requerimientos de la Integración Continua

Estos son algunos requerimientos que se deben tener en cuenta para la Integración Continua:

1. Control de Versiones.
2. Una estructura automatizada.
3. Acuerdo del Equipo.

Una vez revisados los conceptos de estas prácticas agiles, es necesario resaltar que para poder implementarlas se requiere que el personal encargado tenga un rol DevOps y que posean conocimientos sólidos en cuanto a la calidad en el desarrollo de los sistemas. Ver Figura 12.

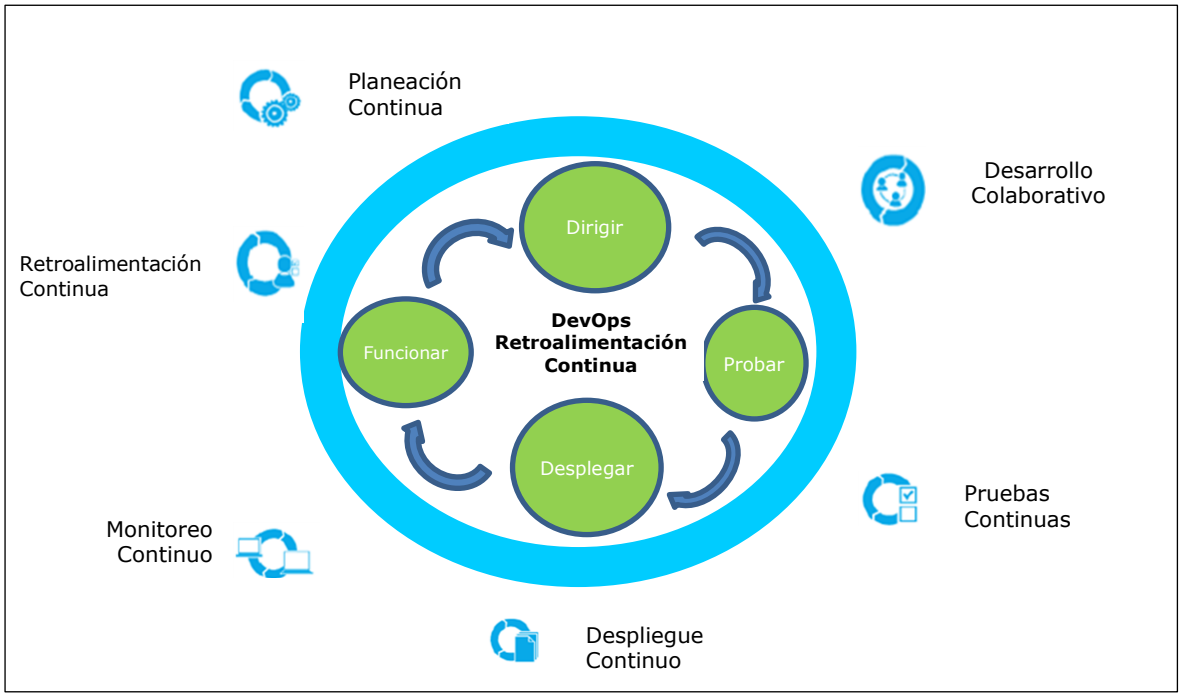

Figura 12: Arquitectura de referencia de DevOps
Fuente: Adaptado de (Sharma & Coyne, 2015)

1.7 Movimiento DevOps

DevOps es uno de los términos más citados en el entorno actual de TI, generalmente está asociado a estrategias de transformación digital y a metodologías de desarrollo ágil.

(Davis & Daniels, 2016) presenta las siglas en inglés de development (desarrollo) y operations (operaciones), que hacen referencia a una cultura o movimiento basados en la colaboración, integración y comunicación entre desarrolladores y profesionales IT.

Según (Davis & Daniels, 2016) existen cuatro áreas de habilidades que se requieren para quienes deseen trabajar en DevOps:

1. Codificación o scripting.
2. Infraestructura.

3. Reingeniería de procesos.
4. Comunicación y colaboración con otros.

En base a esto se puede decir que entre los atributos principales de los DevOps se encuentran:

- Capacidad de utilizar una amplia variedad de tecnologías y herramientas de código abierto.
- Capacidad de código y secuencia de comandos.
- Experiencia con sistemas y operaciones de TI.
- Comodidad con pruebas y despliegues frecuentes de código incremental.
- Fuerte comprensión de las herramientas de automatización.
- Habilidades de gestión de datos.
- Un enfoque fuerte en los resultados empresariales.
- Comodidad con la colaboración, comunicación abierta y alcance a través de las fronteras funcionales.

Para implementar una cultura DevOps dentro de una empresa es importante considerar tres aspectos: personal, procesos y tecnología. A veces resulta necesario hacer cambios organizacionales para apoyar una cultura de responsabilidades compartidas aquí no debe haber silos entre el desarrollo y las operaciones.

Así de la misma forma como se requiere transformar la organización y los procesos empresariales, puede que también se necesite realizar una transición con las plataformas tecnológicas donde se debe buscar modelos de entrega y tecnologías nuevos, como la nube y la virtualización, para facilitar la administración de TI y para inyectar mayor agilidad en la infraestructura.

La automatización se ha convertido en un pilar tecnológico indiscutible en DevOps donde herramientas tecnológicas que se ofrecen en el mercado actual como Docker, Puppet, Jenkins o AWS Lambda deben ser consideraras una u otra para acelerar los procesos de desarrollo.

En la actualidad el desarrollo de aplicaciones ha ido evolucionando hacia una forma más ágil de trabajar, tener equipos de desarrollo preparados para atender los requerimientos de

forma disciplinada, rápida y eficaz es ya una necesidad para las empresas donde la parte de negocio y la parte de desarrollo aparecen representadas en un solo proyecto.

DevOps es un nuevo movimiento que trata de mejorar la agilidad en la prestación de servicios el cual fomenta una mayor colaboración y comunicación entre los equipos de desarrollo y operaciones y evita que los problemas y las necesidades operacionales afecten la calidad del software.

1.7.1 Origen

(Davis & Daniels, 2016), relata el comienzo del movimiento que se dio en la conferencia Agile'08 celebrada en agosto del 2008 en Toronto-Canadá, donde el consultor Patrick Debois expuso cómo se podría llevar el agilismo al mundo de la infraestructura y la administración de sistemas.

En (Davis & Daniels, 2016) se destaca que Patrick Debois había vivido una experiencia muy frustrante en un proyecto para el Ministerio de Finanzas en Bélgica que ni desarrolladores de software ni administradores de sistemas habían logrado sacar adelante y fue durante esta conferencia donde presentó y argumentó por primera vez el concepto. Los profesionales en el desarrollo ágil y las prácticas de entrega continua necesitaban algo más entonces la idea de desarrollar una mayor empatía entre las unidades de negocio (principalmente Desarrollo y Operaciones) y centrarse en las maneras de acercarlas en nombre de mayores eficiencias y ciclos más rápidos de entrega de software fue dada vida y presentada simplemente como "DevOps".

Sin embargo después de celebrarse los "DevOps Days" en Bélgica fue cuando el término "DevOps" se hizo popular en el 2009. La repercusión fue enorme, y el hashtag creado para la ocasión, #DevOps, triunfó en las redes sociales de forma viral, dando nombre a todo un movimiento.

1.7.2 Definición y Objetivo

DevOps es uno de los términos más citados en el entorno actual de TI, generalmente está asociado a estrategias de transformación digital y a metodologías de desarrollo ágil.

En el IT Glosary de (Gartner, 2017) se define DevOps *"un cambio en la cultura de TI, centrándose en la prestación rápida de servicios de TI a través de la adopción de prácticas*

ágiles y mansas en el contexto de un enfoque orientado al sistema. DevOps hace hincapié en la gente (y la cultura), y busca mejorar la colaboración entre las operaciones y los equipos de desarrollo. Las implementaciones de DevOps utilizan tecnología, especialmente herramientas de automatización que pueden aprovechar una infraestructura cada vez más programable y dinámica desde la perspectiva del ciclo de vida."

En la Figura 13 se puede observar la integración, colaboración y comunicación entre el equipo de desarrollo y el equipo de operaciones.

Figura 13: ¿Qué es DevOps?
Fuente: Autor

Además (IBM Corp., 2015) define a DevOps como *"un conjunto de herramientas, conceptos, prácticas y equipo de trabajo estructurado que permite a las empresas una liberación más rápida de sus productos para la satisfacción de sus clientes".* Las empresas que han adoptado DevOps ofrecen con mayor facilidad la liberación y monitoreo de los microservicios, respondiendo así más rápidamente a los requerimientos y resolviendo los problemas que se presentan en producción

Los procesos que se deben considerar al trabajar con DevOps son:

- Métodos Ágiles.
- Integración Continua.
- Entrega Continua.
- Despliegue Continuo.

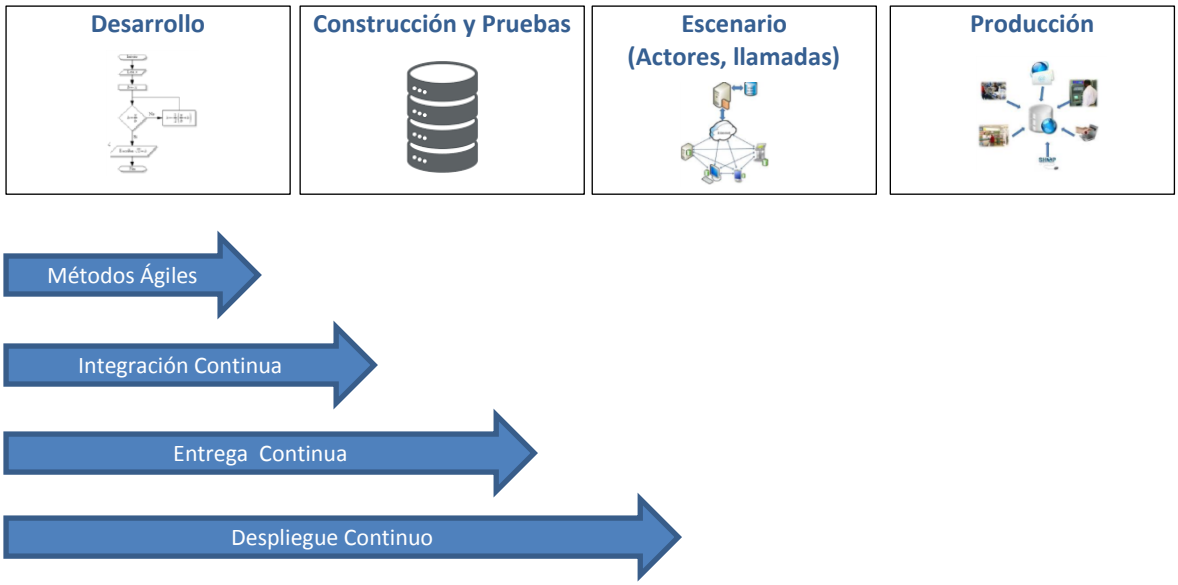

Figura 14: Procesos Continuos DevOps
Fuente: Autor

1.8 Cultura Devops

Como se mencionó en el capítulo anterior para implementar una cultura DevOps dentro de una empresa es importante considerar tres aspectos: personal, procesos y tecnología. Si estos aspectos no están alineados, posiblemente la iniciativa no alcanzará su potencial total.

1.8.1 Personal

Este punto abarca mucho más que solo las capacidades operacionales del equipo de TI. La importancia de este paso significa que debe tener una estrategia descendente con participación de la gerencia superior para que funcione, el propósito de todo esto es lograr que todos vayan en la misma dirección.

Aquí debemos considerar:

1. El diseño de la organización.
2. Las estructuras departamentales.
3. Los roles y las responsabilidades laborales.

4. Las líneas de generación de reportes.
5. La manera en que se incentiva a los empleados.

Esta cultura borra la línea entre las funciones del desarrollador y el personal de operaciones y permite que las prácticas de DevOps se difundan y sean adoptadas por la organización en general.

Una actitud de responsabilidad compartida es un aspecto de la cultura DevOps que fomenta una colaboración más estrecha. La colaboración a menudo comienza con una mayor conciencia de los desarrolladores de las preocupaciones operativas (como el despliegue y el monitoreo) y la adopción de nuevas herramientas y prácticas de automatización por parte del personal de operaciones.

Encontrar los perfiles idóneos se ha convertido en todo un reto para la mayoría de las empresas. Para lo cual hay que tener claras las habilidades que se requieren y se debe analizar si la organización de TI cuenta con el talento de DevOps necesario disponible o caso contrario saber dónde lo puede encontrar. Si lo buscará en el mercado o capacitará al personal existente.

El objetivo más importante para un equipo DevOps es que funcione como un centro de excelencia de colaboración, comunicación e integración.

Si la organización ya tiene implementadas las prácticas agiles como el marco de escala ágil (SAFe) o Scrum se puede aprovechar para la adopción de la Cultura DevOps.

Según el Informe de Estado de DevOps en el 2017 realizado por (Puppet, 2017b) se pudo observar que la demanda por personas conocedoras de DevOps está creciendo rápidamente, ya que las empresas que lo aplican obtienen excelentes resultados: logran implementar código con una frecuencia 30 veces mayor que sus competidores y sus fracasos de implementación son 50% menores.

1.8.2 Procesos

Los procesos definen lo que la gente hace. Una organización puede tener una gran cultura de colaboración, pero si la gente está haciendo las cosas mal o haciendo las cosas correctas en el camino equivocado van directo al fracaso.

Según (Menzel, 2015) en una cultura DevOps los procesos deben estar alineados junto con el conocimiento y las responsabilidades en las personas correctas.

En este punto se debe considerar la organización y sus procesos empresariales, desde la identificación y la priorización de requisitos hasta los lanzamientos de software y los procesos operativos, como la administración de cambios e incidentes.

La gestión del cambio debe incluir procesos que presenten las siguientes capacidades:

- Administrar elementos de trabajo.
- Configuración del flujo de los elementos de trabajo.
- Gestión de la configuración del proyecto.
- Planificación (ágil e iterativa).
- Control de acceso a artefactos basado en roles.

Estos procesos gestionan los elementos de trabajo para todos los proyectos, tareas y activos asociados, y no sólo a los afectados por solicitudes de cambio o errores. Aquí se puede ampliar el concepto de DevOps como un proceso de negocio: una colección de actividades o tareas que producen un resultado específico (servicio o producto) para los clientes.

Un aspecto importante también es entender que cambios serían necesarios hacer en la organización al aplicar Devops con las metodologías agiles y los procesos continuos a su ciclo de vida.

1.8.3 Tecnología

En DevOps 2.0 Toolkit de (Farcic, 2017) nos dice que para lograr una transformación empresarial a DevOps con éxito se debe tener en cuenta la estructura tecnológica que maneja la organización, considerando que las aplicaciones actuales que se encuentran en producción suelen ser monolíticas y se las debe desglosar para que sean más fáciles de lanzar, administrar, mantener y cambiar. A veces los desarrolladores deben esperar que equipos de infraestructura muy fragmentada en silos realicen cambios en los servidores.

La automatización de tareas como pruebas, configuración y despliegue libera a la gente para centrarse en otras actividades valiosas y reduce la posibilidad de error humano.

Gracias a que es automático el proceso de organizar, probar y desplegar código desde herramientas de integración, según las recomendaciones dadas en NIST Cloud Computing Reference Architecture por (Liu et al., 2011) la infraestructura nube (cloud) puede seguir los ritmos que requieren las aplicaciones en el mercado actual.

La infraestructura como código según (Swartout, 2014) es una capacidad básica de DevOps que permite a las organizaciones gestionar la escala y la velocidad, qué entornos necesitan ser aprovisionados y configurados para permitir la entrega continua.

Se dispone de tres tipos de herramientas de automatización para trabajar con infraestructura como código según nos indica (Sharma & Coyne, 2015):

1. **Herramientas centradas en aplicaciones o middleware:** Estas herramientas por lo general son capaces de administrar como código aplicación de servidores y las aplicaciones que se ejecutan en ellos. Estas herramientas pueden incluir bibliotecas de tareas de automatización para las tecnologías que soportan. No pueden realizar tareas de bajo nivel, como configurar un Sistema Operativo, pero pueden automatizar completamente un servidor y tareas a nivel de aplicación.

2. **Herramientas de despliegue y medio ambiente:** Son una nueva clase de herramientas que tienen la capacidad de desplegar las configuraciones de la infraestructura y el código de la aplicación.

3. **Herramientas genéricas:** Estas herramientas no están especializadas para ninguna tecnología y se pueden programar para realizar varios tipos de tareas, desde la configuración de un sistema operativo hasta una configuración de puertos de firewall.

1.9 Etapas del Pipeline de DevOps

En el pipeline de DevOps se presentan etapas que van desde el ambiente de Desarrollo hasta el ambiente de Producción, en la Figura 15 se muestran cada una de las etapas. Esto puede variar de acuerdo a la organización, pero se trata de presentar un estándar que se puede aplicar y modificar en base a las necesidades de las organizaciones y sus procesos.

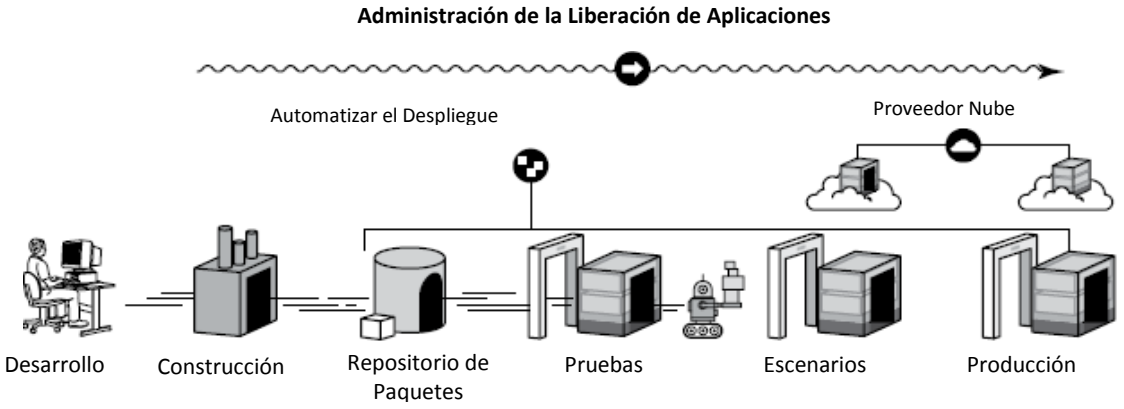

Figura 15: Etapas del Pipeline de DevOps
Fuente: Adaptado de (Sharma & Coyne, 2015)

A continuación se detallan cada uno de los ambientes del pipeline:

Desarrollo

Este ambiente provee herramientas a los desarrolladores para escribir y probar el código. Las herramientas para la gestión de control de fuentes, de colaboración, pruebas unitarias y planificación.

Construcción

En esta etapa se compila el código, se crea y se prueba por unidad lo que se va a desplegar. Las herramientas de compilación que se usan aquí varían de acuerdo a la plataforma que tienen las organizaciones. Por lo general utilizan servidores de compilación para facilitar la demanda de compilaciones que requiere una integración continua.

Repositorio de Paquetes

Conocido también como repositorio de archivos o artefactos permite almacenar el código creado en la etapa de Construcción. También almacenan los archivos asociados requeridos para facilitar el despliegue, tales como los archivos de configuración, de infraestructura y secuencias de comandos de implementación.

Pruebas

En esta etapa los grupos de Control de Calidad, usuarios de aceptación hacen las pruebas reales. Las herramientas que se utilizan aquí son las requeridas por el equipo de Control de Calidad tales como: gestión del entorno de prueba, gestión de datos de prueba.

Escenario y Producción

Aquí es donde se despliegan las aplicaciones. Las herramientas de gestión y monitoreo son las que se utilizan en esta etapa que permitan a las organizaciones supervisar los despliegues realizados en producción.

CAPÍTULO II
IMPACTO DE LOS MICROSERVICIOS Y CONTENEDORES

Hablar del movimiento DevOps con las prácticas de metodologías ágiles nos conduce a los microservicios, aunque no es un requisito obligatorio si facilita y ayuda en el momento de su adopción.

