INSTITUTO SUPERIOR TECNOLÓGICO QUITO

Formamos tu PROPÓSITO DE VIDA

GUÍA GENERAL DE

AMBIENTES COMPUTACIONALES

AUTOR: GABRIEL HOYOS

ISBN: 978-9942-562-63-0

ITQ

WWW.ITQ.EDU.EC

INVESTIGACIÓN

DOXA EDITION

INSTITUTO SUPERIOR TECNOLÓGICO QUITO
Excelencia en Educación Superior

GUÍA GENERAL DE AMBIENTES COMPUTACIONALES

AUTOR: ING. GABRIEL HOYOS

PRIMERA EDICIÓN

AÑO: 2025

TRABAJO EN EDICIÓN:

DOXA EDITION

EQUIPO EDITORIAL:

MSc. ÁNGEL TAYUPANTA

MSc. KEYERMAN TOAPANTA CISNEROS

ISBN: 978-9942-562-63-0

ISBN: 978-9942-562-63-0
9 789942 562630

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

SOBRE EL AUTOR

Gabriel Hoyos Ingeniero de sistemas graduado en la mención desarrollo de aplicaciones para la gestión en la Universidad Politécnica Salesiana en el año 2018. Su carrera ha estado marcada por una basta trayectoria en proyectos relevantes en el manejo, supervisión, soporte e implementación de ERP's así como desarrollo y diseño de los mismos. Su experiencia y habilidades técnicas le han permitido destacarse en el ámbito de la informática, desarrollo y la tecnología gracias a una sólida comprensión de los sistemas de planificación de recursos empresariales (ERP), con experiencia en la implementación, configuración y personalización de varias plataformas ERP para satisfacer las necesidades específicas de las organizaciones.

He liderado y gestionado proyectos de implementación de ERP desde la fase de planificación hasta la ejecución y el soporte post-implementación, asegurando el cumplimiento de plazos y presupuesto soy experto en trabajar con stakeholders para identificar y documentar requisitos de negocio, lo que me permite adaptar el ERP para mejorar la eficiencia operativa y apoyar los objetivos estratégicos de la empresa utilizado mi conocimiento en ERP´s para analizar y optimizar procesos de negocio, reduciendo costos y aumentando la productividad mediante la automatización y la mejora continua.

Además de proporcionar soporte técnico experto y de capacitación a usuarios finales, ayudándolos a maximizar el uso del sistema ERP y resolver cualquier problema técnico que pueda surgir mediante un profundo conocimiento de las tecnologías y herramientas utilizadas en el desarrollo de interfaces de software, incluyendo lenguajes de programación, bibliotecas, y frameworks de front-end como HTML, CSS, JavaScript, React, Angular, entre otros.

Esto a su vez le ha permitido mantenerse al día con las últimas tendencias y avances en el desarrollo de interfaces de software, incorporando nuevos conocimientos y tecnologías en la enseñanza siendo esta la pauta para comprometerse con el campo del desarrollo y trabajar con otros departamentos y disciplinas que me han permitido integrar el diseño, desarrollo e implementación de interfaces en un contexto más amplio, mostrando cómo se relaciona con otros aspectos dentro del desarrollo de software y la tecnología.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ÍNDICE GENERAL

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

GUÍA GENERAL DE AMBIENTES COMPUTACIONALES

1. DESCRIPCIÓN DE LA ASIGNATURA

La presente asignatura proporciona un conjunto de conocimientos, que permitirá al estudiante identificar los diferentes ambientes computacionales que se pueden encontrar en el ámbito del desarrollo de software en el cual los desarrolladores podrán crear distintos tipos de repositorios y como trabajar dentro de cada uno de ellos. Proporciona al alumno conocimiento sobre las herramientas, técnicas y procedimientos necesarios para la creación y configuración de ambientes computacionales ya sean locales, centralizados o distribuidos para facilitar al desarrollo de aplicaciones u otros procesos dentro de una corporación, mediante sistemas de control de versiones en git y github. El egresado podrá participa en cualquier proyecto a nivel nacional o internacional el cual mediante al manejo de repositorio no se le hará difícil encontrar trabajo fuera del país.

2. BIBLIOGRAFÍA

2.1. Básica

Calzada Prado, Javier Francisco, 2010, Repositorios, bibliotecas digitales y CRAI: los objetos de aprendizaje en la educación superior, ALFAGRAMA EDICIONES, 9789871305575

Enrique luna ramirez, 2006, El repositorio de metadatos en un data warehouse, RED REVISTA DE FACULTAD DE INGENIERÍA, 07181337

Este texto aborda los principios fundamentales para utilizar las herramientas que corresponden a un ambiente virtual en el contexto de herramientas de desarrollo, plataformas como GitHub y GitLab son fundamentales para la colaboración y gestión de proyectos. GitHub es conocido por su amplia comunidad y ecosistema de integraciones, mientras que GitLab se destaca por su enfoque integral en el ciclo de vida DevOps, incluyendo integración y despliegue continuos (CI/CD). Además, ProGit es una guía esencial para entender Git, proporcionando conocimientos profundos sobre el control de versiones.

Por otro lado, Docker facilita la implementación de aplicaciones en contenedores, lo que mejora la eficiencia y portabilidad en entornos de desarrollo y producción. Al integrar Docker con GitLab o GitHub, los equipos pueden automatizar el despliegue de aplicaciones de manera más eficiente, aprovechando las capacidades CI/CD de estas plataformas para agilizar los flujos de trabajo.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

En resumen, el diseño de interfaces efectivas, combinado con el uso de herramientas como GitHub, GitLab, ProGit y Docker, permite a los desarrolladores crear experiencias de usuario atractivas y funcionales, mientras optimizan sus procesos de desarrollo y despliegue de software.

2.2. Complementaria

Using Docker: Developing and Deploying Software with Containers, 978-1491915769

El contenido de este texto se centra en la comprensión práctica de Docker, una herramienta poderosa para el despliegue de aplicaciones en contenedores. A lo largo del tiempo, Docker ha evolucionado para ofrecer una solución eficiente y escalable para desarrolladores, permitiéndoles crear, desplegar y gestionar aplicaciones de manera consistente en diferentes entornos. Al finalizar cada unidad, encontrarás ejercicios que te ayudarán a evaluar tu progreso en la comprensión de los temas técnicos relacionados con Docker. Este enfoque práctico sobre Docker es un clásico en el campo del desarrollo de software y ofrece consejos valiosos para crear despliegues de aplicaciones intuitivos y eficientes.

Docker se destaca por su capacidad para proporcionar contenedores ligeros y rápidos, que comparten el mismo kernel del sistema operativo del host, lo que elimina la necesidad de un sistema operativo completo en cada instancia. Además, Docker facilita la portabilidad, escalabilidad y seguridad de las aplicaciones, lo que lo convierte en una herramienta esencial para cualquier proyecto de desarrollo moderno.

No Version Control with Git: Powerful tools and techniques for collaborative software development, 9781449345044

Los contenidos están estratégicamente organizados para presentar una gran cantidad de ejemplos aplicables sobre el uso y aplicación de Git en el desarrollo de software. A través de estos ejemplos, el lector adquiere la capacidad de analizar los resultados obtenidos del uso de Git y su respectivo seguimiento de cambios en el código. De esta manera, se puede comprender cómo Git permite a los equipos colaborar eficientemente, realizar un seguimiento detallado de los cambios en el código base y revertir a versiones anteriores cuando sea necesario.

Git, como sistema de control de versiones distribuido, facilita la gestión de proyectos de software al permitir que múltiples desarrolladores trabajen simultáneamente en diferentes ramas del código, fusionando cambios de manera segura y eficiente. Además, su capacidad para almacenar un historial completo de cambios permite una auditoría precisa y la capacidad de recuperar versiones anteriores del proyecto si es necesario

Docker Deep Dive, 978-1521822807

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Este texto se centra en Docker Desktop, una herramienta esencial para crear, probar y desplegar aplicaciones en contenedores de manera eficiente. Docker Desktop es una aplicación de instalación sencilla que permite a los desarrolladores gestionar contenedores, aplicaciones y imágenes directamente desde su máquina, ya sea en Windows, macOS o Linux. Docker Desktop incluye una interfaz gráfica de usuario (GUI) intuitiva que simplifica la gestión de contenedores y aplicaciones, reduciendo el tiempo dedicado a configuraciones complejas y permitiendo a los desarrolladores centrarse en escribir código. Además, proporciona acceso a una amplia biblioteca de imágenes certificadas y plantillas en Docker Hub, lo que facilita la colaboración y el desarrollo continuo de aplicaciones. Algunos de los componentes clave de Docker Desktop incluyen:

Docker Engine: El motor que ejecuta los contenedores.

Docker CLI client: La interfaz principal para interactuar con Docker.

Docker Compose: Permite definir y ejecutar aplicaciones compuestas por múltiples contenedores.

Kubernetes: Integración para la orquestación de contenedores a gran escala.