En este capítulo se presentan las características de la arquitectura de microservicios, su composición y funcionalidades. Según (Newman, 2015) la forma de interactuar de los microservicios a través de los mensajes HTTP usando una API REST. Como las operaciones CREATE, READ, UPDATE y DELETE son mapeadas en esta arquitectura. Los despliegues se pueden realizar a través máquinas virtuales o de contenedores, en este capítulo nos vamos a centrar en los Contenedores.

Las prácticas de Entrega Continua o un Despliegue Continuo se agilitan con la aplicación de los microservicios y su despliegue en contenedores. Como un contenedor permite hacer las pruebas en entornos cerrados ya que se si presenta un fallo solo se volverá a desplegar en el contenedor afectado. También se presenta un apartado mostrando los beneficios que representa el despliegue de los microservicios en contenedores, como se reduce el impacto cuando se produce un fallo.

2.1 Arquitectura de Microservicios

(Bernal, 2016) menciona que es un estilo de arquitectura que está compuesto por uno o más servicios que se despliegan independientemente y cada uno se enfoca en realizar una tarea única. Un servicio se puede construir utilizando cualquier lenguaje, tecnología e infraestructura.

Los servicios por lo general exponen sus funcionalidades a través de una API REST y el intercambio de mensajes HTTP para interactuar.

2.1.1 Características

- Son pequeños ya que se centran solo en una unidad de trabajo, no existen reglas sobre el tamaño de un microservicio pero se recomienda que sean suficientemente pequeños para poder reescribirlos y mantenerlos con facilidad.

- Un microservicio también necesita ser tratado como una aplicación o un producto, estos deben tener su propio repositorio de administración de código fuente y su propia canalización de entrega para las compilaciones y el despliegue.

- El acoplamiento holgado es una característica absolutamente esencial de los microservicios. El despliegue se lo hace por su propia cuenta, no existe ninguna coordinación con otro microservicio, esto hace que su implementación sea más rápida.

- Tienen un contexto limitado, un microservicio no sabe nada sobre la implementación de otros microservicios.

- Los microservicios se componen juntos para formar una aplicación compleja no necesitan ser escritos usando el mismo lenguaje de programación.

- La comunicación con microservicios es a través de lenguaje neutral para APIs, típicamente Protocolo de Transferencia de Hipertexto (HTTP) basado en recursos API como REST.

2.1.2 REST (Representational State Transfer)

(Newman, 2015) menciona a la *Transferencia de Estado Representacional* más conocido por sus siglas en inglés como REST, como un estilo arquitectónico generalmente usado en las aplicaciones web para permitir que sus recursos se comuniquen entre sí. REST no depende de ningún protocolo pero la mayoría de sus servicios utilizan HTTP como protocolo subyacente.

En este punto solo vamos a mencionar aquellos principios que son necesarios para aplicar el API REST en la integración de los microservicios.

RESTful se llama así a los servicios que son basados en REST. La API RESTful se centra en los roles y recursos de los componentes no toma en cuenta los detalles internos de la implementación. Ver Figura 16.

Para consumir recursos se debe hacer uso de un estándar de formatos de texto de los cuales citaremos a XML Y JSON para los servicios que funcionan a través de HTTP. La

elección de uno de ellos depende de las habilidades del desarrollador y de las herramientas que desee en su API. XML posee una gran variedad de herramientas disponibles a diferencia de los demás.

Otro factor importante a la hora de implementar es conocer a detalle como las operaciones CREATE, READ, UPDATE y DELETE son mapeadas con el respectivo POST de las acciones HTTP: GET, PUT y DELETE.

Como vemos es un requisito primordial conocer todo sobre HTTP ya es uno de los principios básicos para construir APIs REST.

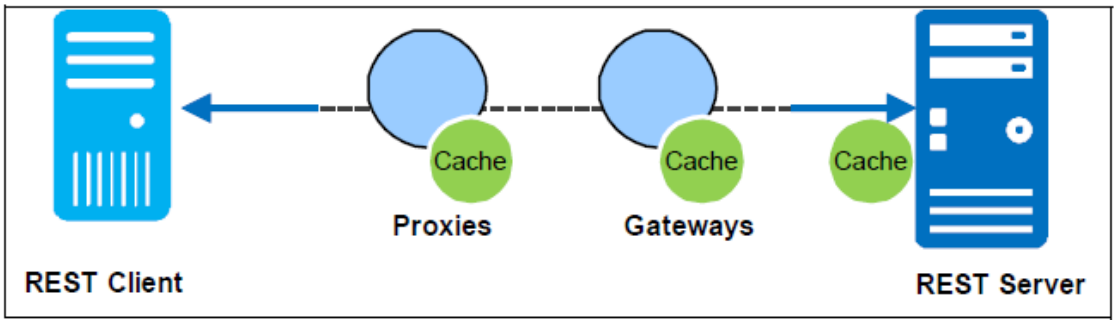

Figura 16: Cachés intermedios en una Arquitectura REST
Fuente: Adaptado de (IBM Corp., 2015)

2.2 Despliegue de Microservicios en Contenedores

Diseñar una arquitectura de microservicios abarca también en pensar en un modelo de despliegue, es decir la forma en que se va a organizar y gestionar el despliegue de cada microservicio.

Los despliegues se pueden realizar a través máquinas virtuales o de contenedores, en este trabajo nos vamos a centrar en los Contenedores (ver Figura 17). Un contenedor permite hacer las pruebas en entornos cerrados ya que se si presenta un fallo solo se volverá a desplegar en el contenedor afectado.

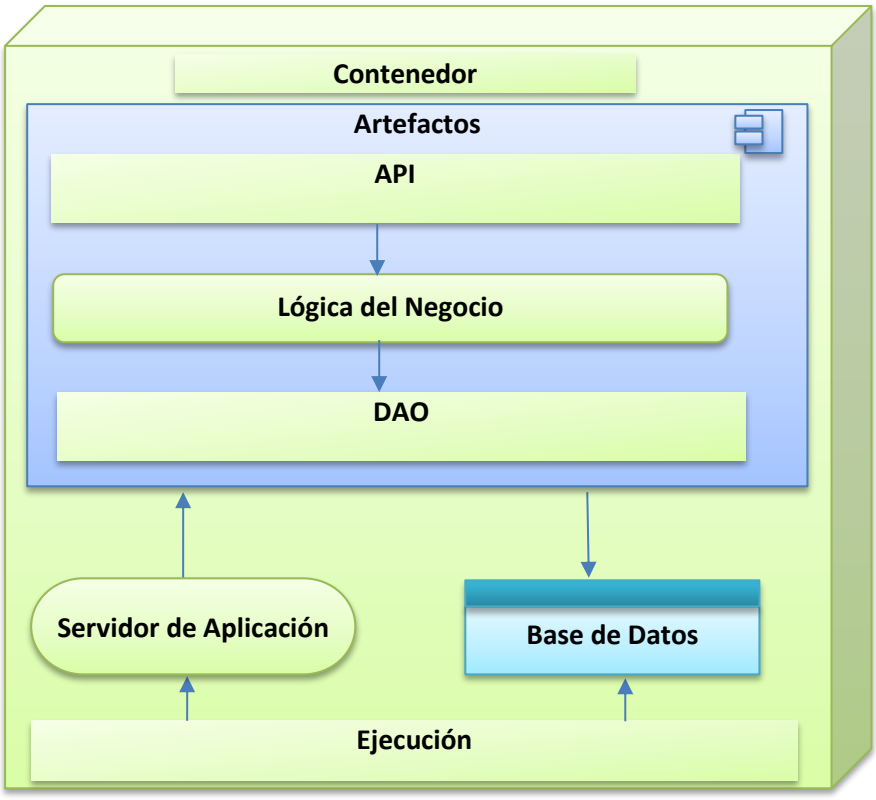

Figura 17: Despliegue de Microservicio en un Contenedor
Fuente: Adaptado de (Farcic, 2017)

Por eso uno de los requisitos de esta arquitectura es tener un principio de entrega continua de la cual se hablará más adelante en el capítulo de los Devops. Esto debido a la cantidad de microservicios que existan los cuales es necesario desplegarlos con frecuencia.

Una vez que ya hemos terminado con el o los cambios requeridos en nuestro microservicio llega la hora de hacer las pruebas y realizar el despliegue. Eso sí con mucha tranquilidad ya

que si se presentan errores no presentará inestabilidad para el sistema ya que fueron diseñados para el fracaso.

2.3 Contenedores

Según de (de la Torre, 2016) los contenedores son máquinas virtuales ligeras y portables que consumen menos recursos de cómputo que las máquinas virtuales tradicionales.

Ofrecen un enfoque diferente donde sus componentes son ligeros y permiten a las aplicaciones moverse entre las nubes con mayor agilidad. Con el uso de contenedores podemos hacer actualizaciones sin afectar a la aplicación en general.

2.3.1 Componentes

Los microservicios se pueden escribir en lenguajes de programación diferentes, se pueden alojar en un servidor de aplicaciones distinto o usar diferentes conjuntos de bibliotecas. Si cada servicio se encuentra dentro de un contenedor su aplicación será mucho más fácil ya que lo único que se hará es escoger el tipo de contenedor y este contendrá todo lo necesario.

Los contenedores contienen todo lo que se necesita para el despliegue de un microservicio, son inmutables y se ejecutan en procesos aislados. Entre sus principales componentes tenemos:

Tabla 5: Componentes de un Contenedor

Componentes	Puede ser
Bibliotecas de tiempo en ejecución	JDK, Python, o la requerida para que corra
Servidor de Aplicaciones	Tomcat, nginx, etc.
Base de Datos	MySql o la más ligera
Artefactos	JAR, WAR

Fuente: Adaptado de (Farcic, 2017)

2.3.2 Herramientas

Tomar una decisión acerca de la herramienta idónea para usar con los contenedores no es una tarea fácil ya que en el mercado se presentan varias opciones entre las cuales presentamos:

- Windows Server 2016 Container.
- Docker.
- Amazon EC2 Container Service (ECS)
- Kubernetes
- Openshift Container Platform 3.4
- Spring Cloud
- Netflix OSS

2.4 Beneficios de la integración de Microservicios y Contenedores en las aplicaciones

Entre las principales razones mencionadas por (IBM Corp., 2015) para el uso de una arquitectura de microservicios con contenedores encontramos:

2.4.1 Implementación Independiente

Uno de los beneficios de usar microservicios y contenedores dentro de un ambiente DevOps es la independencia en la implementación de los servicios. El caso de que un componente presente problemas en la aplicación no significa que todo el sistema se caerá.

Se puede utilizar diferentes tecnologías dentro de cada microservicio. Con esto podemos elegir la herramienta más adecuada para realizar cada trabajo.

Adoptar una nueva tecnología resultará más fácil, debido a que las pruebas se harían primero sobre el microservicio con riego más bajo con esto reducimos el nivel de impacto si llega a resultar mal.

2.4.2 Cambios pequeños, impactos pequeños

Tener un cambio pequeño significa que el impacto también será pequeño, reduciendo así los riesgos y ofreciendo más oportunidades de cambio.

En la Figura 18 se muestra la diferencia de trabajar con grandes cambios y pequeñas porciones de cambios. Por lo general los grandes cambios tendrán cientos de líneas de código desarrolladas de forma aislada que luego serán reunidas y probadas quizás a última hora, lo que posiblemente traerá muchas complicaciones.

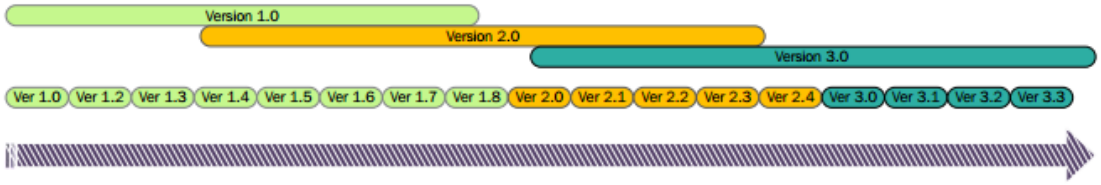

Figura 18: Grandes lanzamientos versus pequeños lanzamientos incrementales
Fuente: Adaptado de (Swartout, 2014)

Al contrario de trabajar con pequeñas porciones de cambios se tendrán pocas líneas de código, menos sobrecarga y una reducción de los problemas de implementar a última hora. Tener Entrega Continua y DevOps fusionados significa trabajar de este modo.

2.4.3 Despliegue, rollback y aislamiento de fallos

El despliegue es mucho más rápido y fácil con los microservicios. Implementar algo pequeño es siempre más rápido (si no más fácil) que desplegar algo grande. En caso de que nos damos cuenta de que hay un problema, ese problema tiene un efecto potencialmente limitado y se puede revertir mucho más fácil. Hasta que retrocedamos, la falla se aísla a una pequeña parte del sistema. La entrega o el despliegue continuo se pueden hacer con la velocidad y las frecuencias que no serían posibles con las aplicaciones grandes.

Los cambios se hacen en un solo servicio y el despliegue se hace de forma independiente del resto del sistema, con esto obtenemos una mayor rapidez y una reducción de los problemas ya que se puede aislar más rápido un servicio individual.

2.4.4 Escalabilidad

Escalabilidad, presentan una mayor eficiencia y rentabilidad para aquellos entornos en donde se deben desplegar una gran carga de trabajo. Se puede escalar solo aquellas piezas que lo necesiten.

CAPÍTULO III
ESQUEMA DE INTEGRACIÓN ENTRE DEVOPS, MICROSERVICIOS Y CONTENEDORES

Fusionar DevOps con microservicios y contenedores en la actualidad ya es un hecho, e incluso se recomienda su integración para una mayor efectividad de las tareas a realizarse dentro de un entorno DevOps. Las metodologías ágiles se están convirtiendo en una base fundamental para las organizaciones, por ello usar una de estas metodologías desde la fase de Planificación permite tener un ciclo de retroalimentación más rápido debido a su orquestación con las prácticas continuas; con esto el equipo de desarrollo puede presentar sus trabajos o entregables en cada finalización de un Sprint.

Según (Farcic, 2017) en las prácticas de Integración Continua, Entrega Continua y Despliegue Continuo el objetivo principal es que cada Sprint esté compuesto por las fases de Codificación, Construcción, Pruebas y Despliegue. La mayoría de los equipos empiezan con una Integración Continua y avanzan lentamente hacia la entrega y el despliegue, ya que los anteriores son prerrequisitos posteriores, además (Farcic, 2017) indica que *" el despliegue continuo, microservicios y contenedores es como mencionar a los tres mosqueteros, cada uno capaz de grandes hechos, pero al unirse, son capaces de mucho más."*

Con el Despliegue Continuo, se proporciona retroalimentación continua y automática de la disponibilidad de aplicaciones y el despliegue en producción, mejorando así la calidad de lo que entregamos y disminuyendo el tiempo para llegar al mercado.

Aplicando esos conceptos se puede continuar con la orquestación de una arquitectura de microservicios y el uso de contenedores dentro del despliegue continuo.

En la Figura 19 se muestra un esquema con el modelo de integración de DevOps aplicando una arquitectura de microservicios con contenedores. Se presentan los diferentes ambientes de interacción que tiene DevOps y cómo estos se convierten en un soporte para las organizaciones cuyo propósito es reducir los costos, evitar riesgos, mejorar la calidad y aumentar la velocidad en la entrega de las aplicaciones, reduciendo también los tiempos de retroalimentación del producto.

3.1 Esquema de integración

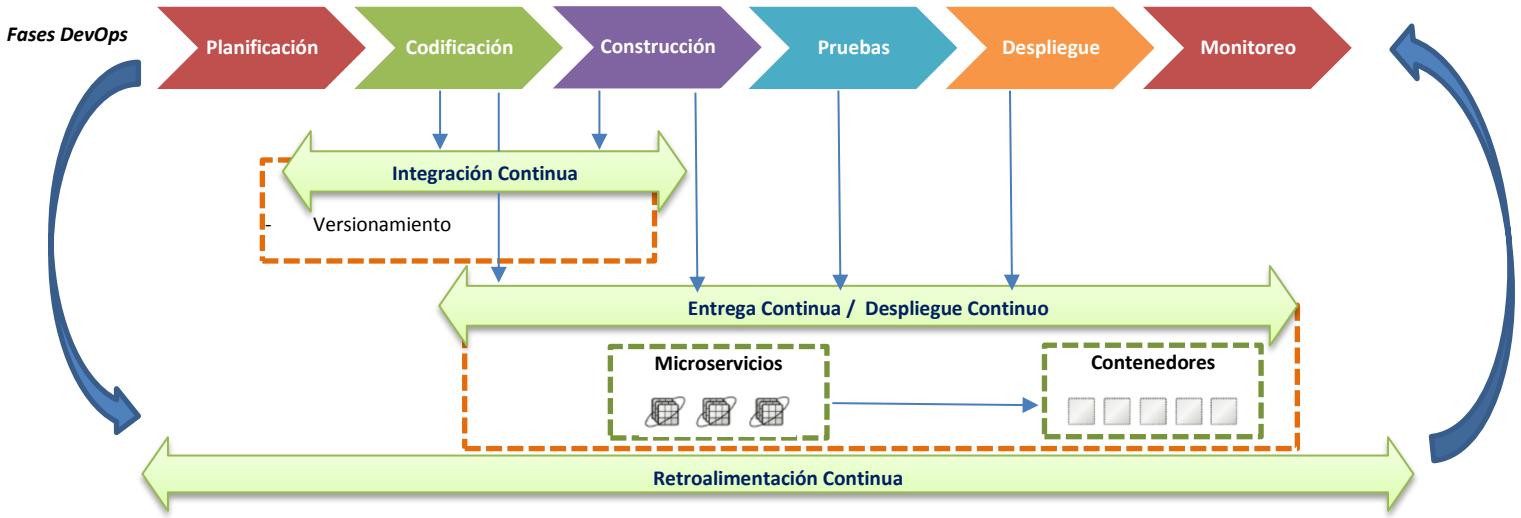

Figura 19: Esquema de Integración entre DevOps, Microservicios y Contenedores
Fuente: Autor

3.2 Fases de DevOps

Cada fase debe contribuir no solo al área de tecnología sino a la organización en general para el alcance de los objetivos con respecto a la entrega de software y lograr un equilibrio entre el costo, velocidad y calidad de las aplicaciones. Se presentan las fases según (Kort, 2016):

Planificación.

En la fase de *Planificación* es donde se recolectan todos los requerimientos, aquí se realizan la creación de las tareas y todas las actividades del proyecto.

Codificación.

La fase de *Codificación* se da con el desarrollo o actualización del código de acuerdo a la tarea planificada. Hay que considerar las herramientas que ayudan a mejorar el rendimiento de esta fase en donde se debe tener un eficaz control de versiones.

Construcción.

En esta fase se realiza la *Construcción* del código con su respectiva compilación, se realiza el análisis estático del código y empieza la ejecución de las pruebas unitarias en caso de presentarse algún error se procede a notificar los fallos a los responsables respectivos.

Pruebas.

En la fase de *Pruebas* el código se revisa antes del despliegue en Producción. Automatizar las pruebas implica el uso de herramientas adecuadas que faciliten realizar las pruebas por separado para la creación de las secuencias de los comandos que se ejecuten continuamente sin intervención manual alguna.

Despliegue.

En la fase de *Despliegue* se configuran las herramientas para que el despliegue de las aplicaciones sea en forma automatizada. En este trabajo se dará a través del despliegue de los microservicios en los contenedores Docker.

Monitoreo.

En la fase de *Monitoreo* se supervisa el funcionamiento de las aplicaciones desplegadas, esto ayuda a detectar posibles fallos que no se hayan controlado en las fases anteriores. Con un continuo seguimiento se puede medir la disponibilidad y el rendimiento del producto brindando así una mayor estabilidad del ambiente en tiempo real. Identificar cada una de las fases de DevOps es esencial para adoptar la cultura en las organizaciones.

3.3 Scrum como Metodología ágil

(Satpathy, 2016) al momento de dar inicio a la creación de un nuevo proyecto indica que hay que escoger el proceso sobre el cual se va a basar; en este caso los procesos seleccionados son de la metodología Scrum. Para familiarizarse con Scrum lo primero que se debe conocer es la terminología necesaria para trabajar con el esquema presentado.

Asignar bloques de tiempo es una de las prácticas que se debe considerar en los procesos de un proyecto Scrum. En la Figura 20 se presenta la duración de las reuniones de Scrum por bloques de tiempo.