Docker Desktop es ideal para desarrolladores que buscan una solución completa para trabajar con contenedores, ya que ofrece una instalación rápida y sencilla de un entorno de desarrollo completo, junto con actualizaciones regulares de seguridad y correcciones de errores

Scott Chacon, Ben Straub, 2025, PRO GIT, AEXPRESS, 0002

Este texto se centra en la importancia de Git en el diseño y desarrollo de proyectos, destacando cómo la usabilidad y la estética son fundamentales para crear experiencias interactivas efectivas. Al igual que un diseñador debe considerar la experiencia del usuario al crear interfaces, un desarrollador debe entender cómo Git facilita la colaboración y el seguimiento de cambios en el código. Git es como un compañero de trabajo que ayuda a mantener el orden y la claridad en el desarrollo de software. Imagina que tienes un proyecto en el que varias personas trabajan juntas. Git permite que cada persona haga cambios en su propia copia del proyecto sin afectar el trabajo de los demás. Los cambios se almacenan en un directorio de trabajo, se preparan en el área de preparación y finalmente se guardan en el repositorio local. Algunos de los elementos clave de Git incluyen:

Confirmaciones: Son como instantáneas del proyecto en un momento dado, permitiendo revertir cambios si es necesario.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ramas: Permiten trabajar en diferentes versiones del proyecto simultáneamente, lo que facilita la colaboración y prueba de nuevas ideas.

Comandos básicos: Como git init, git add, git commit, y git push, que son herramientas esenciales para gestionar el flujo de trabajo.

Al igual que un diseñador debe probar y ajustar su trabajo para asegurarse de que sea intuitivo y atractivo, un desarrollador debe usar Git para probar y ajustar su código, asegurándose de que funcione correctamente y sea fácil de mantener. Los ejercicios prácticos con Git permiten a los desarrolladores consolidar sus habilidades y mejorar su comprensión del sistema de control de versiones, lo que es esencial para crear proyectos de software robustos y escalables.

3. COMPETENCIAS GENÉRICAS Y ESPECÍFICAS

Configurar y gestionar ambientes computacionales, comprendiendo su arquitectura, virtualización, computación de la nube y seguridad. A través de la exploración de herramientas y metrologías m los estudiantes optimizarán el uso de recursos tecnológicos en entornos empresariales y de desarrollo.

4. OBJETIVO GENERAL

Formar competencias para el desarrollo de un perfil ético profesional, aplicando herramientas informáticas además de estándares nacionales e internacionales ligados al ámbito de tecnologías de la información, que le permitirá al estudiante ejecutar proyectos de desarrollo de software que se esté desarrollando en cualquier parte del mundo.

5. FORMACIÓN CIUDADANA, VALORES Y HABILIDADES BLANDAS

- Educación Ambiental: Consumo y Producción Sostenibles
- Valores & Habilidades blandas: amor: comunicación asertiva y escucha activa
- Valores & Habilidades blandas: compromiso social: adaptabilidad
- Valores & Habilidades blandas: cultura: creatividad
- Valores & Habilidades blandas: gratitud: resiliencia
- Valores & Habilidades blandas: justicia: resolución de problemas
- Valores & Habilidades blandas: lealtad: liderazgo

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Valores & Habilidades blandas: optimismo: planificación y gestión del tiempo
- Valores & Habilidades blandas: orgullo nacional: pensamiento crítico
- Valores & Habilidades blandas: respeto: inteligencia emocional.
- Valores & Habilidades blandas: solidaridad: trabajo en equipo
- Valores & Habilidades blandas: tolerancia: flexibilidad
- Valores & Habilidades blandas: verdad y honradez: proactividad

6. NORMAS DE CLASE

Con relación a las normas de clase, es importante destacar que la evaluación de los componentes de gestión académica se compone de tres notas sumativas, cada una con una puntuación máxima de 6.60/6.60, así como un proyecto práctico, como evaluación formativa que se valora con 3.40/3.40, lo que da un total de 10/10 para la calificación del módulo. Los parciales se califican en una escala de hasta 6.60 puntos, representando cada uno el 2.22 de la calificación total de 6.6 puntos. Para presentarse al proyecto final, el estudiante debe haber obtenido al menos 4.50 puntos sumando las tres primeras notas. En caso de no alcanzar este mínimo en el proyecto, se otorga una oportunidad de recuperación dentro de las 48 horas laborables siguientes, según el calendario académico oficial. La nota mínima acumulada requerida para aprobar la asignatura es 7/10, y es esencial mantener al menos un 70% de asistencia a las clases. Los docentes deben informar a los estudiantes sobre sus notas individuales antes de registrarlas en el sistema, y se espera que los alumnos confirmen su aceptación y conformidad con estas calificaciones. Además, los docentes deben entregar un reporte de notas y asistencia a través del SGA y notificado a la coordinación de carrera y registrar las calificaciones en el sistema en un plazo máximo de 5 días posteriores a la recepción del proyecto final.

7. SISTEMA DE EVALUACIÓN

La evaluación de los componentes de gestión académica se compone de tres notas sumativas, cada una con una puntuación máxima de 6.60/6.60, así como un proyecto práctico, como evaluación formativa que se valora con 3.40/3.40, lo que da un total de 10/10 para la calificación del módulo. Los parciales se califican en una escala de hasta 6.60 puntos, representando cada uno el 2.22 de la calificación total de 6.6 puntos. Para presentarse al proyecto final, el estudiante debe haber obtenido al menos 4.50 puntos sumando las tres primeras notas. En caso de no alcanzar este mínimo en el proyecto, se otorga una oportunidad

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

de recuperación dentro de las 48 horas laborables siguientes, según el calendario académico oficial. La nota mínima acumulada requerida para aprobar la asignatura es 7/10, y es esencial mantener al menos un 70% de asistencia a las clases. Los docentes deben informar a los estudiantes sobre sus notas individuales antes de registrarlas en el sistema, y se espera que los alumnos confirmen su aceptación y conformidad con estas calificaciones. Además, los docentes deben entregar un reporte de notas y asistencia a través del SGA y notificado a la coordinación de carrera y registrar las calificaciones en el sistema en un plazo máximo de 5 días posteriores a la recepción del proyecto final.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

8. UNIDADES

8.1 UNIDAD 1 (Introducción a los Ambientes Computacionales en la actualidad)

8.1.1 Temas y Subtemas

Tema 1 : Ambientes Locales y en la Web

Tema 2 : Interaccion de Servidores en el desarrollo de Ambientes computacionales

Tema 3 : Herramientas de desarrollo en los Ambientes computacionales.

El trabajo de cada desarrollador antes de ser un creador de código es ser un implementador y configurador de su ambiente de desarrollo, pero que es un ambiente de desarrollo como se debe realizar la configuración respectiva y la relación entre el ordenador y la persona, a menudo referida como Interacción Humano-Computadora (HCI, por sus siglas en inglés), comenzó a desarrollarse a medida que los ordenadores se volvieron más accesibles y capaces de realizar tareas diversas que requerían interacción directa con los usuarios. Nuestro papel fundamental es poder establecer los prerrequisitos que se necesitan para levantar un ambiente que permita que las instalaciones, configuraciones y adecuaciones de software se vean reflejadas en el resultado de la creación.

8.1.2 ***Ambientes computacionales: Una perspectiva referenciada***

El término "ambientes computacionales" abarca diversos contextos, cada uno con características y propósitos distintos. A continuación, se presentan algunas de las interpretaciones más comunes, con referencias para respaldar la información.

- Ambientes de desarrollo

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Se refieren al conjunto de herramientas y software utilizados para la creación, prueba y depuración de aplicaciones. Esto incluye editores de código, compiladores, depuradores, sistemas de control de versiones y entornos de desarrollo integrados (IDEs). (Editorial Etecé, 2023)

Estos ambientes pueden ser locales o remotos, y su configuración varía según el lenguaje de programación y el tipo de aplicación.

"La evolución de los IDEs ha estado impulsada por la necesidad de aumentar la productividad de los programadores y simplificar el desarrollo de software complejo" (Editorial Etecé, 2023)

En si se refieren al conjunto de herramientas y software que se utilizan para crear, probar y depurar aplicaciones estas incluyen:

- Editores de código (como VS Code, Sublime Text).
- Compiladores e intérpretes (como GCC, Python).
- Depuradores (como GDB, Xdebug).
- Sistemas de control de versiones (como Git).
- Entornos de desarrollo integrados (IDEs) (como Eclipse, IntelliJ IDEA).
- Estos ambientes pueden ser locales (en tu propia computadora) o remotos (en servidores en la nube).

Ambientes de ejecución

Se refieren al entorno en el que se ejecuta una aplicación, incluyendo sistemas operativos, servidores web, bases de datos y plataformas en la nube. La elección del ambiente de ejecución depende de los requisitos de la aplicación y puede influir en su rendimiento y escalabilidad.

"La computación en la nube ha revolucionado los ambientes de desarrollo, permitiendo la creación de entornos escalables y bajo demanda" (Lascano, 2003). Como ejemplos de los entornos de ejecución en el que se establece una o varias aplicaciones se encuentran:

- Sistemas operativos (como Windows, Linux, macOS).
- Servidores web (como Apache, Nginx, Laragon).
- Bases de datos (como MySQL, PostgreSQL).
- Plataformas en la nube (como AWS, Azure, GCP).

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Estos ambientes pueden variar según el tipo de aplicación y sus requisitos y va a depender mucho de ellos para poder tener un correcto uso o desempeño el software final.

Ambientes de aprendizaje computacionales

Se refieren a entornos diseñados para facilitar el aprendizaje de conceptos y habilidades relacionados con la computación, como simuladores, entornos de programación visual y plataformas de aprendizaje en línea.

Estos ambientes buscan proporcionar experiencias interactivas y prácticas para los estudiantes.

Computación ambiental (Ambient computing)

Se refiere a una tecnología que se integra de manera transparente en nuestro entorno cotidiano, anticipándose a nuestras necesidades y ofreciendo soluciones sin necesidad de órdenes explícitas.

El Internet de las cosas (IoT) es un ejemplo de computación ambiental.

*8.1.3 **Evolución del concepto de ambiente de desarrollo y Orígenes de los Ambientes Computacionales***

Prehistoria (antes de los 70):

En los primeros días de la computación, el "ambiente de desarrollo" era a menudo el mismo que el "ambiente de producción". Los programadores trabajaban directamente en las mismas máquinas que ejecutaban el software. Las herramientas eran rudimentarias:

- Tarjetas perforadas
- Cintas magnéticas
- Lenguajes de bajo nivel.

No existía una distinción clara entre desarrollo y operaciones por lo que el ingeniero de software era el encargado de realizar todas y cada una de las tareas para que una aplicación pueda realizar, antes no existía subdivisión de tareas el desarrollador también era el personal de IT que tenía que resolver como trabajar que tipos de variables y puertos se utilizan, etc.

Los 70 y 80: Surgimiento de la separación

A medida que el software se volvía más complejo, surgió la necesidad de separar los entornos de desarrollo de los de producción para poder tener un principio de mantenibilidad y

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

de desarrollo y parche. Se introdujeron conceptos como "entornos de prueba" para verificar el software antes de su implementación. Las herramientas mejoraron: editores de texto, compiladores y depuradores permitiendo tener un relación bidireccional que sea fructífera para el futuro y la mejora en lo que compete a la generación de código.

Los 90 y 2000: La era de los IDEs:

Los entornos de desarrollo integrados (IDEs) se volvieron comunes, proporcionando a los programadores un conjunto completo de herramientas en un solo lugar facilitando así el tema de configuración y manteniendo una herramienta completa, lo malo de este proceso es que existía la probabilidad de que un desarrollador se case o enganche con un único IDE por lo que esto no promulgaba y gran manejo de lenguajes y que con el criterio de mejora se creó la virtualización y comenzó a permitir la creación de entornos de desarrollo más flexibles y aislados al sistema operativo del ordenador.

El auge de internet y las aplicaciones web impulsó la necesidad de entornos de desarrollo más sofisticados y que puedan trabajar en forma distribuida y desacoplada por lo que los IDE fueron quedando poco a poco fuera de foco en cuanto a lo que tiene que ver con el desarrollo de aplicaciones de ultima vanguardia.

La era moderna (2010-presente): DevOps y la nube:

La filosofía DevOps enfatiza la colaboración y la automatización, difuminando aún más las líneas entre desarrollo y operaciones donde lo que se trata de resaltar en esta filosofía es mantener una generación y testear continuamente sobre el código para mantener una mejor manera de ejemplificar el continuo desarrollo y continua implementación. La computación en la nube ha revolucionado los ambientes de desarrollo, permitiendo la creación de entornos escalables y bajo demanda.

La forma de manejo por contenedores (Docker) y la orquestación (Kubernetes) han facilitado la creación de ambientes de desarrollo portátiles y consistentes. La idea de un "ambiente de desarrollo" ha evolucionado desde un concepto implícito hasta una práctica explícita y compleja y que maneja la separación de los entornos de desarrollo y producción ha sido un hito en el argot del desarrollo de software. Las herramientas y tecnologías han transformado radicalmente la forma en que se crean y gestionan estos ambientes computacionales. La evolución del ambiente de desarrollo va de la mano con la evolución de la industria del software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

En cuanto a la acuñación del término es difícil señalar un momento exacto en el que se haya utilizado la frase "ambiente de desarrollo". Es más preciso decir que el concepto se ha desarrollado gradualmente a medida que la industria del software ha madurado con el tiempo. Por lo cual se puede decir en el mundo de la programación, un ambiente de desarrollo será un espacio de trabajo configurado específicamente para permitir a los programadores escribir, probar y depurar código de manera eficiente.

Estos ambientes proporcionan las herramientas y recursos necesarios para crear aplicaciones de software de alta calidad. Pensemos que un ambiente de desarrollo a forma de ejemplo se lo puede considerar como un taller bien equipado donde los desarrolladores pueden crear, probar y mejorar su trabajo sin interrupciones.

*8.1.4 **¿Qué es un Ambiente de Desarrollo?***

Un ambiente de desarrollo es más que un simple conjunto de herramientas; es un ecosistema que facilita el proceso de programación desde la escritura del código hasta su prueba y depuración. Incluye elementos como editores de código, compiladores o intérpretes, depuradores y herramientas de prueba, todos integrados para que los desarrolladores puedan trabajar de manera fluida y organizada. Además, estos ambientes suelen incluir sistemas de control de versiones, como Git, que permiten a los equipos colaborar eficientemente en proyectos complejos.

*8.1.5 **Características Clave de un Ambiente de Desarrollo***

Los ambientes de desarrollo suelen tener tres niveles principales: desarrollo, integración y producción. En el nivel de desarrollo, los programadores escriben y prueban su código inicialmente. Luego, en el nivel de integración, se verifica la fiabilidad del programa antes de su lanzamiento final en el servidor de producción. Además, estos ambientes pueden incluir características avanzadas como autocompletado de código, resaltado de sintaxis y refactorización, lo que facilita el proceso de codificación y depuración. Posterior a la salida de los IDE del marco referencial de programación es que se utilizan técnicas tanto en el proceso de configuración, así como en el despliegue de herramientas desacoplados para poder generar un marco de instalación de cada componente permitiendo tener un ambiente local o virtual configurado.

*8.1.6 **Ventajas de Utilizar un Ambiente de Desarrollo***

Utilizar un ambiente de desarrollo bien configurado ofrece varias ventajas. Por un lado, mejora la eficiencia y productividad de los desarrolladores al proporcionar todas las herramientas necesarias en un solo lugar. Esto reduce el tiempo dedicado a configuraciones y

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

permite centrarse en la creación de software de calidad. Además, facilita la colaboración en equipo, ya que los cambios en el código pueden ser seguidos y revertidos si es necesario, gracias a sistemas de control de versiones como Git.

Ejemplos y Aplicaciones Prácticas

Existen muchos ejemplos de ambientes de desarrollo populares, como Eclipse, NetBeans y Visual Studio Code, cada uno con sus propias características y ventajas dependiendo del lenguaje de programación y las necesidades del proyecto. Estos ambientes no solo son útiles para programadores experimentados, sino también para estudiantes que buscan aprender y mejorar sus habilidades en la programación. Al entender cómo funcionan estos ambientes, los desarrolladores pueden crear software más rápido, más seguro y eficiente.

8.1.7 ***Hitos Importantes en la Historia de la Interacción IDE y Ambientes de Desarrollo***

Primeros pasos (antes de los 70):

Dartmouth BASIC (1964): Aunque no era un IDE completo, marcó un cambio al permitir a los programadores escribir, probar y ejecutar código en un solo entorno interactivo.

Esto sentó las bases para la interacción directa con las computadoras durante el desarrollo.

Edición de texto y compiladores: En esta época, el desarrollo se realizaba principalmente con editores de texto básicos y compiladores separados. El concepto de un "ambiente de desarrollo" era rudimentario, a menudo el mismo que el entorno de producción.

La era de los IDEs emergentes (70s-80s):

Surgimiento de herramientas integradas: A medida que el software se volvía más complejo, se hizo evidente la necesidad de herramientas más integradas. Comenzaron a aparecer editores de código con resaltado de sintaxis y depuradores básicos.

Turbo Pascal (1983): Este IDE de Borland fue un hito importante, ya que combinaba un editor, compilador y depurador en un entorno fácil de usar.

Popularizó la idea de un IDE como una herramienta de productividad para los programadores.

La consolidación de los IDEs (90s-2000s):

Visual Studio (Microsoft):

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Se convirtió en un IDE dominante, ofreciendo un conjunto completo de herramientas para el desarrollo en múltiples lenguajes. Introdujo características avanzadas como la finalización de código inteligente y la depuración visual.

Eclipse (IBM):

Un IDE de código abierto que ganó popularidad por su flexibilidad y extensibilidad.

Se convirtió en una plataforma para el desarrollo en Java y otros lenguajes.

NetBeans (Oracle):

Otro IDE de código abierto que ha sido muy popular para java y otros lenguajes de programación.