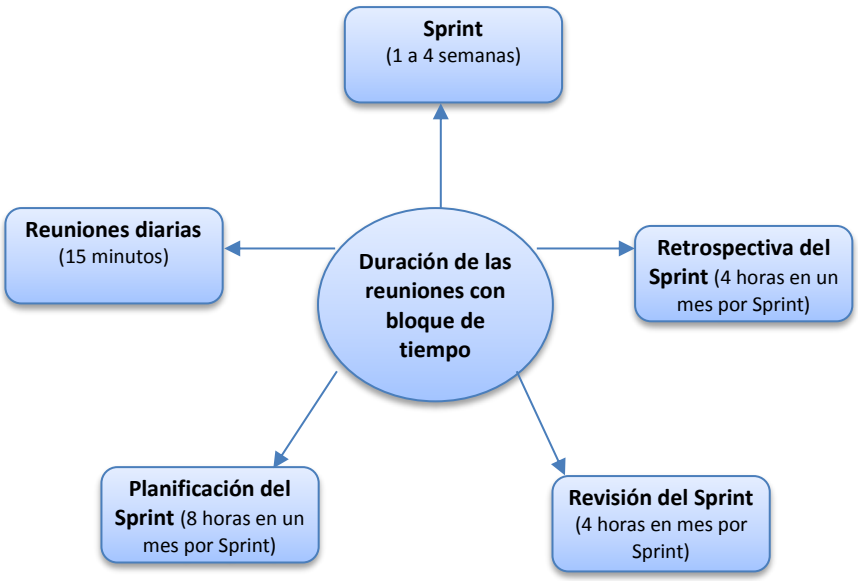

Figura 20: Duración de las reuniones con bloques de tiempo en Scrum
Fuente: Adaptado de (Satpathy, 2016)

Los eventos de interacción que se presentan en el esquema son:

- Sprints
- Backlog de Producto
- Tareas
- Reuniones Diarias (Daily Scrum)

Sprint

Un Sprint es un bloque de tiempo, se lo considera como la base de un proyecto Scrum. Tiene una duración de una a cuatro semanas. Se enlaza con la Reunión Inicial, las Reuniones Diarias (Daily Scrum), la revisión de Sprint y la retrospectiva de Sprint. En la Figura 21 se puede ver el flujo que tiene un Sprint en un proyecto.

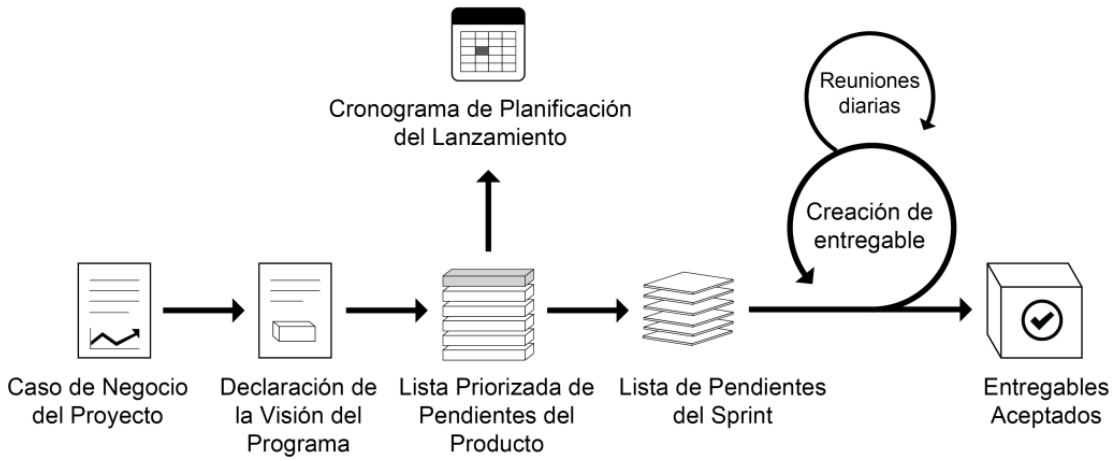

Figura 21: Flujo de Scrum para un Sprint
Fuente: Adaptado de (Satpathy, 2016)

Aquí el Scrum Master se convierte en una guía importante, el cual da protección al equipo Scrum en todo el proceso de la creación de los entregables.

Backlog de Producto

Es una lista de trabajo ordenada donde se presentan los requisitos para los cambios en el producto. El responsable del contenido, ordenación y disponibilidad de esta lista es el Dueño del Producto.

Tareas

Las tareas por lo general nacen en la reunión inicial aunque también se pueden crear durante el transcurso del proyecto. Son ejecutadas durante un sprint.

Equipo Scrum

Es importante identificar a todas las partes del equipo que están involucradas en los procesos. Entre ellos tenemos:

- Dueño del Producto
- Scrum Master
- Equipo de desarrollo
- Tester
- Administrador de Sistema

Dueño del Producto

Es el propietario del producto, es quien decide las características y funcionalidades del proyecto. Se encarga de mantener comunicados a todos los participantes del proyecto.

Scrum Master

Considerado también como el líder del proyecto, se encarga de que se cumplan todos los procesos Scrum. Ayuda en la coordinación y organización de las tareas y los procesos.

Equipo de Desarrollo

Dentro del equipo de desarrollo se incluyen al arquitecto, los desarrolladores, los testers, los administradores de base de datos y administradores del sistema. Se los conoce como el equipo de personas encargados del diseño, construcción y pruebas del producto.

3.4 Integración Continua

Es una práctica del movimiento ágil donde los desarrolladores integran de forma continua su trabajo con el de otros miembros del equipo de desarrollo para luego realizar pruebas de todo el trabajo integrado.

Fue mencionado por (Fowler, 2014) como: *"Práctica de desarrollo de software en donde los miembros del equipo integran su trabajo con frecuencia por lo menos una vez por día. Estas integraciones son verificadas con un build automático incluyendo las pruebas donde se pueden detectar los errores de forma más rápida".*

La integración regular permite descubrir tempranamente los riesgos a los que se exponen los sistemas que pueden ser tanto técnicos como los relacionados con el horario donde los largos periodos de tiempo en el proceso de desarrollo hacen que la aplicación se encuentre en un estado no disponible ya que nadie está interesado en ejecutar toda la aplicación hasta que esté terminada. Los desarrolladores prueban los cambios e incluso pueden ejecutar pruebas unitarias, pero nadie está tratando de iniciar la aplicación y utilizarla en un entorno de tipo producción.

Existen proyectos que utilizan ramas de larga vida o posponen las pruebas hasta el final del desarrollo y una vez que tienen todas las fusiones allí empezar. Muchas veces se dan cuenta de que su software no es apto o no cumple con los requerimientos esperados.

La integración continua representa un cambio de paradigma en el desarrollo, de no existir una continua integración su software debe esperar hasta la etapa de integración para empezar las pruebas de funcionamiento, de presentarse fallos atrasaría en gran manera el proyecto hasta identificar el cambio que lo ocasionó.

Con la integración continua se demuestra que el software funciona correctamente (suponiendo un conjunto suficientemente completo de pruebas automatizadas) con cada cambio. Se conoce el momento exacto en que falla y de esta forma arreglarlo de manera inmediata.

Los equipos que utilizan la integración continua de manera efectiva presentan las aplicaciones más rápido y con menos errores que los equipos que no lo hacen. Los errores

se capturan mucho antes del proceso de entrega y es más simple su arreglo, lo que representa un ahorro significativo tanto en costo como en tiempo. Por lo que se considera como una práctica esencial para los equipos profesionales de desarrollo de software.

Se destaca la importancia de que los equipos usen el control de versiones como una parte fundamental dentro de sus procesos de desarrollo.

3.4.1 Requerimientos para implementar Integración Continua

Hay tres requisitos primordiales que son necesarios para implementar una CI en un equipo de desarrollo que menciona (Davis & Daniels, 2016):

3.4.1.1. Control de Versiones

Se recomienda el uso de un único repositorio para el control de versiones en donde se debe incluir código, scripts de base de datos, pruebas, despliegues y archivos necesarios para crear, instalar, ejecutar y probar las aplicaciones.

No importa el tamaño del proyecto no se puede justificar el no usarlo, no existe un proyecto lo suficientemente pequeño como para prescindir de él.

Existen varias herramientas para el control de versiones, una pueden ser livianas, potentes y hasta libres de licencias.

También conocidos como sistemas de control de fuentes es un mecanismo para mantener múltiples versiones de las aplicaciones.

Su objetivo es mantener y proporcionar el acceso a todas versiones de cada archivo también permitir a los equipos ser distribuidos a través del tiempo y el espacio para una efectiva colaboración.

Si el código se encuentra bajo el control de código fuente y está versionado significa que está disponible para los miembros del equipo en cualquier momento de forma segura.

Para un uso eficaz del sistema de control de versiones se recomiendan los siguientes puntos:

1. Administrar Dependencias
2. Administrar Librerías Externas
3. Administrar Componentes
4. Administrar Configuración de Software
5. Administrar Configuración de Aplicaciones

3.4.1.2. Estructura Automatizada

Debe ser posible para una persona o un equipo ejecutar la construcción, prueba y el proceso de implementación de forma automatizada a través de la línea de comandos. Esto significa que se puede iniciar la compilación desde la línea de comandos.

Se puede comenzar con un programa de línea de comandos que le dice a su IDE que construya la aplicación y luego ejecute las pruebas o puede ser una colección de scripts de compilación de varios niveles.

Se deben crear scripts que se pueden ejecutar a través de la línea de comandos sin su IDE, puede no tener sentido pero existen varias razones para esto:

- Debe ser capaz de ejecutar el proceso de generación automáticamente desde el entorno de integración continua para facilitar la auditoria cuando las cosas van mal.

- Los scripts de compilación deben ser tratados como código, los cuales deberían ser revisados y probados constantemente.

- Facilita el mantenimiento y la depuración de la compilación y permite una mejor colaboración con la gente de operaciones.

3.4.1.3. Acuerdo del equipo

Como la integración continua es una práctica no una herramienta se necesita de un alto grado de compromiso entre los miembros del equipo de desarrollo. En la Figura 22 se presenta un modelo de colaboración en equipo.

Si los miembros no asumen con responsabilidad la adopción de esta disciplina todos los intentos de integración serán en vano y no se presentarán las mejoras de calidad esperadas.

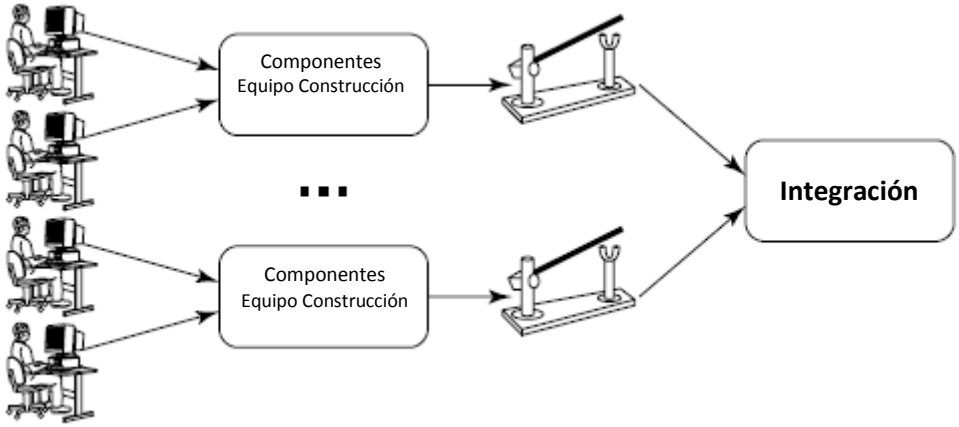

Figura 22: Colaboración vía Integración Continua
Fuente: Adaptado de (Sharma & Coyne, 2015)

Esta práctica permite que grandes organizaciones puedan entregar nuevas versiones de sus aplicaciones de forma rápida y confiable, obteniendo así un retorno mucho más rápido de sus inversiones y con riesgos muy reducidos.

3.5 Entrega Continua

Según (Humble & Farley, 2011) lo define como una práctica que consiste en el proceso de liberar software con frecuencia a través del uso de pruebas automatizadas e integración continúa.

Automatizar los procesos de desarrollo, las pruebas y las liberaciones tiene un alto impacto en la calidad y el costo de las aplicaciones.

3.5.1 Procesos de la Entrega Continua

En la Figura 23, se muestran los procesos que intervienen en la Entrega Continua, como se puede ver tener un repositorio para almacenar el código es una de las partes más importantes en esta práctica ya que los desarrolladores necesitan tener su código disponible todo el tiempo.

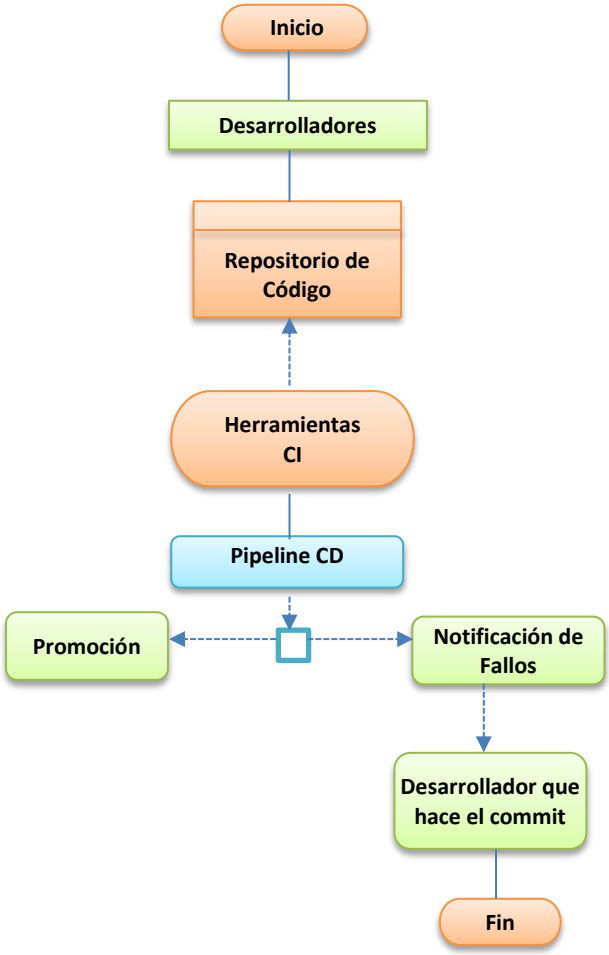

Figura 23: Procesos de la Entrega Continua
Fuente: Adaptado de (Farcic, 2017)

Las herramientas usadas en la Integración Continua para el monitoreo del código se pueden usar para el pipeline de esta práctica. El pipeline usa diferentes ambientes para sus tareas con el objetivo de verificar que el código cumpla con lo esperado. El paso final en este pipeline es el despliegue en el ambiente de producción, siempre y cuando no presente fallos ya que de presentarse deberá ser notificado.

En caso de que un paso falle el proceso se aborta y se notifican los fallos; y son enviados al desarrollador y las partes interesadas.

3.6 Despliegue Continuo.

Es el proceso de implementar los cambios en producción, en donde los despliegues de las aplicaciones han pasado las pruebas y validaciones respectivas reduciendo los niveles de riesgos. Esto permite que el producto sea entregado al cliente de la forma más rápida.

Puede combinarse con la liberación mediante un proceso automatizado que lanza una nueva versión a un pequeño grupo de usuarios y una vez verificado que no existen fallos lanzarlo a producción en forma generalizada.

Estos cambios de software más rápidos representan un alto impacto en la efectividad y en la satisfacción laboral y un mayor rendimiento de los desarrolladores.

3.6.1 Pipeline de Despliegue Continuo

Despliega automáticamente todas las compilaciones que pasaron todas las verificaciones. (Farcic, 2017) lo trata como un proceso automatizado que empieza con el commit al repositorio de código y finaliza con la aplicación o el servicio que se implementa en producción.

Hay casos donde los paquetes se implementan en el servidor de control de calidad antes de ser implementados en producción y las pruebas posteriores al despliegue se realizan dos veces (o tantas veces como el número de servidores de implementación).

Dependiendo del resultado de las pruebas posteriores a la implementación, se podría optar por revertir o habilitar la liberación al público.

Hay que prestar especial atención a las bases de datos (especialmente cuando son relacionales) y asegurar que los cambios que estamos haciendo de una versión a otra son compatibles con versiones anteriores y puedan funcionar en ambas versiones.

La entrega continua y el despliegue continuo tienen pruebas de producción como una necesidad absoluta. Se deben ejecutar pruebas que demuestren que el software desplegado está integrado con el resto del sistema.

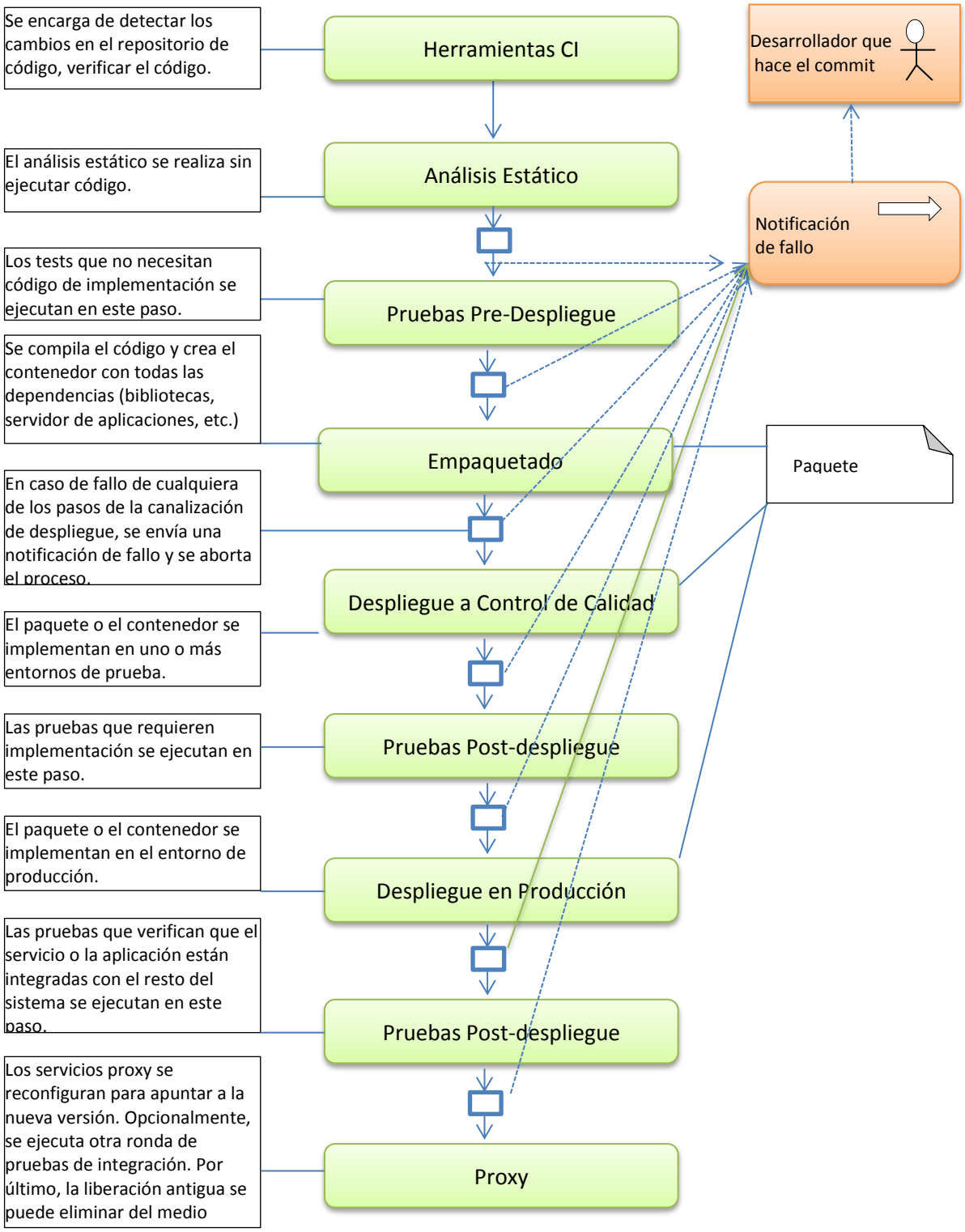

Figura 24: Pipeline de Despliegue Continuo
Fuente: Adaptado de (Farcic, 2017)

En la Figura 24, hay que destacar que las fases del pipeline se realizan en orden particular. Ese orden no sólo es lógico (por ejemplo, no podemos desplegar antes de compilar) sino también en orden del tiempo de ejecución. Las cosas que toman menos tiempo para correr se ejecutan primero. Por ejemplo, como regla general, las pruebas previas al despliegue tienden a ejecutarse mucho más rápido que las que ejecutaremos después del despliegue. El empaquetado termina con los contenedores inmutables, la implementación en un entorno de prueba puede no ser necesaria en absoluto.