La era moderna (2010-presente):

IDE´s ligeros y multiplataforma:

Editores de código como VS Code ganaron popularidad por su ligereza, velocidad y soporte para múltiples lenguajes y plataformas. La nube y el proceso de contenedores han transformado los ambientes de desarrollo, permitiendo la creación de entornos portátiles y consistentes.

IDE´s en la nube:

Han surgido IDE´s basados en la nube, que permiten a los desarrolladores trabajar desde cualquier lugar con conexión a internet esto ha facilitado la colaboración y la creación de ambientes de desarrollo bajo demanda. De igual forma a promulgado el uso del modelo SaaS, donde SaaS es un modelo de distribución de software en el que las aplicaciones se alojan en la nube y se ponen a disposición de los usuarios a través de Internet. En lugar de instalar y mantener el software en sus propios servidores, los usuarios acceden a él a través de un navegador web o una aplicación móvil. Estos hitos muestran la evolución de la interacción ambientes computacionales y los desarrolladores, desde métodos rudimentarios hasta, modelos avanzados que permiten una interacción más natural e intuitiva gracias al uso de la tecnología en pro de mejorar la experiencia del usuario interno en este caso de los diseñadores, desarrolladores y personal relacionado con la creación de sistemas que promueven el manejo de herramientas digitales para la gestión de proyectos.

*8.1.8 **Iteración de los ambientes computacionales en la actualidad***

En el mundo del desarrollo de software, los ambientes de desarrollo representan el eje angular para la creación de aplicaciones modernas. Estos espacios de trabajo están diseñados

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

para permitir a los desarrolladores crear, probar y depurar aplicaciones de manera eficiente. Piensa un momento en un ambiente de desarrollo como un taller bien equipado donde los desarrolladores pueden crear, mejorar y perfeccionar su trabajo sin interrupciones.

Al igual que un artesano necesita un espacio organizado para trabajar, un desarrollador necesita un ambiente de desarrollo bien configurado para producir software de alta calidad, que este a su vez pueda realizar su día a día con un orden prolijo que le permite evidenciar, testear y comprobar sus creaciones, así como resolver y solventar cualquier impase en el o los sistemas.

Evolución de la Interacción de los Ambientes de Desarrollo

La iteración y evolución de los ambientes de desarrollo han sido constantes a lo largo de los años. Desde los primeros lenguajes de programación interactivos hasta los entornos de desarrollo integrados (IDE) modernos, estos ambientes han mejorado significativamente la forma en que los desarrolladores trabajan. Por ejemplo, herramientas como Git han revolucionado la colaboración en equipo, permitiendo a los desarrolladores trabajar juntos en proyectos complejos de manera eficiente. Además, la integración y el despliegue continuos (CI/CD) han acelerado el proceso de llevar aplicaciones desde el desarrollo hasta el proceso de implementación y producción.

Clasificación de los Ambientes de Desarrollo

Los ambientes de desarrollo se pueden clasificar en diferentes niveles, desde el desarrollo local hasta la producción. En el nivel de desarrollo, los programadores escriben y prueban su código inicialmente. Luego, en el nivel de integración, se verifica la fiabilidad del programa antes de su lanzamiento final en el servidor de producción. Además, estos ambientes pueden incluir características avanzadas como autocompletado de código, resaltado de sintaxis y refactorización, lo que facilita el proceso de codificación y depuración. Por ejemplo, IDE´s como Visual Studio Code o IntelliJ IDEA ofrecen una amplia gama de herramientas para mejorar la productividad del desarrollador. Utilizar un ambiente de desarrollo bien configurado ofrece una serie de ventajas. Por un lado, mejora la eficiencia y productividad de los desarrolladores al proporcionar todas las herramientas necesarias en un solo lugar.

Esto reduce el tiempo dedicado a configuraciones y permite centrarse en la creación de software de calidad. Además, facilita la colaboración en equipo, ya que los cambios en el código pueden ser seguidos y revertidos si es necesario, gracias a sistemas de control de versiones como Git. Esto no solo ayuda a mantener el código organizado, sino que también reduce los errores y mejora la calidad general del software un punto de vista que deben tomar en cuenta los estudiantes de desarrollo es que el entender cómo funcionan los ambientes de desarrollo

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

garantizara que puedan ubicarse en cualquier región, medio o país para así crear software robusto y escalable.

Al familiarizarse con herramientas como Git, Docker y diferentes IDE´s, los desarrolladores pueden mejorar su capacidad colaborativa y productiva para la generación de aplicaciones de alta demanda y con una visión más extensa del ambiente laboral. Además, es importante explorar las mejores prácticas en el desarrollo de software, como la integración y el despliegue continuos, para acelerar el proceso de llevar aplicaciones desde el desarrollo hasta la producción. Al final, un ambiente de desarrollo bien configurado no solo es una herramienta, sino una parte integral del proceso creativo y colaborativo que define al desarrollo de software moderno.

Ejemplo Grafico de Entornos de Desarrollo:

En este proceso de los entornos se publican los cambios y los usuarios pueden verlos. Por ejemplo, si en EDteam añadiéramos una nueva característica a la plataforma, luego de meses de pasar por las pruebas y que llegue hasta los usuarios, es que decimos que está en producción. Si quieres garantizar la calidad de tu software, es mejor que comiences a pasar por todas estas etapas. Además, existe algo llamado CI/CD (integración continua o despliegue continuo) que permite pasar por todas las etapas de manera automatizada. proyectos.

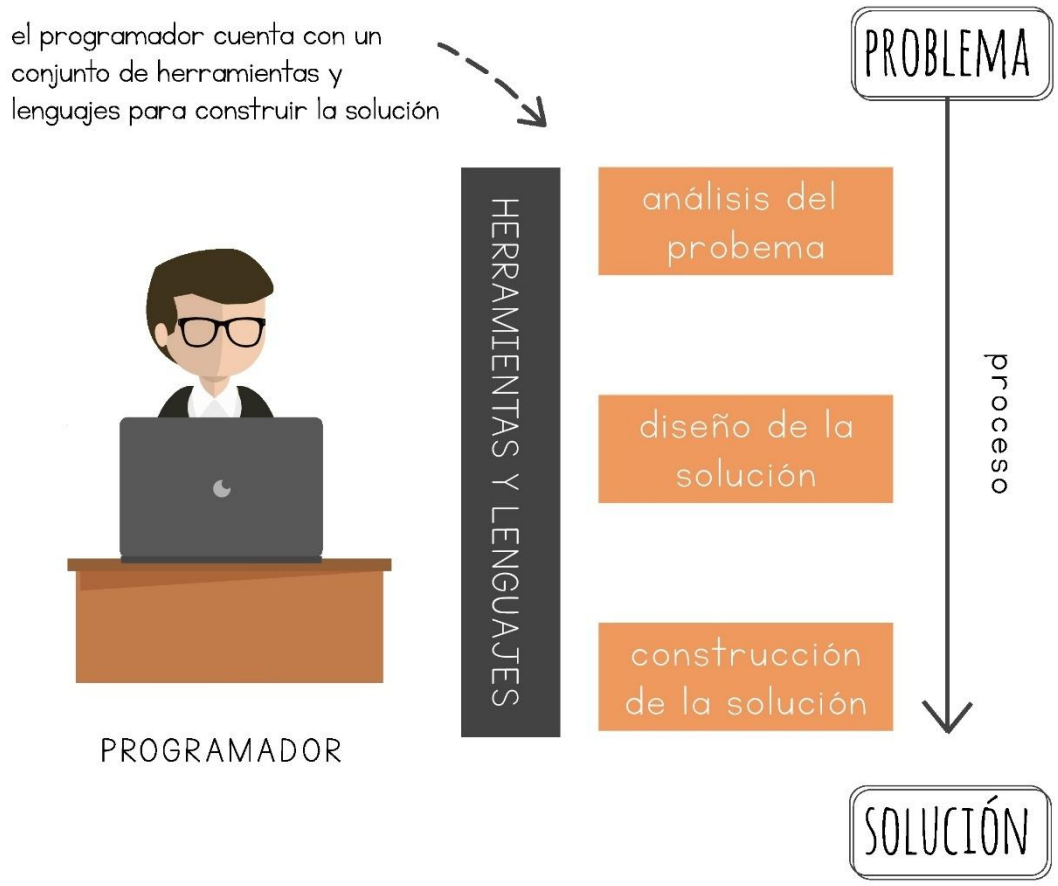
Figura 1
Diseño de Ambiente o Entornos de Desarrollo (Drafts)

Nota: *Proceso de entornos de desarrollo en el software actual.* ***Fuente***: *(Yummerlys, 2022)*

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

En la Figura 1 podemos ver una representación en Drafts que permite comprender como funcionan los ambientes computacionales en la actualidad y como los desarrolladores manejan cada uno de estos ambientes permitiéndoles tener una organización dentro de su trabajo.