Otra técnica muy útil en el contexto del despliegue continuo es la característica de conmutación. Dado que cada compilación se despliega en producción, podemos usarla para deshabilitar temporalmente algunas funciones.

3.6.2 Entrega Continua-Despliegue Continuo, Microservicios y Contenedores

La Entrega Continua (CI), el despliegue continuo (CD), los microservicios y los contenedores podrían parecer no estar relacionados, puesto que el movimiento de DevOps no estipula que los microservicios sean necesarios para el despliegue continuo, ni los microservicios necesitan ser desplegados en contenedores según lo indica (Farcic, 2017).

Pero cuando se combinan se abren nuevos caminos esperando que pasemos a través de ellos. Desarrollos recientes sobre contenedores y el concepto de despliegues inmutables permiten superar muchos de los problemas que los microservicios tenían antes. En la Figura 25 se puede ver el flujo que tiene la integración de Entrega Continua con Despliegue Continuo y los contenedores.

Los microservicios ofrecen mayor libertad para tomar decisiones, un desarrollo más rápido y una mayor escalabilidad de nuestros servicios, los procesos de entrega continua se agilizan con el uso de los microservicios ya que estos cuentan con un ciclo de vida propio.

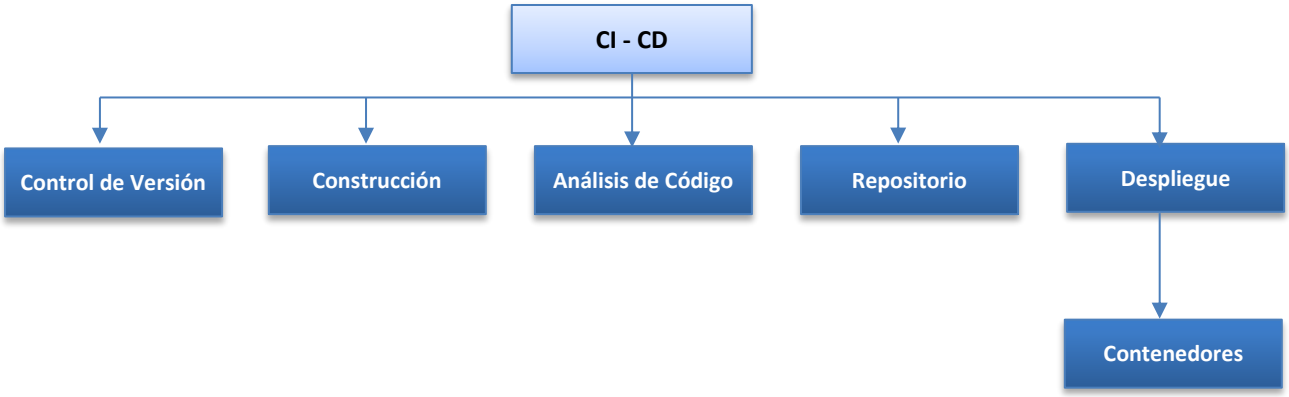

Figura 25: Flujo de CI/CD
Fuente: (Farcic, 2017). Adaptado por Ivonne Karina Farías Alejandro

Se puede combinar todo eso y hacer mucho más, como desplegar con más frecuencia y rápido, ser totalmente automático, lograr cero tiempos de inactividad, tener la capacidad de reversar, proporcionar fiabilidad constante en los entornos, ser capaz de escalar sin mayor esfuerzo.

En la Figura 26 se muestra el flujo que pasa un microservicio desplegado en Docker aplicando las prácticas de Entrega Continua y Despliegue Continuo.

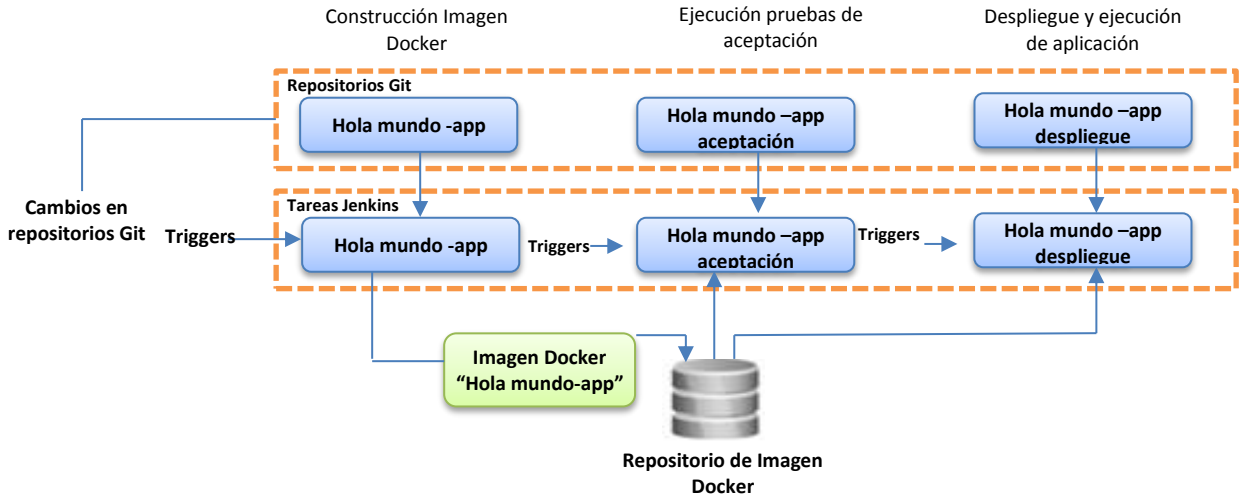

Figura 26: Flujo de CI/CD con Microservicios y Docker
Fuente: Adaptado de (Hauer, 2015)

3.7 Pruebas Continuas

Según (Carrizo, Cucu, Modir, & García, 2015) las pruebas no son otra cosa que la automatización de la ejecución de las pruebas. Aquí cada compilación es probada continuamente, permitiendo al equipo de desarrollo tener una retroalimentación en el menor tiempo posible evitando así futuros problemas antes de pasar a la siguiente etapa de desarrollo de software, con esto el flujo de trabajo se acelera ya que no hay necesidad de realizar todas las pruebas otra vez.

Para la adopción de una cultura DevOps las pruebas continuas es más que una función de control de calidad, es un trabajo integrado que empieza en el momento en que los desarrolladores construyen el código.

Tabla 6: Clases de Pruebas

Tipos de Pruebas	Detalles
Pruebas Unitarias	Prueban una parte, componente o una característica del producto.
Pruebas de humo (Smoke Test)	Es un análisis de bajo nivel que asegura que todos los componentes estén integrados sin problemas.
Pruebas de regresión	Es un proceso que se usa para analizar el comportamiento de un determinado producto cuando se modificado o añadido un nuevo código.
Pruebas funcionales	Se usan para asegurar la correcta ejecución del software, por lo general se aplican después de las pruebas unitarias. Y son realizadas por personal preparado los cuales usan casos de pruebas.

Fuente: (Carrizo et al., 2015), elaborado por Ivonne Karina Farías Alejandro

Si la funcionalidad del producto final presenta problemas las organizaciones pagarían un alto precio al no cumplir con las expectativas de sus clientes, generando pérdidas financieras muy graves.

El impacto de los fallos y el alto costo derivados de esto no es una opción que las organizaciones consideran en su flujo de trabajo.

3.8 Monitoreo Continuo

(Kort, 2016) menciona que el monitoreo constante contribuye a la reducción del impacto que pueden ser ocasionados por los cambios, una supervisión de manera frecuente asegura el buen rendimiento y la disponibilidad de las aplicaciones. Existen dos tipos de monitoreos en DevOps: la de servidores y la del rendimiento de las aplicaciones.

El monitoreo continuo tiene tres disciplinas operacionales:

- Auditoria Continua
- Monitoreo de Controles Continuos
- Inspección Continua de las Transacciones

Con un continuo seguimiento se puede medir la disponibilidad y el rendimiento del producto brindando así una mayor estabilidad del ambiente en tiempo real. Igual que las Pruebas Continuas el monitoreo comienza en la etapa de desarrollo.

3.8.1 Retroalimentación Continua

La retroalimentación Continua se encarga de supervisar el uso de las aplicaciones y proveer todos los datos necesarios de las métricas para el personal interesado.

Entre las actividades mencionadas por (IBM Corp., 2015) tenemos:

- Definir la hipótesis y los hitos de aprendizaje.
- Construir un producto viable.

- Implementar un entorno de pruebas.
- Validar las hipótesis.
- Gestionar el rendimiento.
- Promover la aplicación de la premisa.

3.9 Caso de Estudio

Para complementar el capítulo se va a presentar un caso de estudio donde la aplicación de las prácticas continuas y una cultura DevOps se encuentran implementadas.

Este caso es tomado del área de Gestión de Identidades y Accesos de la Tecnología de la información aplicado por (Harvard University, 2014).

Con la entrega continua se pudo automatizar todo el sistema de entrega, desde la creación y la implementación en desarrollo para las pruebas y la liberación en producción siguiendo los pasos del pipeline.

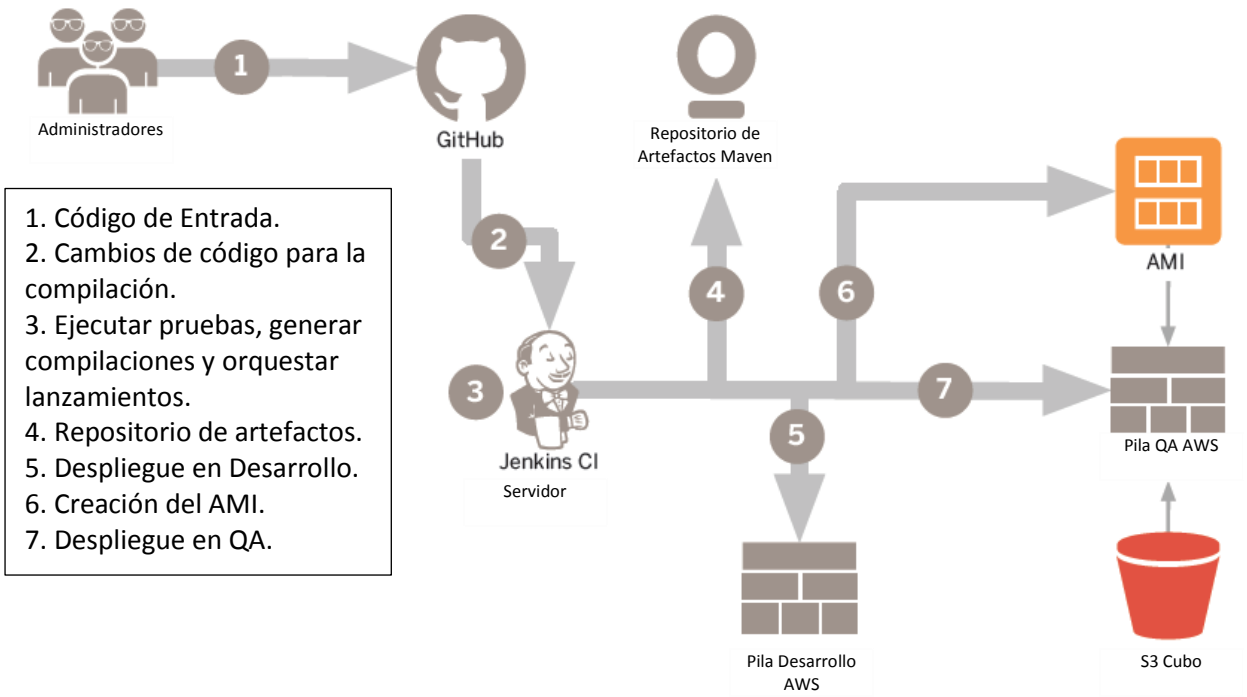

Figura 27: Caso de Estudio. Flujo de Entrega Continua.
Fuente: Adaptado de (Harvard University, 2014)

El servidor de integración continua (CI) Jenkins extrae el código al repositorio de origen y luego se realizan los cambios en el entorno de desarrollo. Cuando se completa el hito en desarrollo, el servidor de CI crea un AWS (Web Services de Amazon) de AMI (Amazon Machine Imagen) basado en un trigger predefinido. Ver Figura 27.

Este AMI es utilizado en todos los entornos superiores con configuración específica de los ambientes.

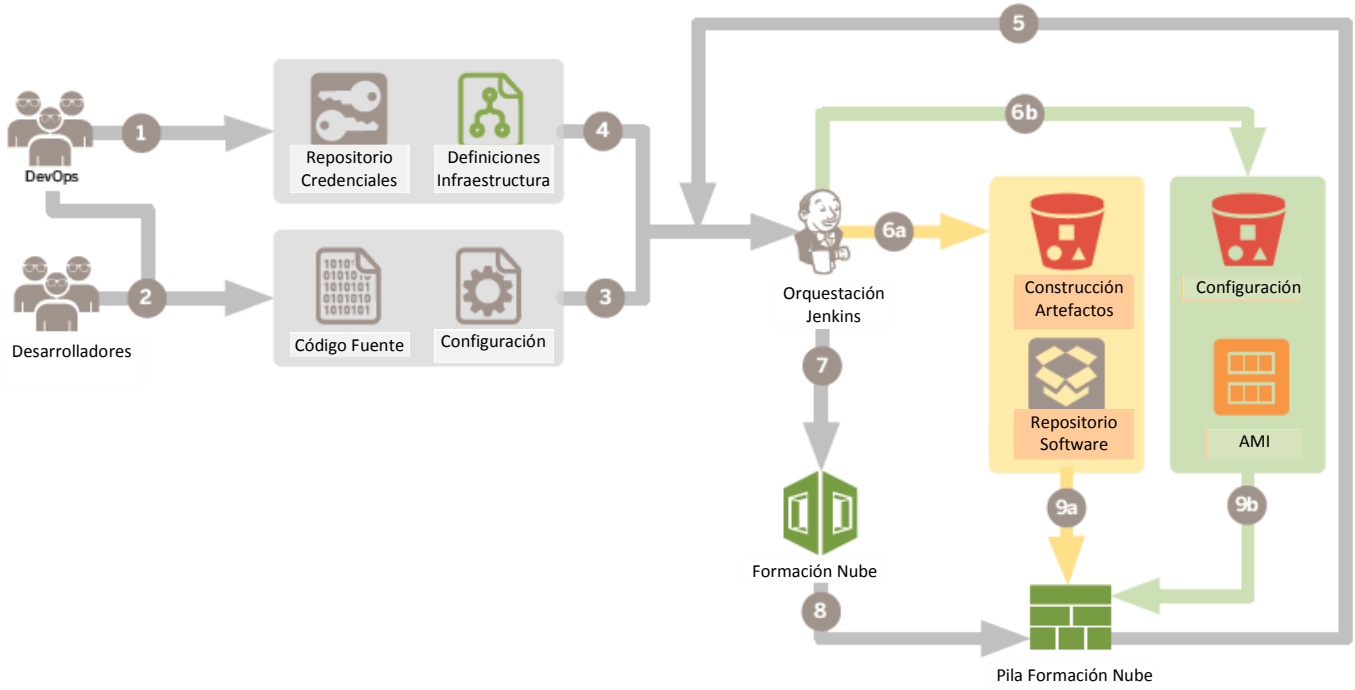

Figura 28: Caso de Estudio. Modelo de Referencia.
Fuente: Adaptado de (Harvard University, 2014)

1. Configuración de las credenciales y las definiciones de la infraestructura.
2. Confirmar el código fuente y configuración de los cambios.
3. Liberación del Git e inicio de la estructura del orquestador.
4. Configuración de las credenciales y definiciones de infraestructura en las plantillas de la nube.
5. Recuperar información sobre los recursos existentes en la nube para configurar la infraestructura.
6. a. Construcción de artefactos, configuraciones y software para almacenamiento.
   b. Agrupar y publicar AMI para el despliegue en producción.

7. El orquestador inicia con el despliegue.
8. Se crean o se actualiza en la pila de la nube.
9. a. Se liberan los artefactos, software y configuraciones.
   b. Se agrupa la imagen con el software desplegado.

Puntos claves del Despliegue en Desarrollo.

Según la publicación de (Harvard University, 2014) nos da como referencia los siguientes puntos claves aplicados para el despliegue en Desarrollo:

- La implementación es inicia con los cambios en los repositorios de control de código fuente de Git que luego son procesados por el orquestador.

- Los artefactos y configuraciones generados se almacenan en S3 como un archivo WAR.

- Las configuraciones se generan en base a fuentes externas, como Credentials Store (para almacenar contraseñas y certificados), un repositorio Git con configuraciones para los módulos de Puppet.

- La pila de aplicaciones se despliega por medio de una plantilla en la nube basada en artefactos y configuraciones.

- Una imagen de máquina de Amazon (AMI) se crea en el servidor de la aplicación construida.

- El AMI liberado se comparte con todas las cuentas AWS de IAM de Harvard y está listo para el proceso de liberación representado en la Figura 28.

Puntos clave del Proceso del Despliegue en Producción.

Según la publicación de (Harvard University, 2014) nos da como referencia los siguientes puntos claves aplicados para el despliegue en Producción:

- En el proceso de liberación de AMI, el orquestador recopila configuraciones del repositorio de credenciales y ejecuta la pila de la nube para la aplicación que se va a implementar.

- Una liberación se realiza actualizando los grupos de escalado automático con la nueva actualización de AMI y los trigger de la pila de la nube.

CAPÍTULO IV
HERRAMIENTAS REQUERIDAS PARA LA CONSTRUCCIÓN DEL AMBIENTE DE DESARROLLO

Para el proceso de adopción de una cultura DevOps es necesario considerar las herramientas adecuadas que ayuden a cumplir con los objetivos de la organización a la hora de presentar sus aplicaciones. La elección de las herramientas puede variar de acuerdo a los objetivos del negocio, sus intereses y prioridades.

Uno de los puntos principales es que las herramientas sean compatibles entre sí, o que tengan los plugins disponibles para su integración, esto facilitará el proceso de automatización de las tareas.

Otro punto importante es el soporte que tiene la herramienta elegida, ya que existen algunas que ofrecen una mayor documentación disponible en la web o como soporte técnico, ya que su nivel de accesibilidad para con los usuarios agiliza y garantiza un correcto funcionamiento y permite que las tareas sigan su flujo normal, optimizando el tiempo de respuesta.

La creación de procesos con las herramientas adecuadas permite tener condiciones altamente favorables para una metodología ágil con sus prácticas continuas y su integración con DevOps, microservicios y contenedores.

Abarcar todas las fases del ciclo de vida, desde la planificación hasta el monitoreo es de gran importancia ya que una acertada elección de sus herramientas puede garantizar el éxito o el fracaso de la adopción de la cultura DevOps.

En este capítulo se va a hacer una breve descripción de algunas herramientas que intervienen en estas actividades de acuerdo a los puntos descritos en el apartado de las consideraciones que se presentan a continuación.

Las herramientas que se mencionan varían de acuerdo a los tipos de licencia comercial y de código abierto.

.

4.1 Consideraciones

Para una mejor selección de las herramientas se recomienda considerar ciertos puntos que pueden contribuir para facilitar este proceso, tales como:

- Licencia: Si son Open Source, Comercial. El tipo de licencia que tiene GPL[2], BSD[3], EPL[4], etc.
- Tipo de lenguaje: La compatibilidad de los lenguajes ya no son un problema con DevOps y las prácticas continuas, pero es necesario conocer en qué tipo están desarrollados los pluggins por motivos de futuras integraciones y compatibilidad con otras herramientas.
- Compatibilidad: La compatibilidad con los diferentes sistemas y servidores es un punto importante.