Figura 2
Proceso de flujo de resolución de problemas en el desarrollo

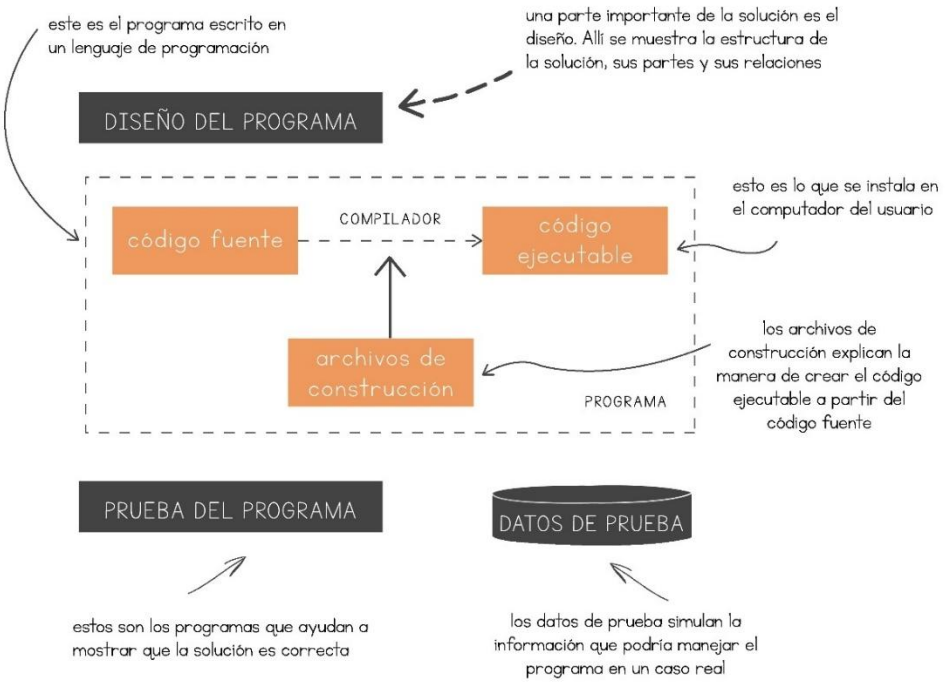


Nota: *Simulación del proceso que va acorde con los ambientes disponibles.* ***Fuente:*** *Autor*

En la Figura 2 podemos ver como el desarrollador tiene que ubicar de forma perpendicular las herramientas o lenguajes acordes a los pasos para la resolución del problema, esto independientemente de la herramienta de trabajo que se haya utilizado con esto el desarrollador será el encargado de levantar su ambiente para la resolución del problema planteado, cabe recalcar que este es solo un prototipo del proceso y este puede manejarse de otra forma dado que dependerá mucho de la empresa de desarrollo o el área de TI que es la encargada a veces de gestionar las aplicaciones que van instaladas en los equipos para el desarrollo correcto.

Figura 3
Herramientas y sus componentes

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC



Nota: Ejemplo del proceso de un compilador cuando se realice el desarrollo. **Fuente:** Autor

En la Figura 3 podemos ver las configuraciones de la herramienta funcionan internamente para desplegar un programa:

Existen muchos tipos de lenguajes de programación, entre los cuales los más utilizados en la actualidad son los llamados lenguajes de programación orientada a objetos. En este libro utilizaremos Java que es un lenguaje orientado a objetos muy difundido y que iremos presentando poco a poco, a medida que vayamos necesitando sus elementos para resolver problemas.

Un programa es la secuencia de instrucciones (escritas en un lenguaje de programación) que debe ejecutar un computador para resolver un problema.

El tercer elemento de la solución son los archivos de construcción del programa. En ellos se explica la manera de utilizar el código fuente para crear el código ejecutable. Este último es el que se instala y ejecuta en el computador del usuario.

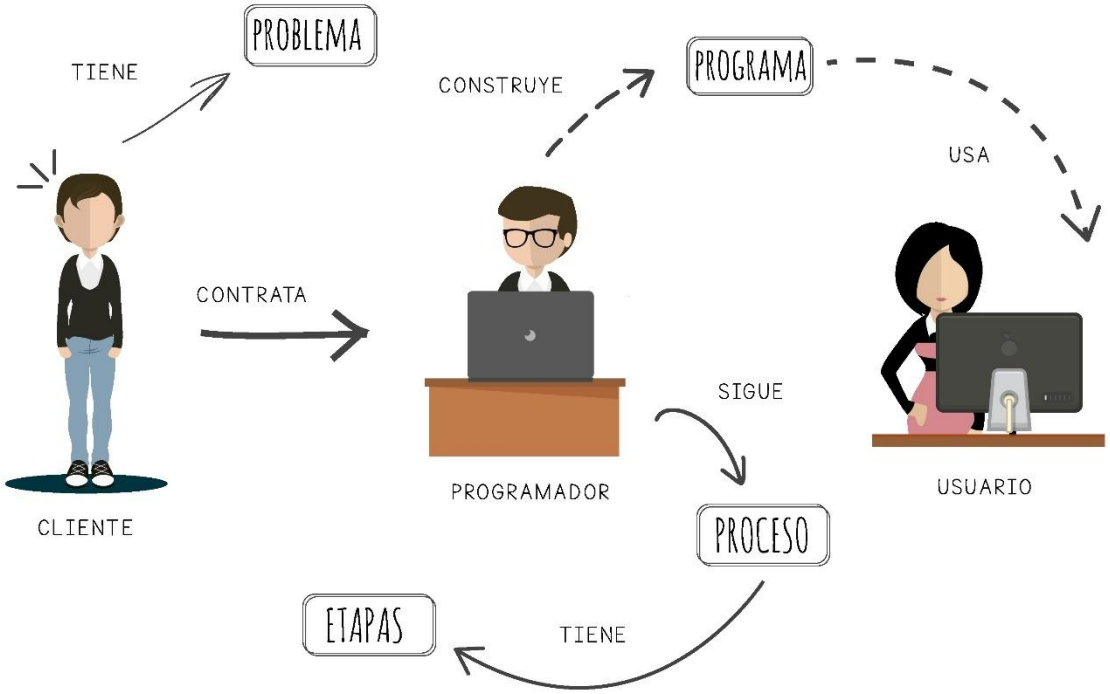
El programa que permite traducir el código fuente en código ejecutable se denomina compilador. Antes de poder construir nuestro primer programa en cualquier ambiente de desarrollo o lenguaje, por ejemplo, tendremos que conseguir el respectivo compilador del lenguaje.

Todo este entorno representa nuestra área de trabajo definida en un marco el mismo puede ser de un equipo personal (laptop, desktop, Tablet) y también podemos escoger otros

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

tipos de marcos de trabajo para la gestión de ambientes computacionales siempre evidenciando la necesidad de que el desarrollo de aplicaciones web, móviles y cliente servidor se ajusten a estas configuraciones.

Figura 4
Flujo de trabajo de un Desarrollador – Equipo de Trabajo

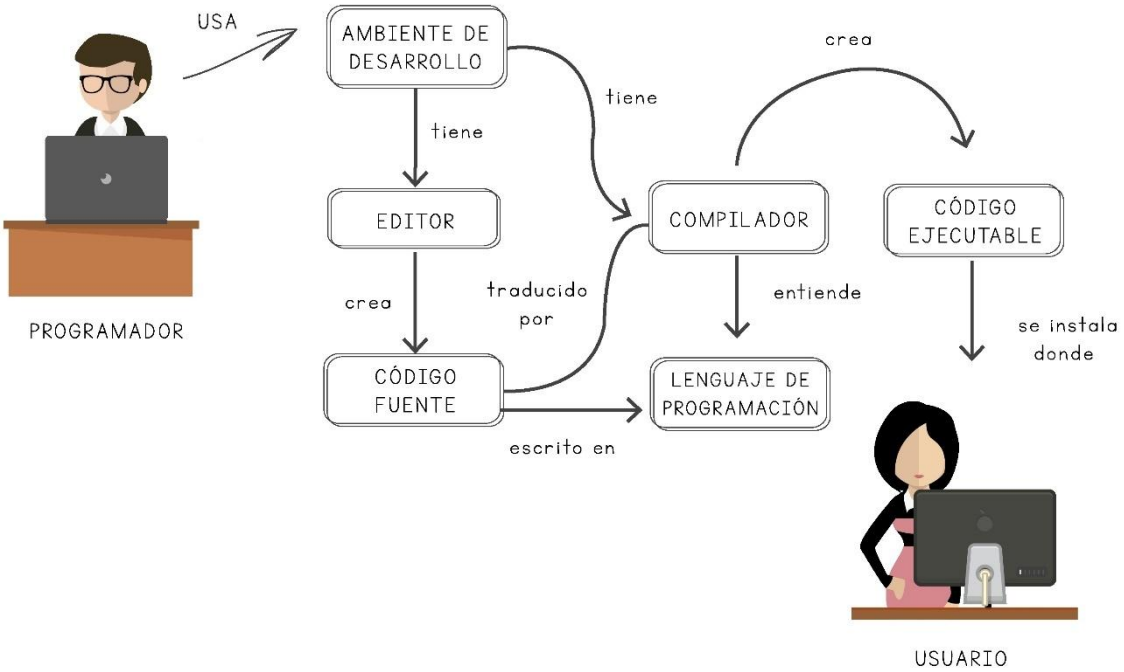


Nota: *Flujo de trabajo y etapas de desarrollo.* ***Fuente:*** *Autor*

En la Figura 4 podremos revisar las etapas del desarrollo en el mismo se pueden ver cómo funciona el flujo general, a su vez este se puede interpretar por los usuarios, y los estudiantes donde se recuerda que en la parte inferior de la imagen que existen etapas definidas para cada proceso, esto ya sea en base a una metodología de desarrollo actual. Es imprescindible que se elija esta metodología que permita administrar y gestionar estas etapas del desarrollo de software.