4.2 Herramientas que abarcan el ciclo de vida de DevOps y los flujos de Integración-Entrega-Despliegue Continuo

Hay herramientas que contribuyen directamente con el flujo de las prácticas continuas CI-CD (Integración Continua-Despliegue Continuo), presentadas en la Figura 29, estas herramientas colaboran en conjunto con el ciclo de vida de DevOps para automatizar las tareas.

Antes de elegir una herramienta específica es importante conocer cuáles son las que se encuentran disponibles en el mercado y las que cumplen con los requisitos para una integración eficaz entre las diferentes fases.

[2] Licencia Pública General de GNU o GNU General Public License
[3] Berkeley Software Distribution
[4] Licencia Pública Eclipse

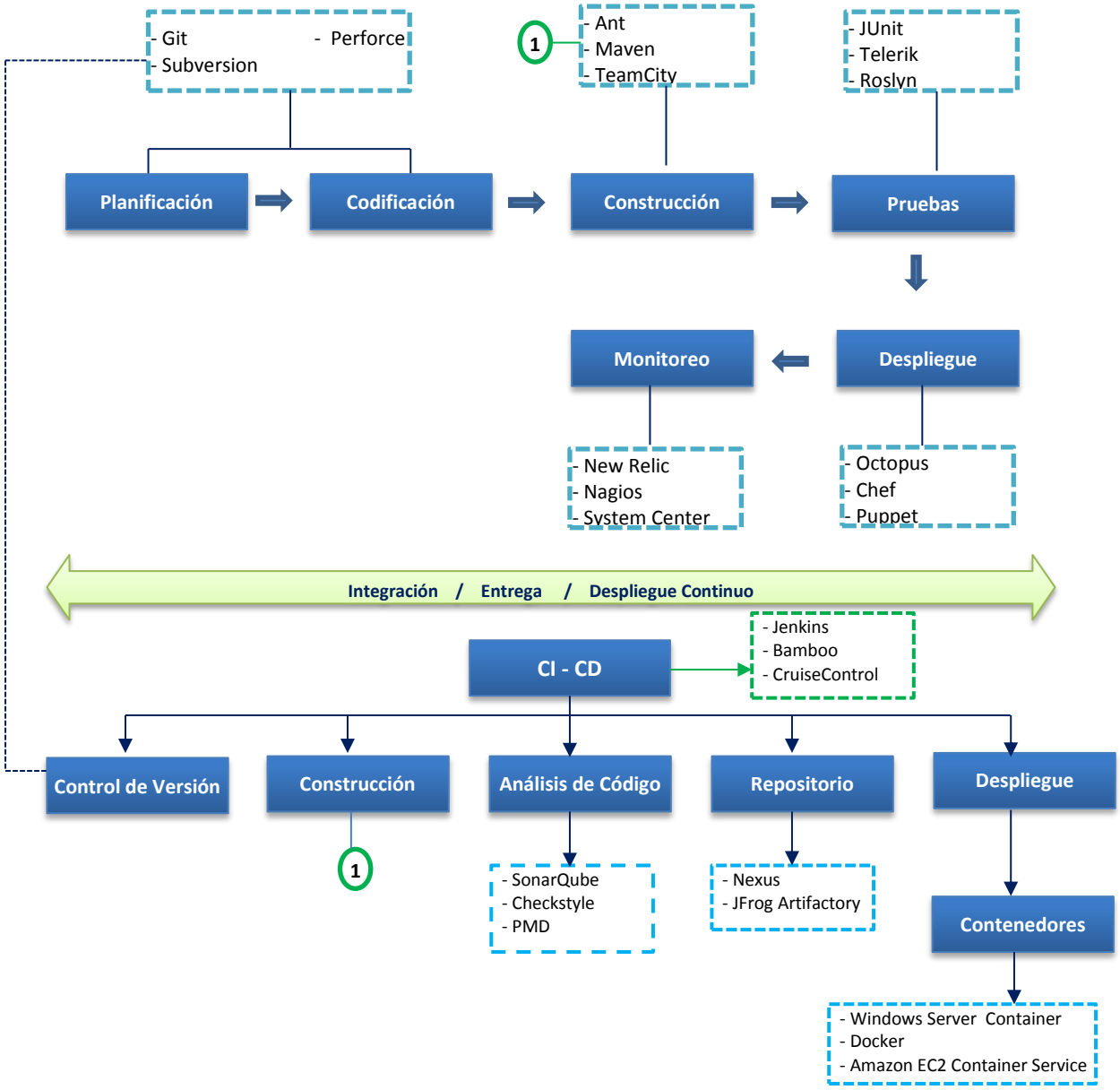

Figura 29. Herramientas que abarcan el ciclo de vida de DevOps y los flujos de Integración-Entrega-Despliegue Continuo
Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro, basado en (Farcic, 2017)

4.2.1 Planificación

(Kort, 2016) indica que en la fase de *Planificación* es donde se recolectan todos los requerimientos, aquí se realizan la creación de las tareas y todas las actividades del proyecto.

Todo lo que se realice aquí se convierte en un punto clave para dar inicio al ciclo de vida del proyecto. La aplicación de la metodología de desarrollo Scrum es la que se aplicará para la planificación de las actividades, lo que implica que las herramientas que se escojan deberán ser compatibles para integrarse con estos conceptos.

Las herramientas que intervienen tanto en las fases de Planificación y Codificación se describen en el siguiente apartado.

4.2.2 Codificación

La fase de *Codificación* se da con el desarrollo o actualización del código de acuerdo a la tarea planificada. Hay que considerar las herramientas que ayudan a mejorar el rendimiento de esta fase en donde se debe tener un eficaz control de versiones según fue mencionado por (Kort, 2016).

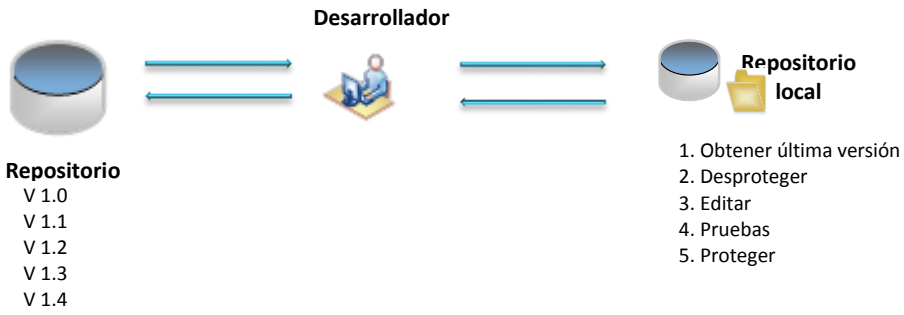

Figura 30. Flujo de Trabajo de un Sistema de Control de Versiones
Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.2.1 Control de Versiones

El Control de Versiones, se trata de un repositorio central el cual se encuentra compartido entre el equipo de los desarrolladores, en donde se registran todos los cambios realizados en un archivo.

Un sistema eficaz de control de versiones debe permitir administrar el código fuente, controlar los cambios, gestionar los elementos de trabajo, dar el respectivo seguimiento de los errores, permitir múltiples entregas de código, debe permitir revertir los cambios a versiones anteriores, hacer comparaciones entre los fuentes, tener un historial completo de los cambios y quien los realizó y ser compatibles con las diferentes herramientas para una correcta compilación. Ver Figura 30.

A continuación se describen brevemente las características de las herramientas mencionadas para este apartado.

Tabla 7: Herramientas fase de Codificación

Herramienta	Descripción	Características
Git	Es un Sistema Open Source para el Control de Versiones creado por Linux. Colabora directamente en la etapa de desarrollo para la gestión del código fuente ya que permite realizar un seguimiento de todos los cambios realizados. Con este sistema se asegura la integridad de los datos y es un apoyo para el flujo de trabajo.	- Permite el desarrollo no lineal. - Facilita el desarrollo distribuido. - Permite el manejar eficientemente proyectos de gran tamaño. - Cada historia cuenta con una autenticación criptográfica. - A nivel de Protocolos: HTTP (Protocolo de Transferencia de Hipertexto), FTP (Protocolo de Transferencia de Archivos). - A nivel de Repositorios es compatible con Subversion. - También cuenta con una emulación de Servidor CVS, lo que permite ser usado por clientes CVS y tiene complementos adicionales IDE que facilitan el acceso a los

		repositorios Git. - A nivel de Sistemas Operativos como: Windows, Solaris, MacOS, BSD. Entre los plugins disponibles tenemos: - JGit es una librería hecha en Java. Usada para las aplicaciones Java. Es usada por Gerrit y EGit para el IDE de Eclipse. - JS-Git hecho en JavaScript. - Dulwich Git hecho para Python. Maven JGit-Flow Plugin.
Subversion (SVN)	Es una herramienta para el control de versiones Open Source bajo licencia Apache/BSD. Todos los cambios realizados los guarda mediante el concepto de revisión, de esta forma solo guarda las últimas modificaciones. Puede ser usado en distintas máquinas por varios usuarios, los cuales pueden administrar y modificar los mismos datos desde sus respectivas ubicaciones.	- Es muy flexible, permite crear, copiar y borrar carpetas de forma similar al trabajo en un disco local. - Manejo eficaz de archivos binarios. - Se pueden seleccionar los archivos que se desean bloquear. Al integrarse con Apache se pueden usar todas las opciones de este en el momento de autenticar los archivos de tipo PAM, SQL, LDA, etc. Existen servicios para proyectos Open Source para proveer almacenamiento usando Subversion: SourceForge, Google Code, Project Kenai, CodePlex. Tiene pluggins para Jenkins, Jira,

		Eclipse.
Perforce	Es un Sistema de Control de Versiones con licencia comercial desarrollado por Perforce Software, Inc. Tiene una base de datos centralizada en donde se almacenan los repositorios conocidos como (depots) versionados donde se guardan los archivos. Su conexión es a través de TCP. Dependiendo de la configuración del Servidor los archivos de texto se pueden codificar en ASCII o Unicode.	- Soporte para archivos binarios, ASCII, Unicode. -Los trabajos pueden ser almacenados de forma temporal y se puede cambiar de tarea. - Se pueden comprimir los archivos para su almacenamiento y transmisión. - Se pueden replicar los archivos y los metadatos. - Se guarda un historial completo de las revisiones de los archivos modificados, eliminados copiados. - Tiene un gestor para implementar políticas y restricciones. - Permite gestionar archivos históricos para liberar espacio en el disco. - Posee envió de alertar mediante email o RSS. - El Servidor de Perforce es gratuito para 20 usuarios con un número ilimitado de archivos, de superarse los usuarios se debe comprar la licencia. - Con los plugins adecuados es compatible para Windows, Linux, Solaris, MacOS, y FreeBSD. Tiene plugins para Visual Studio, Eclipse, Jenkins, Jira.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas en la fase de Codificación

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 8: Cuadro Comparativo de Herramientas en la fase de Codificación

Criterios	Git	Subversion (SVN)	Perforce
Licencia	Open Source	Open Source	Comercial
Tipo de Licencia	GNU GPL v2	Apache/BSD	
Sistema Operativo	Windows, Solaris, Mac OS, BSD	Windows, Linux, Mac OS	Windows, Linux, Solaris, Mac OS, y FreeBSD.
Documentación disponible	web, libro	web, libro	web, libro
Pluggins disponibles	JGit, JS-Git, Dulwich Git, Maven JGit-Flow.	Para Jenkins, Jira y Eclipse	Para Visual Studio, Eclipse, Jenkins, Jira.
Lenguaje de desarrollo	C, Bourne Shell, Perl1	C	C++ y java

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.3 Construcción

Una vez que se ha descargado el código fuente se da inicio a la fase de *Construcción* del código con su respectiva compilación, se realiza el análisis estático del código y empieza la ejecución de las pruebas unitarias en caso de presentarse algún error se procede a notificar los fallos a los responsables respectivos.

Si las prueban han resultado satisfactorias se almacenan los fuentes en el respectivo repositorio. Ver Figura 31.

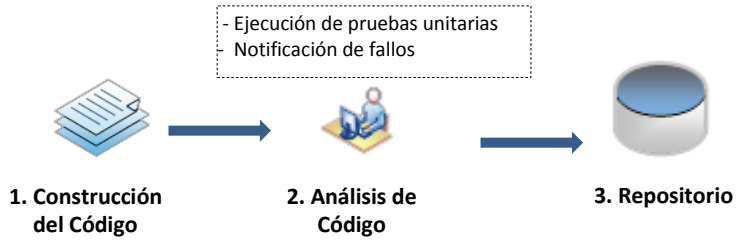

Figura 31. Flujo de Trabajo para la Construcción, Análisis de Código, Repositorio
Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

A continuación se describen las herramientas que intervienen en esta fase:

Tabla 9: Herramientas fase de Construcción

Herramienta	Descripción	Características
Ant	Apache Ant es un software Open Source para automatizar los procesos de compilación creado por Apache Software Foundation. Se usa para programar las tareas repetitivas o mecánicas durante la fase de construcción y de compilación.	- Está desarrollado en Java. Se basa en la configuración de archivos XML y clases Java para realizar las tareas, es ideal para las soluciones multiplataforma. - Tiene extensiones para las tareas de Perforce, Net, EJB, Git. Tiene plugins disponibles para Eclipse, JUnit.
Maven	Es una herramienta Open Source, creada por Apache Software Foundation para la gestión y la construcción de proyectos específicamente Java. Tiene funcionalidades similares a las de Ant, pero su modelo de	- Basa las construcciones de los proyectos en el modelo POM[5] (Project Object Model). - Puede trabajar en red. - Su estructura está basada en pluggins. - Maven es compatible con Netbeans, Git, Eclipse, IntelliJ

[5] POM, Modelo de Proyecto de Objeto. Es un archivo de tipo XML el cual tiene información del proyecto y registra los detalles de configuración necesarios para la construcción del proyecto.

	configuración es más simple, basado en formato XML. Permite compilar proyectos Java y ejecutar pruebas unitarias, también se pueden generar paquetes jar, war, ear.	IDEA, JDeveloper 11G (11.1.1.3) - Tiene plugins para Eclipse, JUnit, Subversion, Jira, Git.
TeamCity	Es una herramienta con licencia comercial, es un servidor de gestión e integración basado en Java. Colabora con la administración de la estructura del proyecto.	- Ayuda con la gestión del proyecto mediante la supervisión y la generación de informes estadísticos. - Es compatible con Amazon EC2 y ofrece cobertura con .Net y Java y Ruby. Se puede integrar con Eclipse, Visual Studio, IntelliJ IDEA. - Los Sistemas de Control de Versiones compatibles son: Git, Subversion. Team Foundation Server (2005, 2008, 2010), IBM Rational Clear Case. CVS. Borland StarTeam, Mercurial. - Tiene plugins para Gradle, Docker, Octopus, .Net, Artifactory.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas en la fase de Construcción

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 10: Cuadro Comparativo de Herramientas en la fase de Construcción

Criterios	Ant	Maven	TeamCity
Licencia	Open Source	Open Source	Comercial
Tipo de Licencia	Apache License 2.0	Licencia Apache 2.0	-
Sistema Operativo	Windows, Solaris, Unix, Mac OS, Linux	Windows, Solaris, Unix, Mac OS, Linux	Windows, Linux, Unix, Mac OS
Documentación disponible	web, libro	web, libro	web, libro
Pluggins disponibles	Para Eclipse, JUnit	Para Eclipse, JUnit, Subversion, Jira, Git.	Para Gradle, Docker, Octopus, .Net, Artifactory.
Lenguaje de desarrollo	java	java	java

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.3.1 Análisis de Código

Aunque el código compile sin problemas se pueden presentar errores en tiempo de ejecución no controlados o que no se ajustan de acuerdo a los procedimientos requeridos. Las herramientas en este punto deben permitir analizar el código antes de la compilación y así evitar los errores, deben permitir la creación y la configuración efectiva de las reglas necesarias para obtener todos los resultados esperados.

Tabla 11: Herramientas para el Análisis de Código

Herramienta	Descripción	Características
Checkstyle	Es una herramienta para el análisis de código que se usa en la fase de desarrollo. Sirve para la comprobación del código fuente de Java si cumplen con todas las reglas de codificación. Tiene licencia tipo GPL.	- Su construcción se basa en un archivo JAR el cual se puede ejecutar dentro de una máquina virtual de Java o una tarea de Ant. - Es compatible para Eclipse, Maven, Netbeans, Gradle, SonarQube. - Entre los plugins disponibles se encuentran Eclipse-CS, Checkstyle Beans, Maven Checkstyle Plugin, Gradle Checkstyle, SonarQube Checkstyle Plugin.
SonarQube	Es un software Open Source creado para evaluar el código fuente. Maneja estándares para la codificación.	- Mantiene un historial de las métricas. - Se pueden hacer pruebas unitarias. - Presenta informes sobre duplicación de código. - Con los pluggins adecuados es compatible con Maven, Ant, Atlassian, Bamboo, Jenkins, Hudson, Eclipse, IntelliJ IDEA, Visual Studio, Gradle, Jenkins, Hudson.
PMD	Es un software analizador de código fuente. Creado bajo licencia de BSD. Su trabajo consiste en usar un conjunto de reglas las cuales pueden ser personalizadas que permiten identificar los errores.	- Tiene un detector para código duplicado o muerto. - Soporta varios lenguajes entre ellos: PHP, Java, JavaScript, C, C ++, C#, PLSQL, Groovy, Ruby, Fortran, Python. Es compatible con Bamboo, Jenkins, Maven, Eclipse,

		JBuilder, NetBeans, JDeveloper. - Entre los pluggins disponibles se encuentran: Maven PMD plugin, Gradle: The PMD Plugin, Eclipse plugin, NetBeans plugin, JBuilder plugin, JDeveloper plugin, IntelliJ IDEA plugin.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para el Análisis de Código

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 12: Cuadro Comparativo de Herramientas Análisis de Código

Criterios	Checkstyle	SonarQube	PMD
Licencia	Open Source	Open Source	Comercial
Tipo de Licencia	GPL	GPL	BSD
Sistema Operativo	Windows, Mac OS, Linux	Windows, Mac OS, Linux	Windows, Linux, Mac OS
Documentación disponible	web	web, libro	web, libro
Pluggins disponibles	Para Eclipse-CS, Checkstyle Beans, Maven Checkstyle Plugin, Gradle Checkstyle, SonarQube Checkstyle Plugin.	Para Maven, Ant, Atlassian, Bamboo, Jenkins, Hudson, Eclipse, IntelliJ IDEA, Visual Studio, Gradle, Jenkins, Hudson.	Para Maven PMD plugin, Gradle: The PMD Plugin, Eclipse plugin, NetBeans plugin, JBuilder plugin, JDeveloper plugin, IntelliJ IDEA plugin.
Lenguaje de desarrollo	java	java	java

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.3.2 Repositorio

Los gestores de repositorios se convierten en una pieza clave para la práctica de integración continua. Las herramientas a utilizar deben contribuir para acelerar las compilaciones reduciendo el tiempo de búsqueda de las librerías.

No se debe confundir un gestor de repositorio con un sistema de control de versiones, no se pueden almacenar ambos por igual.

Los gestores de repositorios consisten en un repositorio interno para la gestión de librerías donde se guardan los artefactos de las aplicaciones que han sido desplegados en los diferentes ambientes (pruebas, pre-producción, producción).

A la hora de elegir la herramienta adecuada hay que considerar que sean compatibles con los servidores de integración continua que se han escogido para el desarrollo del proyecto.