Figura 5
Proceso técnico de Desarrollo de Software (Drafts).

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC



Nota: *Flujo de trabajo y etapas de desarrollo.* ***Fuente:*** *Autor*

Los mapas mentales deben explicar el por qué generamos un feedback que retroalimentara el proceso de desarrollo al agregar una forma más intuitiva de comprender este proceso, donde se sugiere incluir cada etapa y categoría que permita facilitar la búsqueda de proyectos mediante un ambiente de desarrollo acorde a las necesidades del proyecto.

Entorno de desarrollo

Se trata de la computadora local en la que trabaja cada programador. Cada desarrollador tiene una copia del proyecto en su computador, es decir, un clon del repositorio en dónde hacen los cambios necesarios para luego enviarlos al repositorio central. (EDTeam, 2022). Ajustar el proceso de un equipo de desarrollo proporcionando una experiencia de uso mejorada y más eficiente mediante las herramientas o entornos previstos. Este proceso iterativo no solo garantiza que el producto final cumpla con las necesidades de los desarrolladores para continuar creando, sino que también permite al equipo de desarrollo abordar problemas y mejoras de manera proactiva para obtener un mejor entorno.

Entorno de integración

En esta etapa, los cambios de todos los programadores tienen que unirse sin que se rompa el software. Para eso, existen pruebas automatizadas que evitan que los cambios que generen conflictos sean aprobados. (EDTeam, 2022)

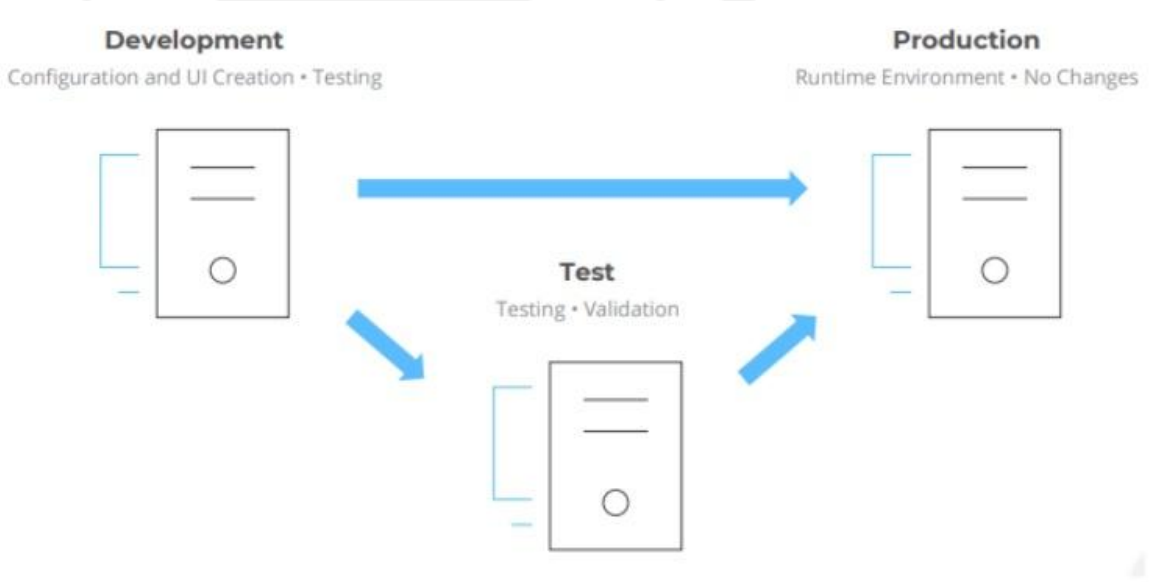
(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Después de iterar y ajustar, el proceso de desarrollo e implementación en la aplicación, proporcionando una experiencia de usuario mejorada y más eficiente. Este proceso iterativo no solo garantiza que el producto final cumpla con las necesidades, sino que también permite definir los estándares que permitirán abordar problemas y mejoras de manera proactiva, basándose en datos reales y generando una cultura de aseguramiento de la calidad, antes que se suban cambios q no se encuentren completamente revisados similar a el proceso de GIT.

Entornos de pruebas

Ahora, en este entorno, se prueba el software como si fueras un usuario. Es decir, interactúas con la aplicación para ver si funciona como debería. También existe un software que permite automatizar estas pruebas.

Figura 6
Presentación de ambientes de una empresa de desarrollo para la gestión de proyectos en FIGMA



Nota: *Granja de servidores de una empresa de desarrollo.* ***Fuente:*** *Autor*

En la Figura 6 podemos evidenciar el resultado de levantar servidores expuestos en el ejemplo anterior planteaba manejar pruebas, la idea es tener una web que permita gestionar los proyectos de amanera eficiente, además la herramienta proporcione la visión necesaria del alcance final mediante un software que pueda implementar las soluciones tanto en el server de pruebas, así como en el de desarrollo.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

*8.1.9 **Iteración de Ambientes de Desarrollo***

La iteración de ambientes de desarrollo se refiere al proceso continuo de evolución y mejora de estos entornos. A lo largo de la historia, hemos pasado de ambientes simples y rudimentarios a entornos complejos y sofisticados. (Martin, 2021)

Ambientes Tradicionales:

Inicialmente, los desarrolladores trabajaban directamente en las máquinas de producción.

- Las herramientas eran básicas: editores de texto y compiladores.
- La separación entre desarrollo y operaciones era mínima.

Ambientes Modernos:

La virtualización y la contenerización (Docker) han revolucionado los ambientes de desarrollo.

Los IDE´s (Entornos de Desarrollo Integrados) proporcionan herramientas completas en un solo lugar. (Martin, 2021)

- La nube permite crear ambientes escalables y bajo demanda.
- La filosofía DevOps promueve la colaboración y la automatización.

Clasificación de Ambientes de Desarrollo

Los ambientes de desarrollo se pueden clasificar de varias maneras, según su propósito y características:

Ambientes Locales:

- Se encuentran en la computadora del desarrollador.
- Permiten un control total y una rápida iteración.

Herramientas como XAMPP y Laragon facilitan la creación de ambientes locales.

Ambientes Remotos:

- Se encuentran en servidores remotos, ya sea en la nube o en servidores propios.
- Permiten la colaboración y el acceso desde cualquier lugar.
- Plataformas como AWS, Azure y GCP ofrecen ambientes remotos escalables.

Ambientes de Desarrollo Integrados (IDE’s):

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

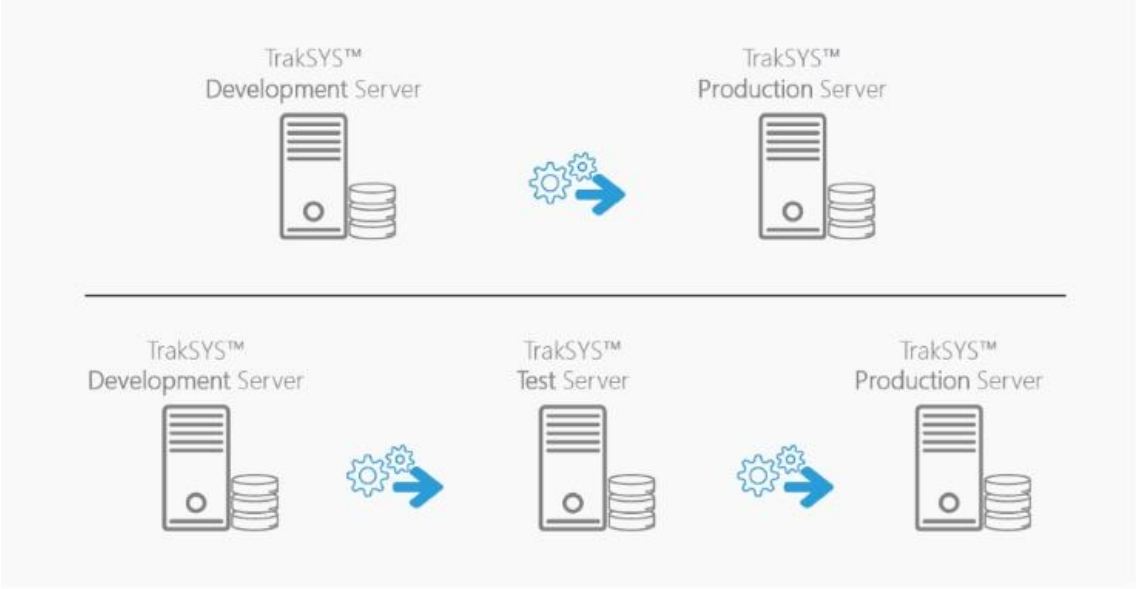
Proporcionan un conjunto completo de herramientas para el desarrollo como ejemplos: Visual Studio, Eclipse, IntelliJ IDEA son los encargados de facilitar la escritura, prueba y depuración de código. (Editorial Etecé, 2023)

Ambientes de Contenerización:

Utilizan contenedores (Docker) para crear ambientes aislados y portátiles que aseguran la consistencia entre diferentes entornos y complementan la implementación y el escalado de aplicaciones. (EDTeam, 2022)

Al comprender la iteración y clasificación de ambientes de desarrollo es útil para los estudiantes de desarrollo ya que les permite elegir las herramientas y tecnologías adecuadas para sus proyectos permitiendo optimizar el flujo de trabajo y aumentar su productividad. Los ambientes de desarrollo han evolucionado significativamente a lo largo de la historia, adaptándose a las necesidades cambiantes de la industria del software. La clasificación de estos ambientes proporciona una estructura para comprender sus diferentes tipos y propósitos.