A continuación se presentan las herramientas para este apartado:

Tabla 13: Herramientas para la Gestión de Repositorios

Herramienta	Descripción	Características
Nexus Repository OSS	Es un repositorio open Source creado por Sonatype bajo licencia EPL. Permite el almacenamiento en caché de repositorios remotos, da mayor estabilidad y control, gracias a que ofrece una disponibilidad del 100%.	-Facilita el intercambio de artefactos internos. - Soporta la colaboración entre los equipos mediante la distribución de componentes entre ellos. - Nexus OSS tiene componentes que son compatibles con Docker, java, Nuget, Ruby Maven. - Existen versiones de Nexus OSS compatibles con Mac OS x, Windows, Unix.
JFrog Artifactory	Artifactory de JFrog es una herramienta de repositorio, de licencia Propietaria. La cual ofrece apoyo en el ciclo de desarrollo	- Provee al equipo DevOps de las herramientas necesarias para la gestión eficiente del código desde

	como un gestor de repositorios binarios.	las máquinas de los desarrolladores hasta el ambiente de producción. - Permite la integración con herramientas de Integración Continua, como son, Jenkins, Hudson, TeamCity, Bambú, Team Foundation Server, Chef, Puppet, Ansible, Docker entre otras.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para la Gestión de Repositorios

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 14: Cuadro Comparativo de Herramientas Gestión de Repositorios

Criterios	Nexus Repository OSS	JFrog Artifactory
Licencia	Open Source	Comercial
Tipo de Licencia	EPL	-
Sistema Operativo	Mac OS, Windows, Unix.	Mac OS, Windows, Linux, Solaris
Documentación disponible	web, libro	web
Pluggins disponibles	Para Docker, java, Nuget, Ruby Maven.	Para Jenkins, Hudson, TeamCity, Bambú, Team Foundation Server, Chef, Puppet, Ansible, Docker.
Lenguaje de desarrollo	Java	C, C++

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.4 Pruebas

En la fase de *Pruebas* el código se revisa antes del despliegue en Producción. Automatizar las pruebas implica el uso de herramientas adecuadas que faciliten realizar las pruebas por separado para la creación de las secuencias de los comandos que se ejecuten continuamente sin intervención manual alguna.

Las pruebas requeridas para comprobar la funcionalidad dentro de en un entorno DevOps son según (Carrizo et al., 2015):

- Pruebas Unitarias
- Pruebas de humo (Smoke Test)
- Pruebas de regresión
- Pruebas funcionales

Las herramientas que intervienen en esta fase deben ser altamente compatibles con las herramientas de análisis de código. A continuación se describen las herramientas de esta fase:

Tabla 15: Herramientas para la fase de Pruebas

Herramienta	Descripción	Características
Junit	Es una herramienta Open Source con licencia EPL, que permite hacer pruebas unitarias para aplicaciones Java. Su framework cuenta con varias maneras de mostrar los resultados, una en modo de texto, una tarea en Ant o como un gráfico Swing[6] o AWT[7].	- Permite controlar pruebas de regresión, para controlar si el nuevo código cumple con los requerimientos pasados y que su funcionalidad no ha sido alterada después de haber sido modificados. - Evalúa los valores de retorno, si las clases cumplen con todas las especificaciones, enviando los respectivos mensajes como respuesta, si han tenido fallos o han sido exitosas. - Es compatible con Eclipse y

[6] Swing, biblioteca para interfaces gráficas de java
[7] Abstract Window Toolkit, librería para la construcción de interfaces gráficas

		NetBeans, ya que cuentan con plugins que tienen las plantillas para la creación automática de las pruebas por clases lo que ayuda al colaborador enfocarse solo en las pruebas y no en la creación de ellas.
Roslyn	Es un compilador Open Source de Visual Basic y C#. Posee una API de análisis de código que soportan estos lenguajes.	- En el compilador se puede comprobar la sintaxis, completar el código, se lo puede hacer desde el Visual Studio como de la línea de comandos. - Tiene un árbol de sintaxis propio. - Tiene símbolos para analizar la estructura de las clases sin necesidad de revisar el código a profundidad.
Test Studio-Telerik	Test Studio es una herramienta comercial creado por Progress.	- Permite implementar protocolos de prueba. - Está compuesto por un conjunto de pruebas automatizadas. - Test Studio es fácil de implementar ya que es compatible con lenguajes C# y VB.NET.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para la fase de Pruebas

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 16: Cuadro Comparativo de Herramientas para la fase de Pruebas

Criterios	Junit	Roslyn	Test Studio-Telerik
Licencia	Open Source	Open Source	Comercial
Tipo de Licencia	EPL	Apache	-
Sistema Operativo	Windows, Mac OS, Linux	Linux, Mac OS, and Windows	Windows, Linux, Mac OS
Documentación disponible	web, libro	web, libro	web, libro
Pluggins disponibles	Para Jenkins, Eclipse, GitHub	Para GitHub	Para Jenkins
Lenguaje de desarrollo	Java	Visual Basic y C#	ASP.NET AJAX

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.5 Despliegue

En la fase de *Despliegue* se configuran las herramientas para que el despliegue de las aplicaciones sea en forma automatizada. En este trabajo se dará a través del despliegue de los microservicios en los contenedores Docker.

Para lo cual se hace muy necesario en nuestro caso que las herramientas mencionadas en este apartado sean compatibles para realizar el despliegue con los contendedores Docker y a la vez también deben ser compatibles con las herramientas antes mencionadas para la Integración continua.

Tabla 17: Herramientas para la fase de Despliegue

Herramienta	Descripción	Características
Octopus	Es un servidor para automatizar despliegues. Dirigida para los desarrolladores en .Net. Su licencia es de tipo Propietaria.	- Provee de seguridad y confiabilidad en los entornos de prueba y producción para las aplicaciones Asp.net o servicios Windows. - Octopus se encarga de la distribución de las aplicaciones que se encuentran empaquetadas. - Permite la integración con TeamCity y con Microsoft Team Foundation Server.
Puppet	Es una herramienta Open Source escrita en Ruby para la gestión y configuración de código creada bajo licencia GNU (GPL).	- Está compuesto por un lenguaje declarativo que permite describir la configuración del sistema. - Puede ser distribuido por Linux, Ubuntu, Fedora, Mac Os X, Unix y Windows. - Permite que el camino para la adopción de los microservicios y contenedores sea mucho más fácil.
Chef	Es una herramienta escrita en Ruby y Erlang, creada para la gestión de la configuración de código. Tiene una licencia de tipo Apache.	- Asegura que los recursos se encuentren configurados de forma correcta. - Se puede ejecutar en modo cliente-servidor. - Permite el despliegue continuo permitiendo configurar pruebas, elaborar informes con mayor rapidez.

		- Se puede integrar con plataformas de la nube como OpenStack, Amazon EC2, Google Cloud Platform, Microsoft Azure, - Es compatible con Linux y Windows, Mac Os X, Solaris, Ubuntu.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para la fase de Despliegue

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 18: Cuadro Comparativo de Herramientas para la fase de Despliegue

Criterios	Octopus	Puppet	Chef
Licencia	Comercial	Open Source	Open Source
Tipo de Licencia	-	GNU (GPL)	Apache
Sistema Operativo	Linux, Windows, Mac OS	Linux, Solaris, Ubuntu, Fedora, Mac Os, Unix y Windows, BSD	Linux y Windows, Mac Os, Solaris, Ubuntu
Documentación disponible	web	web, libro	web, libro
Pluggins disponibles	Para TeamCity y Microsoft Team Foundation Server.	Nagios, New Relic	Nagios
Lenguaje de desarrollo	C, C++	Ruby	Ruby y Erlang

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.6 Monitoreo

En la fase de *Monitoreo* se supervisa el funcionamiento de las aplicaciones desplegadas, esto ayuda a detectar posibles fallos que no se hayan controlado en las fases anteriores.

Con un continuo seguimiento se puede medir la disponibilidad y el rendimiento del producto brindando así una mayor estabilidad del ambiente en tiempo real.

La herramienta a utilizar para esta fase debe permitir el monitoreo de los errores, notificar cuando se presentan los fallos, poder configurar las alertas necesarias y también debemos considerar que sea compatible con el lenguaje de programación usado para la construcción de las aplicaciones de los diferentes tipos, ya sean web o móviles, y que ofrezcan el soporte para los contenedores Docker que son con los que se trabaja en ese documento.

Tabla 19: Herramientas para la fase de Monitoreo

Herramienta	Descripción	Características
Nagios	Es una herramienta Open Source bajo licencia GNU, creado para el monitoreo y vigilancia.	- Es muy versátil a la hora de consultar y generar alertas las cuales pueden ser entregadas por diferentes medios, como SMS o correo electrónico. - Permite el monitoreo de los servicios de red como POP3, SMTP, HTTP, también permite el monitoreo remoto mediante SSH o SSL cifrados. - Se pueden desarrollar plugins en diferentes herramientas tales como Ruby, C++, Python.
New Relic	Es una herramienta para monitorizar las máquinas y ver en tiempo real los recursos disponibles. Su licencia es de tipo propietaria.	- Se pueden monitorear los procesos, red, la carga de las páginas, aplicaciones móviles, tanto a nivel de errores de datos, excepciones o por timeout. - Presenta informes estadísticos

		sobre el rendimiento de las aplicaciones, sobre la velocidad de la red, o el uso de la memoria. - Muy sencillo de instalar.
System Center	Es una herramienta creada por Microsoft. Conocida por sus siglas como SCOM (System Center Operations Manager).	- Permite la administración para sistemas operativos mediante una interfaz que presenta la información del rendimiento y estado del sistema. - Facilita la monitorización y la gestión de las alertas. - Permite la integración con otras herramientas mediante el uso de management packs (MP). - Con uso adecuado de MP se puede integrar con Linux, Tomcat.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para la fase de Monitoreo

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 20: Cuadro Comparativo de Herramientas para la fase de Monitoreo

Criterios	Nagios	New Relic	System Center
Licencia	Open Source	Comercial	Comercial
Tipo de Licencia	GNU	-	Microsoft EULA
Sistema Operativo	Linux, Mac OS, Windows	Unix, Linux, Mac OS, Solaris, Windows	Windows, Linux, Unix, Mac OS
Documentación disponible	web, libro	web	web, libro
Pluggins disponibles	Para Puppet, chef	Para Chef, Puppet	Para Puppet
Lenguaje de desarrollo	C y Perl	java	Microsoft .NET Framework 2.0.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

4.2.7 Servidores de Integración/Entrega Continua

Los servidores de Integración y Entrega Continua actúan como un enlace para todo el proceso del ciclo de vida de los DevOps, estos deben permitir realizar las operaciones del checkout, las compilaciones del código, las creaciones de los artefactos ya sean war, jar, ear, los despliegues en los contenedores y las ejecuciones de las pruebas y el monitoreo. Todo esto debidamente integrado con sus respectivas herramientas.

Las herramientas de Integración Continua deben ser compatibles con las herramientas mencionadas anteriormente para el Control de versiones, repositorios, pruebas, despliegue y monitoreo.

A continuación se describen las herramientas que intervienen en este apartado:

Tabla 21: Herramientas de CI/CD

Herramienta	Descripción	Características
Jenkins	Jenkins es una herramienta Open Source, basada en Java. Bajo licencia Creative Commons and MIT[8]. Colabora con las prácticas de Integración y Entrega Continua, permite automatizar todo tipo de tareas. Su plataforma es Servlet Container.	- Al estar basado en Java es fácil de instalar en Windows, Mac OS X, Unix. - Soporta herramientas de control de versiones como Subversion, Git, Mercurial, Perforce, Clearcase. - Se pueden ejecutar proyectos con base en Ant o Maven. - Tiene una amplia gama de plugins disponibles para su integración con otras herramientas, entre ellos encontramos: Javadoc, Mailer, Credentials, Ssh-slaves. - Las notificaciones se pueden dar en Twitter, Android, RSS, Google Calendar, E-mail, XMPP, IRC.

[8] MIT, Massachusetts Institute of Technology

		- Su IDE de integración: NetBeans, IntelliJ IDEA, Eclipse.
Bamboo	Es un software propietario creado por Atlassian. Es una herramienta que sirve como apoyo para las prácticas de integración, despliegue y entrega continua. Su plataforma es Servlet Container.	- Permite crear planes para realizar las compilaciones, se puede configurar los activadores que dan inicio a las compilaciones. - Se pueden automatizar las pruebas y configurarlas en forma paralela, lo que agilita la detección de errores. - Permite la integración con Bitbucket, HipChat, Jira, Fisheyey, AWS CodeDeploy y Docker. - Las notificaciones se pueden dar en XMPP, RSS, E-mail, Remote API, Google Talk. - Su IDE de integración: Visual Studio, Eclipse, IntelliJ IDEA.
CruiseControl	Es una herramienta Open Source de integración continua basada en Java, también permite compilar y ejecutar tests.	- Permite la integración con proyectos de otras herramientas. - Para el control de versiones se puede integrar con: Subversion, Git, TFS, Clearcase, Perforce, CVS. - Para los constructores permite la integración con Maven, Rake, Ant.

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para la CI/CD

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 22: Cuadro Comparativo de Herramientas para la CI/CD

Criterios	Jenkins	Bamboo	CruiseControl
Licencia	Open Source	Comercial	Open Source
Tipo de Licencia	Creative Commons and MIT	-	BSD
Sistema Operativo	Windows, Mac OS, Unix, Linux	Windows, Mac OS, Linux	Windows, Linux, Mac OS
Documentación disponible	web, libro	web	web
Pluggins disponibles	Para Subversion, Git, Mercurial, Perforce, Clearcase, Ant, Maven.	Para Bitbucket, HipChat, Jira, Fisheyey, AWS CodeDeploy y Docker.	Para Maven, Rake, Ant, Subversion, Git, TFS, Clearcase, Perforce, CVS
Lenguaje de desarrollo	java	java	java

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Cuadro Comparativo de Herramientas para Contenedores

Para una mejor elección de las herramientas se presenta un cuadro comparativo con los criterios más relevantes para la toma de decisiones.

Tabla 23: Cuadro Comparativo de Herramientas para Contenedores

Criterios	Windows Server Container	Docker	Amazon EC2 Container Service
Licencia	Comercial	Open Source	Comercial
Tipo de Licencia	-	Apache License 2.0	-
Sistema Operativo	Windows, Linux, Mac OS	Windows, Mac OS, Linux	Windows, Linux, Ubuntu, Mac OS

Documentación disponible	web, libro	web, libro	web, libro
Pluggins disponibles	Para Jenkins	Para Jenkins, Bamboo, Cruise Control, Ansible, chef, Puppet, Salt.	Para Jenkins, GitHub
Lenguaje de desarrollo	.Net Core	Go	Java, php, python, ruby, .net

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

CAPÍTULO V

MODELO PARA LA CONSTRUCCIÓN DEL AMBIENTE DE DESARROLLO

Luego de identificar las herramientas disponibles para usar en las distintas fases del ciclo de vida de DevOps el último paso es la selección de las más adecuadas e idóneas para definir nuestro modelo de desarrollo.

Siendo la automatización de los procesos es una de las tareas principales de una cultura DevOps las herramientas a utilizarse deberán contribuir con la gestión y la construcción de los proyectos de una manera más sencilla y ágil.

Las herramientas adecuadas van a permitir mejorar la calidad del producto, permiten acelerar los procesos y las tareas, facilitan la implementación de las pruebas continuas.

Para llegar a la elección de las herramientas que se van a recomendar en los dos modelos propuestos a continuación, se tomaron en cuenta los puntos antes descritos en el capítulo anterior donde se mencionan las consideraciones a la hora de elegir una determinada herramienta.

Se debe tener en cuenta el tipo de licencia, el tipo de lenguaje, su compatibilidad entre las herramientas que intervienen a lo largo de los procesos de las fases, que puedan ofrecer los plugins necesarios a la hora de integrarse con las otras herramientas y el soporte adecuado que se encuentre a disposición de las organizaciones.

A continuación se presentan dos modelos, el primero basado en licencias Open Source y el segundo basado en licencias Propietarias.

5.1 Modelo basado en Herramientas con licencias Open Source

En la Figura 32 se presenta un modelo basado en herramientas con una licencia tipo Open Source. Para la elección de las herramientas en este modelo se elaboró una matriz de evaluación la cual se encuentra en el Anexo 1 de este documento.

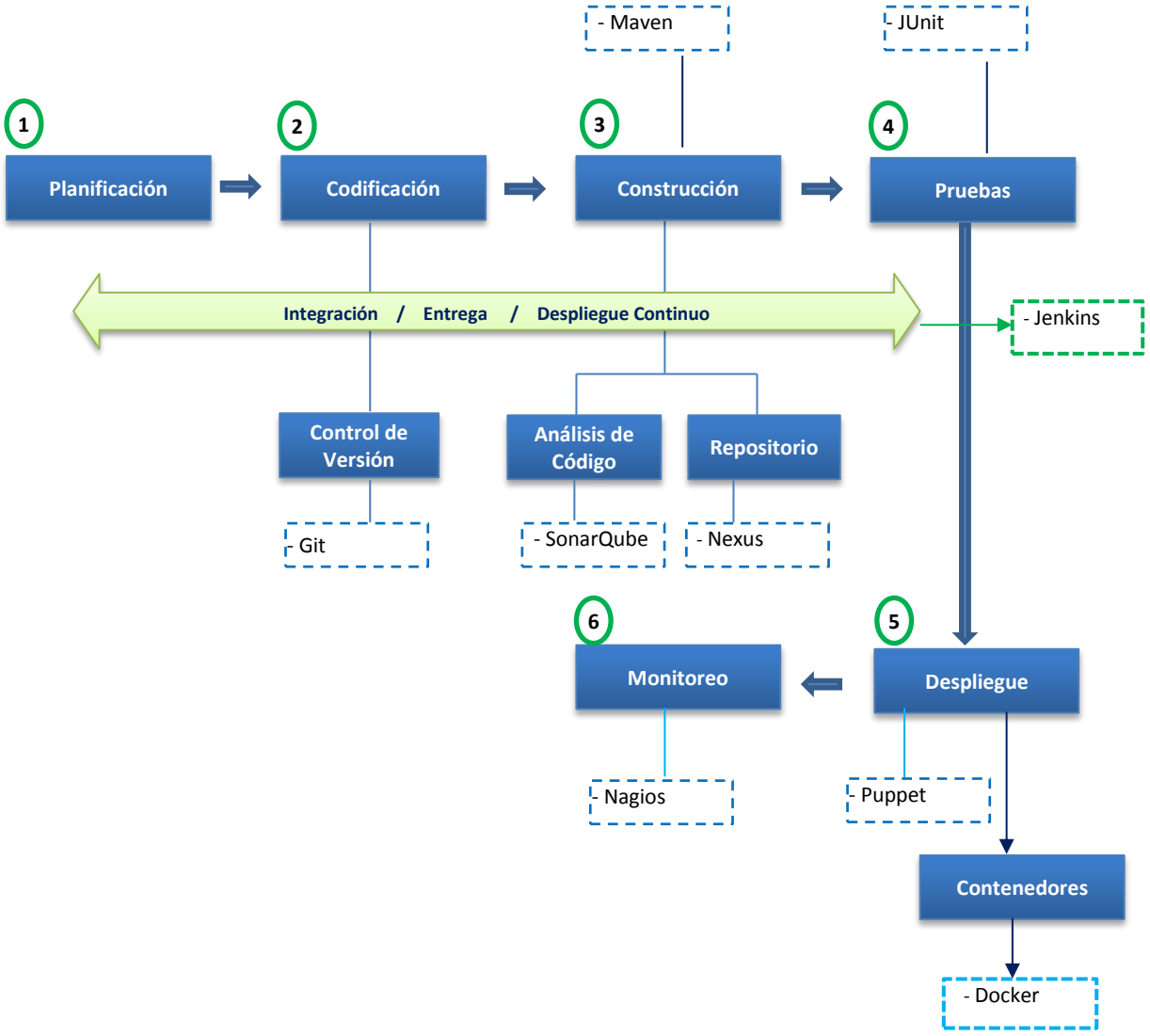

Figura 32: Modelo basado en Herramientas con licencias Open Source
Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Para una mejor comprensión del modelo en la Figura 33 se presenta el flujo propuesto a seguir de acuerdo a las fases y las prácticas continuas en el entorno DevOps.

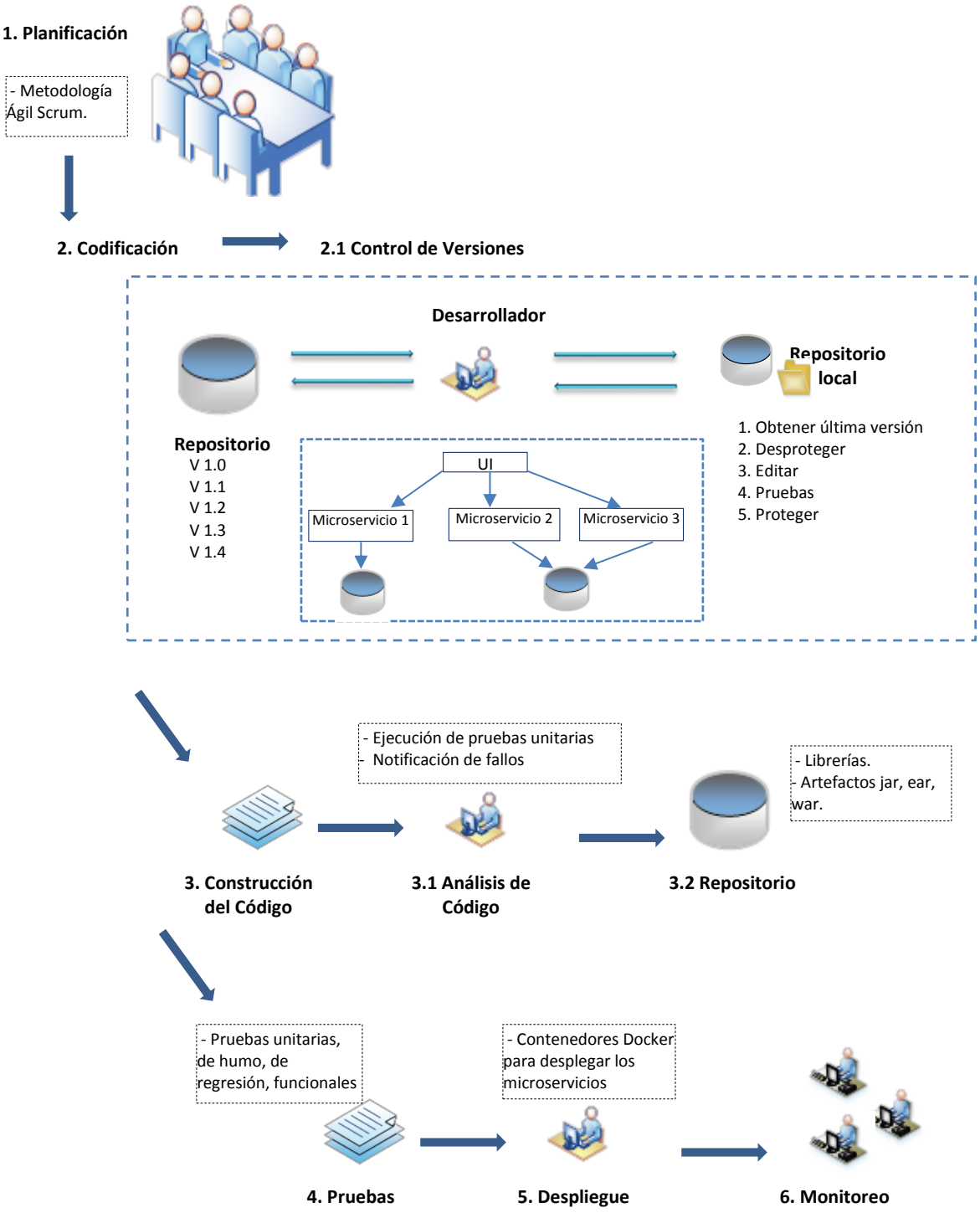

Figura 33: Flujo de trabajo de las herramientas que abarcan el ciclo de vida de DevOps con Integración-Entrega y Despliegue Continuo
Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Ahora se puede describir el proceso del modelo de acuerdo al flujo de trabajo con las herramientas open source.

1. Planificación

El flujo da inicio en la fase de Planificación con la aplicación de la metodología ágil Scrum, formado por los siguientes elementos según (Satpathy, 2016):

1. *Reunión Inicial* dirigida por del Dueño del Producto.
2. *Construcción del Backlog* con el Dueño del Producto, el Scrum Master y el Administrador del Sistema.
3. Elaboración del *Backlog del Producto*.
4. *Planeación del Spring* con el Dueño del Producto, el Administrador del Sistema, el Scrum Master y el equipo de Desarrollo.
5. *Daily Scrum* con el Scrum Master, el Equipo de Desarrollo y el Tester.
6. *Revisión del Sprint* con el Dueño del Producto, el Administrado del Sistema, equipo de Desarrollo y el Scrum Master.
7. *Sprint Retrospectivo* con el Dueño del Producto, el Administrado del Sistema, equipo de Desarrollo y el Scrum Master.

2. Codificación

Luego de haber realizado las planificaciones de las tareas y actividades en los respectivos Sprints se procede con la fase de Codificación.

Como se mencionó al comienzo de este documento nuestro enfoque es tener una Arquitectura de Microservicios las cuales serán desplegadas más adelante en contenedores Docker.

Para el desarrollo de los microservicios podemos usar distintos tipos de lenguaje de programación, ya que esta arquitectura nos permite integrar diferentes tipos de lenguajes sin problemas. Por ejemplo Python es un lenguaje muy potente para el desarrollo rápido de microservicios.

2.1 Control de Versiones

Para el control de versiones recomendamos usar Git como una herramienta de licencia Open Source creado por Linux. Con este sistema se asegura la integridad de los datos y es un apoyo para el flujo de trabajo.

Entre las características más importantes mencionadas por (“Git,” 2017) se encuentran:

- Permite el desarrollo no lineal.
- Facilita el desarrollo distribuido.
- Permite el manejar eficientemente proyectos de gran tamaño.
- Cada historia cuenta con una autenticación criptográfica.
- Cuenta con una emulación de Servidor CVS, lo que permite ser usado por clientes CVS y tiene complementos adicionales IDE que facilitan el acceso a los repositorios Git.
- A nivel de Repositorios es compatible con Subversion.
- A nivel de Sistemas Operativos es compatible con: Windows, Solaris, MacOS, BSD.

Puede usar los Protocolos: HTTP (Protocolo de Transferencia de Hipertexto), FTP (Protocolo de Transferencia de Archivos).

Entre los plugins disponibles tenemos:

- JGit es una librería hecha en Java. Usada para las aplicaciones Java. Es usada por Gerrit y EGit para el IDE de Eclipse.
- JS-Git hecho en JavaScript.
- Dulwich Git hecho para Python.
- Maven JGit-Flow Plugin.

3. Construcción

Para la fase de Construcción recomendamos usar Maven como una herramienta de licencia Open Source.

Apache Maven encontrado en (Porter, 2017) se basa en la creación de la estructura de un proyecto y un archivo que contiene todas las propiedades del mismo, denominado POM, el cual está escribo en XML en donde se registra el tipo de licencia, las dependencias del proyecto, los repositorios de artefactos, los plugins que colaboran con la adición de las nuevas funcionalidades del proyecto.

Permite compilar proyectos Java y ejecutar pruebas unitarias, también se pueden generar paquetes jar, war, ear.

Maven es compatible con Netbeans, Git, Eclipse, IntelliJ IDEA, JDeveloper 11G (11.1.1.3). Tiene plugins para Eclipse, JUnit, Subversion, Jira, Git.

3.1. Análisis de Código

Para el Análisis de Código recomendamos usar SonarQube como una herramienta de licencia Open Source, creado para evaluar el código fuente.

Entre las características más importantes mencionadas por (“SonarQube,” 2017) se encuentran:

- Maneja estándares para la codificación.
- Mantiene un historial de las métricas.
- Se pueden hacer pruebas unitarias.
- Presenta informes sobre duplicación de código.

Con los pluggins adecuados es compatible con Maven, Ant, Atlassian, Bamboo, Jenkins, Hudson, Eclipse, IntelliJ IDEA, Visual Studio, Gradle, Jenkins, Hudson.

3.2. Repositorio

Para el Repositorio recomendamos usar Nexus como una herramienta Open Source creado por Sonatype bajo licencia EPL.

Entre las características más importantes mencionadas por (“Nexus Repository OSS,” 2017) se encuentran:

- Permite el almacenamiento en caché de repositorios remotos, da mayor estabilidad y control, gracias a que ofrece una disponibilidad del 100%.
- Facilita el intercambio de artefactos internos.
- Soporta la colaboración entre los equipos mediante la distribución de componentes entre ellos.

Nexus OSS tiene componentes que son compatibles con Docker, java, Nuget, Ruby Maven. Existen versiones de Nexus OSS compatibles con Mac OS x, Windows, Unix.

4. Pruebas

Para la fase de Pruebas recomendamos usar JUnit como una herramienta Open Source con licencia EPL, que permite hacer pruebas unitarias para aplicaciones Java.

Entre las características más importantes mencionadas por (“JUnit,” 2017) se encuentran:

- Su framework cuenta con varias maneras de mostrar los resultados, una en modo de texto, una tarea en Ant o como un gráfico Swing o AWT.
- Permite controlar pruebas de regresión, para controlar si el nuevo código cumple con los requerimientos pasados y que su funcionalidad no ha sido alterada después de haber sido modificados.
- Evalúa los valores de retorno, si las clases cumplen con todas las especificaciones, enviando los respectivos mensajes como respuesta, si han tenido fallos o han sido exitosas.

Es compatible con Eclipse y NetBeans, ya que cuentan con plugins que tienen las plantillas para la creación automática de las pruebas por clases lo que ayuda al colaborador enfocarse solo en las pruebas y no en la creación de ellas.

5. Despliegue

Puppet es una herramienta Open Source escrita en Ruby para la gestión y configuración de código creada bajo licencia GNU (GPL).

Entre las características más importantes mencionadas por (“Puppet,” 2017a) se encuentran:

- Está compuesto por un lenguaje declarativo que permite describir la configuración del sistema.
- Puede ser distribuido por Linux, Ubuntu, Fedora, Mac Os X, Unix y Windows.
- Permite que el camino para la adopción de los microservicios y contenedores sea mucho más fácil.

6. Monitoreo

Nagios es una herramienta Open Source bajo licencia GNU, creado para el monitoreo y vigilancia.

Entre las características más importantes mencionadas por (“Nagios,” 2017) se encuentran:

- Es muy versátil a la hora de consultar y generar alertas las cuales pueden ser entregadas por diferentes medios, como SMS o correo electrónico.
- Permite el monitoreo de los servicios de red como POP3, SMTP, HTTP, también permite el monitoreo remoto mediante SSH o SSL cifrados.
- Se pueden desarrollar plugins en diferentes herramientas tales como Ruby, C++, Python.

7. Contenedores

Docker es un proyecto open source creado para automatizar el despliegue de aplicaciones en contenedores portátiles que pueden ejecutarse ya sea en la nube o localmente según (“Docker,” 2017).

Docker también es una empresa que promueve y desarrolla esta tecnología en colaboración con proveedores como Linux y Microsoft.

Permite el empaquetado de las aplicaciones, al ser una herramienta estándar que tiene todo lo necesario para que funcionen como el código, las bibliotecas.

8. Servidores de Integración/Entrega Continua

Como servidor de Integración Continua Open Source se recomienda usar Jenkins, que es una herramienta basada en Java, bajo licencia de Creative Commons y MIT. Colabora con las prácticas de Integración y Entrega Continua, permite automatizar todo tipo de tareas.

Entre las características más importantes mencionadas por (“Jenkins,” 2017) se encuentran:

- Su plataforma es Servlet Container.
- Al estar basado en Java es fácil de instalar en Windows, Mac OS X, Unix.
- Soporta herramientas de control de versiones como Subversion, Git, Mercurial, Perforce, Clearcase.
- Se pueden ejecutar proyectos con base en Ant o Maven.
- Las notificaciones se pueden dar en Twitter, Android, RSS, Google Calendar, E-mail, XMPP, IRC.
- Su IDE de integración: NetBeans, IntelliJ IDEA, Eclipse.

Tiene plugins disponibles para: Git, TFS, Perforce, Accurev, CMVC, Darcs, BitKeeper, CA Harvest, Mercurial, Bazaar, PlasticsSCM, StartTeam, PVCS, Surround, Dimensions, Unity Asset, Vault/Fortress, ClearCase, y SourceSafe.

5.2 Modelo basado en Herramientas con licencias Propietarias

En la Figura 32 se presenta un modelo basado en herramientas con una licencia tipo Propietaria.

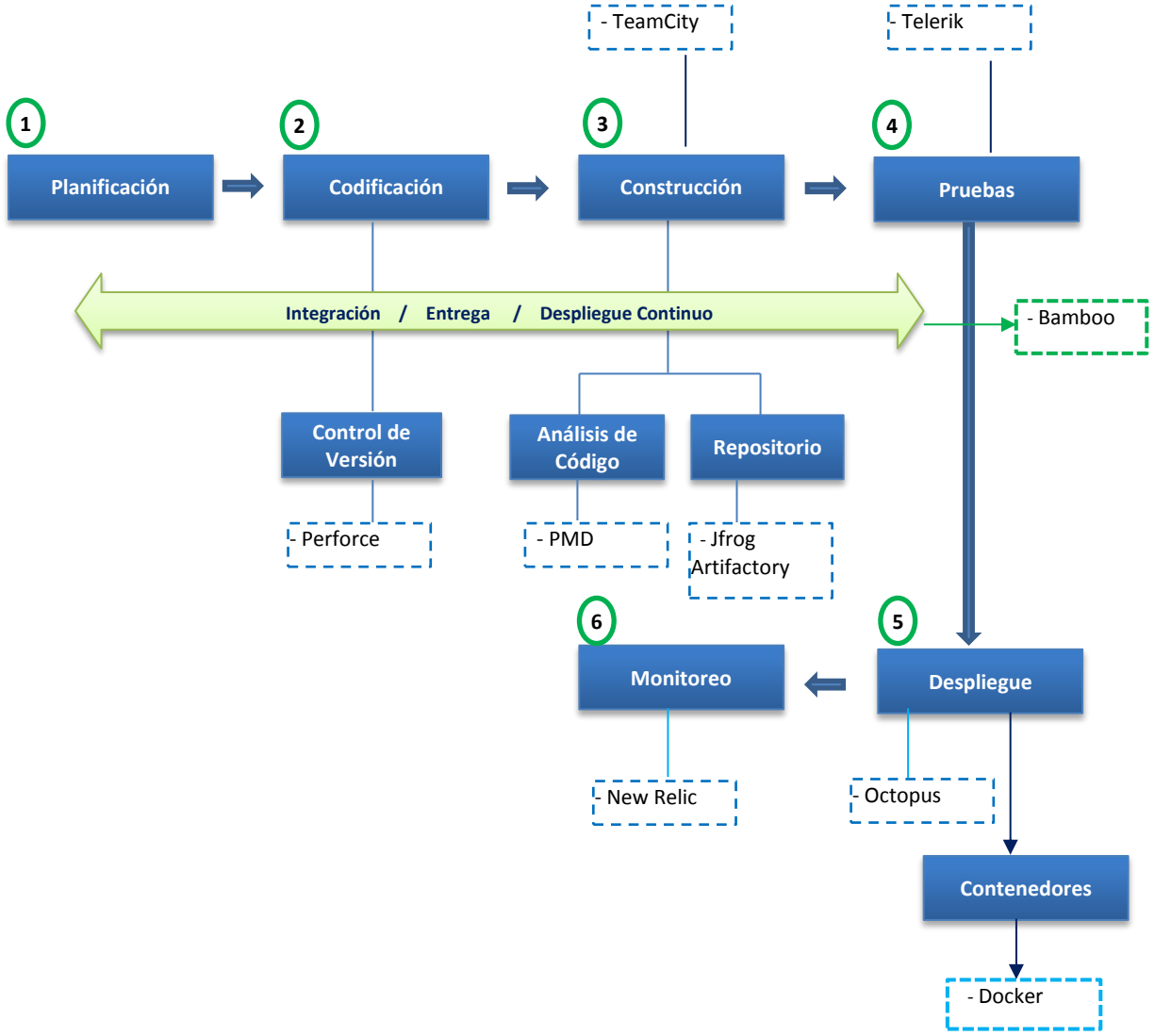

Figura 34: Modelo basado en Herramientas con licencias Propietarias
Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Se va a describir el proceso del modelo de acuerdo al flujo de trabajo presentado en la Figura 33 con las herramientas de licencia tipo Propietarias.

1. Planificación

El flujo da inicio en la fase de Planificación con la aplicación de la metodología ágil Scrum, formado por los siguientes elementos mencionados por (Satpathy, 2016):

1. *Reunión Inicial* dirigida por del Dueño del Producto.
2. *Construcción del Backlog* con el Dueño del Producto, el Scrum Master y el Administrador del Sistema.
3. Elaboración del *Backlog del Producto*.
4. *Planeación del Spring* con el Dueño del Producto, el Administrador del Sistema, el Scrum Master y el equipo de Desarrollo.
5. *Daily Scrum* con el Scrum Master, el Equipo de Desarrollo y el Tester.
6. *Revisión del Sprint* con el Dueño del Producto, el Administrado del Sistema, equipo de Desarrollo y el Scrum Master.
7. *Sprint Retrospectivo* con el Dueño del Producto, el Administrado del Sistema, equipo de Desarrollo y el Scrum Master.

2. Codificación

Luego de haber realizado las planificaciones de las tareas y actividades en los respectivos Sprints se procede con la fase de Codificación.

Como se mencionó al comienzo de este documento nuestro enfoque es tener una Arquitectura de Microservicios las cuales serán desplegadas más adelante en contenedores Docker.

Para el desarrollo de los microservicios podemos usar distintos tipos de lenguaje de programación, ya que esta arquitectura nos permite integrar diferentes tipos de lenguajes sin problemas. Por ejemplo Python es un lenguaje muy potente para el desarrollo rápido de microservicios.

2.1 Control de Versiones

Para el control de versiones recomendamos usar Perforce como una herramienta con licencia de tipo Propietaria, desarrollado por Perforce Software, Inc. Tiene una base de datos centralizada en donde se almacenan los repositorios conocidos como (depots) versionados donde se guardan los archivos. Su conexión es a través de TCP.

Entre las características más importantes mencionadas por (“Perforce,” 2017) se encuentran:

- Dependiendo de la configuración del Servidor los archivos de texto se pueden codificar en ASCII o Unicode.
- Tiene Soporte para archivos binarios, ASCII, Unicode.
- Los trabajos pueden ser almacenados de forma temporal y se puede cambiar de tarea.
- Se pueden comprimir los archivos para su almacenamiento y transmisión.
- Se pueden replicar los archivos y los metadatos.
- Se guarda un historial completo de las revisiones de los archivos modificados, eliminados copiados.
- Tiene un gestor para implementar políticas y restricciones.
- Permite gestionar archivos históricos para liberar espacio en el disco.
- Posee envió de alertar mediante email o RSS.

El Servidor de Perforce es gratuito para 20 usuarios con un número ilimitado de archivos, de superarse los usuarios se debe comprar la licencia.

Con los plugins adecuados es compatible para Windows, Linux, Solaris, MacOS, y FreeBSD. Tiene plugins para Visual Studio, Eclipse, Jenkins, Jira.

3. Construcción

Para la fase de Construcción recomendamos usar TeamCity como una herramienta de licencia Propietaria, es un servidor de gestión e integración basado en Java.

Entre las características más importantes mencionadas por (“TeamCity,” 2017) se encuentran:

- Ayuda con la gestión del proyecto mediante la supervisión y la generación de informes estadísticos.
- Posee una gran variedad de plugins disponibles para la integración con otras herramientas.
- Su interfaz de usuario es muy flexible, permite asignar funciones, clasificarlos en grupos.
- Colabora con la administración de la estructura del proyecto.
- Es compatible con Amazon EC2 y ofrece cobertura con .Net y Java y Ruby. Se puede integrar con Eclipse, Visual Studio, IntelliJ IDEA.

Los Sistemas de Control de Versiones compatibles son: Git, Subversion. Team Foundation Server (2005, 2008, 2010), IBM Rational Clear Case. CVS. Borland StarTeam, Mercurial.

Tiene plugins para Gradle, Docker, Octopus, .Net, Artifactory.

3.1. Análisis de Código

Para el Análisis de Código recomendamos usar PMD como una herramienta de licencia Comercial de BSD.

Entre las características más importantes mencionadas por (“PMD,” 2017) se encuentran:

- Su trabajo consiste en usar un conjunto de reglas las cuales pueden ser personalizadas que permiten identificar los errores.

- Tiene un detector para código duplicado o muerto.
- Soporta varios lenguajes entre ellos: PHP, Java, JavaScript, C, C ++, C#, PLSQL, Groovy, Ruby, Fortran, Python.
- Es compatible con Bamboo, Jenkins, Maven, Eclipse, JBuilder, NetBeans, JDeveloper.