Figura 7
Modelo de actualización e interacción de aplicaciones.



Nota: *Se representa a una forma implementar software que se alinean más estrechamente con las necesidades de la empresa.* ***Fuente:*** *Autor*

En la Figura 7 podremos evidenciar cómo funcionaba antes de implementarse un servidor de pruebas el paso a producción era directo lo cual generaba inconvenientes dado que pueden enviarse errores a producción.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Por lo tanto el desarrollo de software y sus respectivas etapas parten de un proceso de creación y luego que puede tomar cierto tiempo relativamente aún no definido, para su implementación pero mediante estas herramientas podemos pre evaluar el producto final, si bien este modelo es funcional por completo permite tener la perspectiva menos expuesto a errores y sobre el modelo de implementación estará más cerca y a la mano del área de desarrollo de software todos son parte de este proceso, a continuación se enumeran una lista de las herramientas más utilizadas para realizar tareas de implementación , mantenimiento y seguimiento revisaremos sus respetivas características , ventajas y desventajas:

Git

Git es un sistema de control de versiones distribuido, esencial para el seguimiento de cambios en el código fuente. Permite la colaboración eficiente entre desarrolladores, la creación de ramas y la gestión de versiones. (Atlassian, 2024)

Características:

- Control de versiones distribuido.
- Gestión de ramas y fusiones.
- Historial de cambios detallado.
- Comunidad amplia y soporte extenso.

Ventajas:

- Facilita la colaboración y el seguimiento de cambios.
- Permite la creación de ramas para el desarrollo paralelo.
- Ofrece un historial completo de cambios.

Desventajas:

- La línea de comandos puede ser intimidante para principiantes.
- Los conflictos de fusión pueden ser complejos.

Jenkins

Jenkins es un servidor de automatización de código abierto que permite la integración y entrega continua. Es altamente configurable y extensible mediante plugins. (Atlassian, 2024)

Características:

- Gran cantidad de plugins disponibles.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Flexibilidad para personalizar pipelines.
- Soporte para múltiples lenguajes y plataformas.

Ventajas:

- Altamente personalizable y extensible.
- Gran comunidad y soporte.
- Soporte para una amplia gama de herramientas.

Desventajas:

- La configuración puede ser compleja.
- Requiere mantenimiento y administración.
- Puede consumir muchos recursos.

GitLab CI/CD

GitLab CI/CD está integrado en la plataforma GitLab y permite la automatización de pipelines de CI/CD. Utiliza archivos “gitlab-ci.yml” para definir los pipelines. (Atlassian, 2024)

Características:

- Integración directa con GitLab.
- Facilidad de configuración de pipelines.
- Soporte para contenedores Docker.

Ventajas:

- Integración perfecta con GitLab.
- Facilidad de uso y configuración.
- Funcionalidades CI/CD muy completas.

Desventajas:

- Dependencia de la plataforma GitLab.
- Menor flexibilidad en comparación con Jenkins.

GitHub Actions

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

GitHub Actions es la plataforma de CI/CD nativa de GitHub, que permite la automatización de workflows se encarga de utilizar archivos .yml para definir los workflows. (Atlassian, 2024)

Características fuertes:

- Integración directa con GitHub.
- Gran comunidad y mercado de acciones.
- Flexibilidad para crear workflows personalizados.

Ventajas:

- Integración perfecta con GitHub.
- Gran cantidad de acciones disponibles.
- Facilidad de uso y configuración.

Desventajas:

- Menor flexibilidad en comparación con Jenkins.
- Dependencia de la plataforma GitHub.

8.1.10 ***Autoevaluación 1***

- ¿Cuál es el propósito principal de un ambiente de desarrollo?

Ejecutar aplicaciones en producción

Facilitar la escritura, prueba y depuración de código

Servir como un sistema operativo independiente

Almacenar grandes volúmenes de datos

Big data

Datos

- ¿Qué herramienta se usa comúnmente para el control de versiones en desarrollo de software?

Jenkins

Git

Kubernetes

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Docker

C#

PHP

- ¿Qué tecnología ha revolucionado los ambientes de desarrollo al permitir la creación de entornos escalables y bajo demanda?

Computación en la nube

Inteligencia artificial

Redes neuronales

Algoritmos genéticos

Procesos

Pasos de programación

- ¿Cuál de las siguientes NO es una característica de un ambiente de desarrollo?

Autocompletado de código

Resaltado de sintaxis

Interfaz gráfica obligatoria

Control de versiones

Dataminig

Proceso de autocontrol

- ¿Cuál de las siguientes es una ventaja de usar un IDE?

Requiere menos memoria en comparación con editores de texto simples

Proporciona herramientas integradas para facilitar el desarrollo

No permite el uso de sistemas de control de versiones

Hace que los programas se ejecuten más rápido

Interfaz de desarrollo

Entorno de Dev

- ¿Qué tipo de ambientes de desarrollo se utilizan principalmente para colaborar en proyectos desde cualquier lugar?

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ambientes locales

Ambientes remotos

Ambientes offline

Ambientes de baja latencia

Ambiente local

Ambiente externo

- ¿Cuál de los siguientes es un ejemplo de un entorno de ejecución?

Visual Studio Code

MySQL

Sublime Text

GitHub

SQLLite

SQLServer

- ¿Cuál es el propósito de los contenedores como Docker en el desarrollo de software?

Ejecutar múltiples sistemas operativos en la misma máquina

Crear ambientes de desarrollo portátiles y consistentes

Reemplazar los IDEs tradicionales

Facilitar el diseño gráfico en aplicaciones web

Concatenación

Revisión

- ¿Qué caracteriza a la filosofía DevOps?

Separación estricta entre desarrollo y operaciones

Eliminación de pruebas automatizadas

Colaboración y automatización en desarrollo y operaciones

Uso exclusivo de software propietario

Filantrópica

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Obtención de datos y procesos

- ¿Qué representan los entornos de pruebas en el desarrollo de software?

El entorno donde se ejecuta el software en producción

Un entorno donde se prueba el software antes de su implementación

Un sistema de respaldo para código fuente

Un entorno exclusivo para pruebas de hardware

Entorno de revisión.

Proceso de revisión de desarrollo.

8.1.11 ***Resumen de la Unidad 1***

En la presente unidad se exponen que antes de convertirse en creadores de código, los desarrolladores deben desempeñarse como implementadores y configuradores de su entorno de trabajo. Se destaca la importancia de configurar adecuadamente el ambiente de desarrollo, lo que no solo implica instalar y ajustar herramientas, sino también definir la interacción entre el ser humano y el computador, conocida como Interacción Humano-Computadora (HCI). Esta interacción ha evolucionado a medida que los ordenadores se han vuelto más accesibles y potentes, permitiendo una relación más directa y eficiente entre el usuario y la máquina. El concepto de "ambiente de desarrollo" se amplía para incluir tanto las herramientas y software que facilitan la creación, prueba y depuración de aplicaciones, como los entornos donde se ejecutan dichas aplicaciones.

La idea central es que un ambiente de desarrollo bien configurado es el pilar que soporta la eficiencia, calidad y escalabilidad del software, siendo esencial para la productividad de los equipos y para la formación de nuevos desarrolladores. El documento define el ambiente de desarrollo como un ecosistema integral que incluye:

Herramientas de Desarrollo:

- Editores de código: Herramientas como Visual Studio Code o Sublime Text que permiten escribir y editar el código de forma eficiente.
- Compiladores e intérpretes: Programas que traducen el código fuente a lenguaje máquina o que lo interpretan directamente (por ejemplo, GCC para C/C++ o Python para lenguajes interpretados).

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Depuradores: Herramientas que facilitan la detección y corrección de errores en el código (como GDB o Xdebug).
- Sistemas de control de versiones: Software como Git que permite el seguimiento y manejo de cambios en el código, facilitando la colaboración en equipo.

Ambientes de Ejecución:

Se refieren al entorno en el que la aplicación se ejecuta, abarcando sistemas operativos, servidores web, bases de datos y plataformas en la nube (por ejemplo, Apache, MySQL, AWS, etc.). La elección de un ambiente de ejecución adecuado puede afectar directamente el rendimiento y escalabilidad del software.