Entre los pluggins disponibles se encuentran: Maven PMD plugin, Gradle: The PMD Plugin, Eclipse plugin, NetBeans plugin, JBuilder plugin, JDeveloper plugin, IntelliJ IDEA plugin.

3.2. Repositorio

Para la gestión de repositorio recomendamos usar JFrog Artifactory como una herramienta de licencia Propietaria.

Entre las características más importantes mencionadas por (“JFrog,” 2017) se encuentran:

- Ofrece apoyo en el ciclo de desarrollo como un gestor de repositorios binarios.
- Provee al equipo DevOps de las herramientas necesarias para la gestión eficiente del código desde las máquinas de los desarrolladores hasta el ambiente de producción.
- Permite la integración con herramientas de Integración Continua, como son, Jenkins, Hudson, TeamCity, Bambú, Team Foundation Server, Chef, Puppet, Ansible, Docker entre otras.

4. Pruebas

Para la fase de Pruebas recomendamos usar Telerik como una herramienta de licencia de tipo Propietaria creado por Progress.

Entre las características más importantes mencionadas por (“Telerik,” 2017) se encuentran:

- Permite implementar protocolos de prueba.
- Está compuesto por un conjunto de pruebas automatizadas.
- Test Studio es fácil de implementar ya que es compatible con lenguajes C# y VB.NET.

5. Despliegue

Para el despliegue recomendamos usar Octopus como una herramienta de licencia Propietaria. Es un servidor para automatizar despliegues. Dirigido para los desarrolladores en .Net.

Entre las características más importantes mencionadas por (“Octopus,” 2017) se encuentran:

- Provee de seguridad y confiabilidad en los entornos de prueba y producción para las aplicaciones Asp.net o servicios Windows.
- Octopus se encarga de la distribución de las aplicaciones que se encuentran empaquetadas.
- Permite la integración con TeamCity y con Microsoft Team Foundation Server.

6. Monitoreo

New Relic, es una herramienta para monitorizar las máquinas y ver en tiempo real los recursos disponibles. Su licencia es de tipo propietaria.

Entre las características más importantes mencionadas por (“New Relic,” 2017) se encuentran:

- Se pueden monitorear los procesos, red, la carga de las páginas, aplicaciones móviles, tanto a nivel de errores de datos, excepciones o por timeout.
- Presenta informes estadísticos sobre el rendimiento de las aplicaciones, sobre la velocidad de la red, o el uso de la memoria.
- Muy sencillo de instalar.

7. Contenedores

Docker es un proyecto open source creado para automatizar el despliegue de aplicaciones en contenedores portátiles que pueden ejecutarse ya sea en la nube o localmente según (“Docker,” 2017).

Docker también es una empresa que promueve y desarrolla esta tecnología en colaboración con proveedores como Linux y Microsoft.

Permite el empaquetado de las aplicaciones, al ser una herramienta estándar que tiene todo lo necesario para que funcionen como el código, las bibliotecas.

8. Servidores de Integración/Entrega Continua

Como servidor de Integración Continua de licencia tipo Propietaria se recomienda usar Bamboo, creado por Atlassian. Es una herramienta que sirve como apoyo para las prácticas de integración, despliegue y entrega continua.

Entre las características más importantes mencionadas por (“Bamboo,” 2017) se encuentran:

- Su plataforma es Servlet Container.
- Bamboo soporta la integración con Perforce, Git, Mercurial, Subversion.
- Permite crear planes para realizar las compilaciones, se puede configurar los activadores que dan inicio a las compilaciones.
- Se pueden automatizar las pruebas y configurarlas en forma paralela, lo que agilita la detección de errores.
- Las notificaciones se pueden dar en XMPP, RSS, E-mail, Remote API, Google Talk.
- Su IDE de integración: Visual Studio, Eclipse, IntelliJ IDEA.

Permite la integración con Bitbucket, HipChat, Jira, Fisheyey, AWS CodeDeploy y Docker.

CONCLUSIONES

Al terminar el presente trabajo se puede concluir que:

DevOps es una cultura, movimiento o práctica que hace hincapié en la colaboración y la comunicación de los desarrolladores de software y otros profesionales de TI, al automatizar el proceso de entrega de software y cambios de infraestructura.

DevOps garantiza que los entornos de desarrollo, entornos físicos y procesos de una organización se configuren para entregar nuevas versiones producción lo más rápidamente posible.

La adopción DevOps obligará a las organizaciones a adoptar nuevas habilidades desde una perspectiva técnica y cultural.

En un esquema eficiente de despliegue de aplicaciones, los microservicios representan las características de escalabilidad, reutilización y resiliencia, y los contenedores pueden resolver la mayoría de los problemas de eficiencia de recursos.

La integración e interacción de DevOps y microservicios, permite a los equipos de desarrollo trabajar con aplicaciones poco acopladas y elegir el esquema tecnológico más adecuado para sus necesidades.

Los contenedores y microservicios son útiles en un ambiente DevOps debido a que promueven cambios pequeños e incrementales, así como diferencias mínimas entre los entornos de desarrollo y prueba, y el entorno de producción final.

El ambiente de construcción propuesto, permitirá a las empresas garantizar la integración e interacción entre microservicios y contenedores, y a través de DevOps aumentar la frecuencia de implementación y conducir a un tiempo de comercialización más rápido de los productos.

RECOMENDACIONES

Al adoptar una cultura DevOps en la empresa es importante considerar las siguientes recomendaciones:

- Las organizaciones que adopten un enfoque DevOps, deben entender el cambio implica un cambio cultural a nivel organizacional, por lo que será necesario implementar planes de entrenamiento dirigido al staff y personal técnico.

- La transacción hacia un enfoque DevOps, implica la implementación de un plan de adaptación/migración de la infraestructura organizacional actual (entorno de desarrollo - entorno físico - procesos) hacia un esquema de desarrollo y operaciones, y entregas continuas.

- Las organizaciones que implementen microservicios y contendores deberán establecer un esquema metodológico (conjunto de pasos) que conjuntamente con DevOps generen los resultados deseados por las organizaciones.

- Se recomienda la aplicación del esquema propuesto como un modelo completo debido a que este cubre cada una de las fases del ciclo de vida DevOps y las prácticas ágiles de integración, entrega y despliegue de microservicios y contenedores.

Bibliografía

Ball, B. (2015). *Applications and Microservices with Docker*. (A. Williams, Ed.) (Vol. 2).

Bamboo. (2017). Recuperado de https://es.atlassian.com/software/bamboo

BBVA API MARKET. (2014). Las 5 licencias de software libre más importantes que todo desarrollador debe conocer. Recuperado de https://bbvaopen4u.com/es/actualidad/las-5-licencias-de-software-libre-mas-importantes-que-todo-desarrollador-debe-conocer

Bernal, G. (2016). *Arquitecturas con Microservicios*.

Bieberstein, N., Laird, R. G., Jones, D. K., & Mitra, T. (2008). *Executing SOA*. (Pearson Education, Ed.). Boston.

CA Technologies. (2015). Perspectivas de Devops 2. Recuperado de http://www.idglat.com/afiliacion/whitepapers/Perspectivas DevOps CA 1.pdf?tk=/

Camacho, E., Cardeso, F., & Nuñez, G. (2004). *Arquitecturas de software, Guía de Estudio*.

Campos, Y. (2008). Estilos Arquitectónicos. Recuperado de http://estilosarquitectonicos.blogspot.com/

Carrizo, S., Cucu, S., Modir, S., & García, M. (2015). *Using Liberty for DevOps, Continuous Delivery, and Deployment*. (Redbooks, Ed.) (First Edit). Poughkeepsie, NY: Redbooks.

Davis, J., & Daniels, K. (2016). *Effective DevOps*. (B. Anderson, Ed.) (First Edit). Sebastopol: O'Reilly Media, Inc.

De la Torre, C. (2016). *Containerized Docker Application Lifecycle with Microsoft Platform and Tools*.

Docker. (2017). Recuperado de https://www.docker.com/

Earnshaw, A. (2016). What a Devops Engineer. Recuperado de https://puppet.com/blog/what-a-devops-engineer

Edwards, D. (2009). The (Short) History of DevOps. Recuperado de http://ccsubs.com/video/yt%3Ao7-IuYS0iSE/the-short-history-of-devops/subtitles?lang=es

Farcic, V. (2017). *The DevOps 2.0 Toolkit Automating the Continuous Deployment Pipeline with Containerized Microservices*. Leanpub book.

Fowler, M. (2014). Microservicios. Recuperado de https://martinfowler.com/articles/microservices.html

Gartner. (2017). IT Glosary. Recuperado de http://www.gartner.com/it-glossary/devops/

Git. (2017). Recuperado de https://git-scm.com/

Gómez, J. (2014). Importancia del Gobierno SOA en la Organización. Recuperado de https://www.ibm.com/developerworks/ssa/library/govSOA/

Gosuke, M. (n.d.). RSpec pruebas. Recuperado de http://serverspec.org/

Gutierrez, D. (2013). Estilos Arquitectónicos. Recuperado de https://es.slideshare.net/piojosnos/clase-08a-estilosarquitectonicos

Hauer, P. (2015). Tutorial: Continuous Delivery with Docker and Jenkins. Recuperado de https://blog.philipphauer.de/tutorial-continuous-delivery-with-docker-jenkins/

Harvard University. (2014). Reference Model.

Humble, J., & Farley, D. (2011). *Continuos Delivery*. (I. Pearson Education, Ed.). Boston.

IBM Corp. (2015). *Microservices from Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach*. Poughkeepsie, NY: Redbooks.

ISO/IEC/IEEE 42010. (2011). *International Standard ISO/IEC/IEEE 42010 - Systems and software engineering - Architecture description* (Vol. 2011). Switzerland.

Jenkins. (2017). Recuperado de https://jenkins.io/

JFrog. (2017). Recuperado de https://www.jfrog.com/open-source/

JUnit. (2017). Recuperado de http://junit.org/junit4/

Kavis, M. (2014). Nagios Monitoring Strategy. Recuperado de https://devops.com/nagios-monitoring-strategy/

Kort, W. De. (2016). *DevOps on the Microsoft Stack*. (W. Spahr, Ed.).

Kruchten, P. (2006). *El Modelo de "4+1" Vistas de la Arquitectura del Software* (Vol. 12).

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2011). NIST Cloud Computing Reference Architecture Recommendations of the National Institute of Standards and.

M., S., & Garlan, D. (1994). An Introduction to Software Architecture. Recuperado de https://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf

Manifesto for Agile Software. (2001). Principios del Manifiesto Ágil. Recuperado de http://agilemanifesto.org/iso/es/principles.html

Menzel, G. (2015). *DevOps - The Future of Application Lifecycle Automation*.

Microsoft Corporation. (2009). *Microsoft Application Architecture Guide*.

Microsoft Corporation. (2016). Arquitectura de aplicaciones de .NET: Diseño de aplicaciones y servicios. Recuperado de https://msdn.microsoft.com/es-es/library/ms954595.aspx

Nagios. (2017). Recuperado de https://www.nagios.org/

Nexus Repository OSS. (2017). Recuperado de https://www.sonatype.com/nexus-repository-oss

New Relic. (2017). Recuperado de https://newrelic.com/

Newman, S. (2015). *Building Microservices*. (M. L. and B. MacDonald, Ed.). Sebastopol, California: O'Reilly Media, Inc.

Octopus. (2017). Recuperado de https://octopus.com/

Perforce. (2017). Recuperado de https://www.perforce.com/

PMD. (2017). Recuperado de https://pmd.github.io/

Porter, B. (2017). Maven. Recuperado de https://maven.apache.org/

Puppet. (2017a). Recuperado de https://puppet.com/

Puppet. (2017). *State of DevOps Report*. Recuperado de puppet.com/state-of-devops-report

Red Hat. (2017). Devops tools. Recuperado de https://www.ansible.com/devops-tools

Reinoso, C., & Kicillof, N. (2004). *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. Buenos Aires, Arg.

Ríos, A. F. (2016). Patrón de Arquitectura Broker. Recuperado de http://es.slideshare.net/montoya118/broker-37606068

Rubin, K. (2013). *Essential Scrum: a practical Guide to the most popular agile process*. (Addison-Wesley, Ed.). Michigan: Pearson Education, Inc.

Satpathy, T. (2016). *Una guía para el CUERPO DE CONOCIMIENTO DE SCRUM (Guía SBOK™)* (2016th ed.). Arizona: VMEdu, Inc. Recuperado de www.scrumstudy.com

Sharma, S., & Coyne, B. (2015). *DevOps for Dummies*. (R. Mengle, Ed.). Hoboken, Nueva Jersey: John Wiley & Sons, Inc.

Software Engineering Institute - University Carnegie Mellon. (2016). Community Software Architecture Definitions. Recuperado de http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

Sommerville, I. (2011). *Ingeniería de Software*. (Editorial Pearson Educación, Ed.). México.

Wilsenach, R. (2015). Devops Culture. Recuperado de https://martinfowler.com/bliki/DevOpsCulture.html

Erl, T. (2009). *SOA Design Patterns*. Prentice Hall.

Matsumura, M., Brauel, B., & Shah, J. (2009). Adopción de SOA para Dummies. Hoboken, Nueva Jersey: Wiley Publishing.

SonarQube. (2017). Recuperado de https://www.sonarqube.org/

Swartout, P. (2014). Continuous Delivery and DevOps – A Quickstart Guide. Packt Publishing Ltd.

TeamCity. (2017). Recuperado de https://www.jetbrains.com/teamcity/

Telerik. (2017). Recuperado de http://www.telerik.com/teststudio

ANEXOS

1. Matriz de Evaluación de las Herramientas

Para la elección de las herramientas se han elaborado las siguientes matrices de evaluaciones de acuerdo a criterios considerados de mayor interés que se recomiendan considerar en la hora de elegir una herramienta idónea.

Ponderación de los criterios

En la Tabla 24 se detalla cómo ponderan cada uno de los criterios del presente trabajo:

Tabla 24: Ponderación de los criterios

Criterio	Peso
Tipo de Licencia	10
Pluggins disponibles	4
Compatibilidad Sistemas Operativos	8
Lenguaje de Desarrollo	3
Documentación disponible	6

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

- Tipo de Licencia: Si es Open Source o Comercial. En el caso de que sea Open Source es importante identificar el tipo de licencia que tiene GPL, BSD, Apache.

  Según (BBVA API MARKET, 2014) nos muestra la importancia de estas licencias:

  GNU GPL: Licencia libre, abierta, con atribuciones del autor.
  BSD: Licencia libre, abierta sin atribuciones del autor.
  Apache: Licencia libre, con propiedad intelectual, con atribuciones del autor.
  EPL: Licencia libre, abierta, de Eclipse.
  MIT: Licencia libre del Instituto Tecnológico de Massachusetts.

- Pluggins Disponibles: A mayor número de pluggins disponibles se extiende la compatibilidad con otras herramientas.

- Tipo de lenguaje: Es necesario conocer en qué tipo están desarrollados los pluggins por motivos de futuras integraciones con otras herramientas.

- Compatibilidad Sistemas Operativos: La compatibilidad con los diferentes sistemas operativos es un punto importante a la hora de elegir una herramienta.

- Documentación Disponible: El soporte que ofrece la herramienta se convierte en un factor de mucha ayuda para los desarrolladores.

Valoración de las Herramientas

En la siguiente tabla se detalla la valoración de las herramientas por cada criterio:

Tabla 25: Valoración de las herramientas según los criterios

Criterio		Calificación	Peso
Tipo de Licencia			**10**
	Libre	10	
	Abierta	9	
	Sin atribuciones del autor	8	
	Con atribuciones del autor	7	
	Comercial	6	
Compatibilidad Sistemas Operativos			**8**
	Windows, Linux, Mac OS, Solaris, BSD	10	
	Windows, Linux, Mac OS	9	
	Sólo Windows	8	
Documentación disponible			**6**
	Web	10	
	Libros	9	
Pluggins disponibles			**4**
	>= 5	10	
	<5 y >=3	9	
	<3	8	
Lenguaje de Desarrollo			**3**
	C, C++	10	
	Java	9	
	Otros	8	

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas en la fase de Codificación

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas en la fase de Codificación

Criterios		Herramientas					
		Git		Subversion (SVN)		Perforce	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	27	270	6	60
Sistema Operativo	8	10	80	9	72	10	80
Documentación disponible	6	19	114	19	114	19	114
Pluggins disponibles	4	10	40	9	36	9	36
Lenguaje de desarrollo	3	18	54	10	30	19	57
Total			**548**		**522**		**347**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas en la fase de Construcción

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas en la fase de Construcción

Criterios		Herramientas					
		Ant		Maven		TeamCity	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	26	260	6	60
Sistema Operativo	8	10	80	10	80	9	72
Documentación disponible	6	19	114	19	114	19	114
Pluggins disponibles	4	8	32	10	40	10	40
Lenguaje de desarrollo	3	9	27	9	27	9	27
Total			**513**		**521**		**313**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para el Análisis de Código

A continuación se presenta la matriz usada para evaluar las herramientas en esta etapa:

Matriz de Evaluación Herramientas Análisis de Código

Criterios		Herramientas					
	Peso	Checkstyle		SonarQube		PMD	
		Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	26	260	27	270
Sistema Operativo	8	9	72	9	72	9	72
Documentación disponible	6	10	60	19	114	19	114
Pluggins disponibles	4	10	40	10	40	10	40
Lenguaje de desarrollo	3	9	27	9	27	9	27
Total			**459**		**513**		**523**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para la Gestión de Repositorios

A continuación se presenta la matriz usada para evaluar las herramientas en esta etapa:

Matriz de Evaluación Herramientas Gestión de Repositorios

Criterios		Herramientas			
	Peso	Nexus Repository OSS		TeamCity	
		Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	6	60
Sistema Operativo	8	9	72	9	72
Documentación disponible	6	19	114	10	60
Pluggins disponibles	4	9	36	10	40
Lenguaje de desarrollo	3	9	27	10	30
Total			509		262

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para la fase de Pruebas

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas para la fase de Pruebas

Criterios		Herramientas					
		Junit		Roslyn		Test Studio-Telerik	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	26	260	6	60
Sistema Operativo	8	9	72	9	72	9	72
Documentación disponible	6	19	114	19	114	19	114
Pluggins disponibles	4	9	36	8	32	8	32
Lenguaje de desarrollo	3	9	27	8	24	8	24
Total			**509**		**502**		**302**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para la fase de Despliegue

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas para la fase de Despliegue

Criterios		Herramientas					
		Octopus		Puppet		Chef	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	6	60	26	260	26	260
Sistema Operativo	8	9	72	10	80	9	72
Documentación disponible	6	10	60	19	114	19	114
Pluggins disponibles	4	8	32	8	32	8	32
Lenguaje de desarrollo	3	10	30	8	24	8	24
Total			**254**		**510**		**502**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para la fase de Monitoreo

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas para la fase de Monitoreo

Criterios		Herramientas					
		Nagios		New Relic		System Center	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	6	60	6	60
Sistema Operativo	8	9	72	10	80	9	72
Documentación disponible	6	19	114	10	60	19	114
Pluggins disponibles	4	8	32	8	32	8	32
Lenguaje de desarrollo	3	18	54	9	27	8	24
Total			**532**		**259**		**302**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para la CI/CD

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas para la CI/CD

Criterios		Herramientas					
		Jenkins		Bamboo		CruiseControl	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	26	260	6	60	27	270
Sistema Operativo	8	9	72	9	72	9	72
Documentación disponible	6	19	114	10	60	10	60
Pluggins disponibles	4	10	40	10	40	10	40
Lenguaje de desarrollo	3	9	27	9	27	9	27
Total			**513**		**259**		**469**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro

Matriz de Evaluación de Herramientas para la Contenedores

A continuación se presenta la matriz usada para evaluar las herramientas en esta fase:

Matriz de Evaluación Herramientas para los Contenedores

Criterios		Herramientas					
		Windows Server Container		Docker		Amazon EC2 Container Service	
	Peso	Pts.	Res.	Pts.	Res.	Pts.	Res.
Tipo de Licencia	10	6	60	26	260	6	60
Sistema Operativo	8	9	72	9	72	9	72
Documentación disponible	6	19	114	19	114	19	114
Pluggins disponibles	4	8	32	10	40	8	32
Lenguaje de desarrollo	3	8	24	8	24	9	27
Total			**302**		**510**		**305**

Fuente: Autor, elaborado por Ivonne Karina Farías Alejandro