Entornos de Aprendizaje:

Se explican también los ambientes dedicados al aprendizaje, que incluyen simuladores, entornos de programación visual y plataformas interactivas que facilitan el proceso de formación en computación.

Computación Ambiental (Ambient Computing):

Este concepto se relaciona con tecnologías que se integran de forma transparente en el entorno cotidiano, anticipándose a las necesidades del usuario, como ocurre en aplicaciones del Internet de las Cosas (IoT).

Se enfatiza que estos componentes no solo deben estar presentes, sino integrados de manera coherente para que el proceso de desarrollo sea fluido y adaptable a las exigencias actuales del mercado y de la educación en desarrollo de software. En la presente unidad se clasifica los ambientes de desarrollo según diversos criterios, lo que resulta muy útil tanto para la práctica profesional como para la enseñanza. Las categorías principales son:

Ambientes Locales:

Son aquellos configurados en la computadora personal del desarrollador. Permiten un control total sobre el entorno y son ideales para pruebas rápidas y desarrollo individual. Herramientas como XAMPP o Laragon facilitan la creación de estos ambientes.

Ambientes Remotos:

Estos se encuentran alojados en servidores remotos, ya sea en la nube o en infraestructura propia. Ofrecen ventajas en términos de colaboración y accesibilidad, ya que el

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

equipo de desarrollo puede trabajar desde cualquier ubicación. Plataformas como AWS, Azure y Google Cloud Platform son ejemplos de ambientes remotos escalables.

Ambientes Integrados (IDE's):

Los IDE's representan ambientes completos que integran herramientas de edición, compilación, depuración y control de versiones en una sola interfaz. Ejemplos destacados incluyen Visual Studio Code, IntelliJ IDEA y Eclipse, que se adaptan a diferentes lenguajes y necesidades de desarrollo.

Ambientes de Contenerización:

Con el auge de la virtualización, la contenerización se ha convertido en una práctica común para asegurar la portabilidad y consistencia entre diferentes entornos. Docker es el ejemplo paradigmático, permitiendo que las aplicaciones se ejecuten de la misma manera sin importar el entorno subyacente.

Esta clasificación ayuda a los estudiantes y profesionales a identificar y seleccionar el ambiente adecuado según el tipo de proyecto y los requerimientos específicos, fomentando una mayor eficiencia y robustez en el desarrollo de software. Para los estudiantes de desarrollo, conocer el funcionamiento y la configuración de estos ambientes es fundamental. Un entorno bien definido no solo mejora la experiencia de aprendizaje, sino que también prepara a los futuros desarrolladores para enfrentar desafíos reales en el ámbito profesional.

Ejemplos Prácticos y Herramientas Relevantes:

Git: Es un sistema de control de versiones distribuido esencial para la gestión de cambios en el código. Se destaca su capacidad para trabajar de manera colaborativa y gestionar ramas y fusiones de código, aunque se menciona que la línea de comandos puede ser un desafío para principiantes.

Jenkins: Esta herramienta de automatización de código abierto facilita la integración y entrega continua (CI/CD). Su flexibilidad y gran cantidad de plugins lo convierten en una opción robusta, aunque su configuración puede resultar compleja y consumir recursos.

GitLab CI/CD y GitHub Actions: Estas plataformas integradas en GitLab y GitHub, respectivamente, permiten la automatización de workflows mediante archivos de configuración (.yml). Aunque cada una presenta ventajas particulares en términos de integración y facilidad de uso, también tienen limitaciones en flexibilidad en comparación con soluciones como Jenkins. Cada una de estas herramientas se adapta a distintos escenarios y necesidades, permitiendo a los equipos de desarrollo estructurar sus procesos de manera lógica y concisa. La

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

correcta elección e integración de estas tecnologías es fundamental para garantizar que el ambiente de desarrollo respalde de forma óptima la creación, prueba y despliegue de software.

Integración y Mejora Continua

Otro aspecto importante que se aborda es la iteración constante en la configuración de los ambientes de desarrollo. Se enfatiza que la mejora continua es un proceso que abarca desde la selección de herramientas hasta la adopción de nuevas tecnologías que optimicen el flujo de trabajo. La filosofía DevOps, por ejemplo, ha difuminado las fronteras entre el desarrollo y las operaciones, promoviendo una cultura de colaboración y automatización que resulta esencial en la era moderna del desarrollo de software.

Además, en la unidad se realiza representaciones gráficas (figuras) que ilustran el flujo de trabajo, la interacción entre diferentes ambientes y el proceso de resolución de problemas en el ciclo de desarrollo. Estas representaciones son útiles tanto para la enseñanza como para la implementación práctica, ya que permiten visualizar de manera clara las etapas del proceso y cómo se integran las distintas herramientas.

En síntesis, el documento ofrece una visión integral de los ambientes de desarrollo, abarcando desde sus orígenes y evolución histórica hasta las prácticas modernas que involucran la nube, la contenerización y la automatización de procesos. Se destaca la importancia de contar con un ambiente bien configurado para mejorar la eficiencia, reducir errores y fomentar una colaboración más efectiva en el desarrollo de software.

Para docentes y profesionales esta sinopsis brindara diversas opciones didácticas y metodológicas para enseñar a los estudiantes sobre la configuración y gestión de ambientes de desarrollo. Se puede emplear ejemplos prácticos, casos de estudio y representaciones gráficas para ilustrar la evolución histórica, la clasificación y los procesos involucrados en la creación y mantenimiento de estos entornos.

Además, la comprensión de herramientas como Git, Jenkins, Docker y los IDEs actuales se convierte en un eje fundamental para estructurar cursos y talleres que preparen a la nueva generación de desarrolladores. En el ámbito académico, estructurar los procesos en etapas claramente definidas (desarrollo, integración, pruebas y producción) permite a los estudiantes tener una visión sistémica del ciclo de vida del software, fomentando buenas prácticas y la adopción de metodologías ágiles y orientadas a la calidad.

Finalmente, la integración de conceptos como la interacción humano-computadora y el uso de ambientes computacionales ofrece una perspectiva ampliada sobre cómo la tecnología se adapta y evoluciona para responder a las necesidades del usuario final, enriqueciendo así el

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

contenido curricular y preparando a los futuros desarrolladores para afrontar los desafíos que se afronta en el mundo real.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

8.2 UNIDAD 2 (Git: todo el poder de administrar tus proyectos.)

*8.2.1 **Temas y Subtemas***

*8.2.2 **Historia y Origen de GIT***

Creación del Término "GIT" de Linux Torvalds: En si fue creado por la necesidad de manejar de forma coherente y ordena los cambios que se van realizando a medida que se va avanzando con el software. Aunque Torvalds es conocido por su sentido del humor, y "Git" es una palabra corta y simple, fácil de recordar. Él mismo ha dicho que el nombre no tiene un significado profundo, y que lo eligió porque sonaba bien. En inglés coloquial, "git" puede tener una connotación negativa, ya que su significando es algo así como "idiota" o "inútil". (Chacon & Straub, 2024)

Torvalds ha hecho referencia a esto, diciendo que él es un "git". Nada más desligado de la realidad por todas sus contribuciones. Esto puede interpretarse como una forma de restar importancia a la herramienta, o como una broma autocrítica. En si el nombre "Git" fue elegido por Linus Torvalds por su simplicidad y, posiblemente, por su connotación coloquial en inglés. No tiene un significado técnico específico.

*8.2.3 **Preámbulo: El Caos del Control de Versiones Centralizado***

Antes de Git, el control de versiones estaba dominado por sistemas centralizados como Subversion (SVN) y CVS. Estos sistemas almacenaban el historial de versiones en un servidor central, lo que presentaba varios problemas:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Punto único de fallo: Si el servidor central fallaba, todo el historial de versiones se perdía.
- Rendimiento lento: Las operaciones requerían comunicación con el servidor central, lo que podía ser lento, especialmente para proyectos grandes.
- Flujo de trabajo limitado: Los flujos de trabajo de ramificación y fusión eran complejos y propensos a errores.

Por lo que antes de la llegada de Git, el control de versiones de software era un panorama diverso y a menudo complicado. Los desarrolladores enfrentaban desafíos significativos para rastrear cambios, colaborar en proyectos y mantener la integridad del código. Aquí hay un resumen de los sistemas y prácticas que prevalecieron antes de Git:

8.2.4 ***Sistemas de Control de Versiones Centralizados (CVCS)***

Estos sistemas, como CVS (Concurrent Versions System) y Subversion (SVN), utilizaban un repositorio centralizado para almacenar todas las versiones del código. Los desarrolladores trabajaban en copias locales, pero los cambios debían confirmarse en el repositorio central. Esto facilitaba la administración y el control, pero también creaba un punto único de falla.

Si el servidor central se caía, los desarrolladores no podían confirmar cambios ni acceder al historial esto a su vez afectaba las operaciones ya que podían ser lentas, especialmente para proyectos grandes o con conexiones de red deficientes. La creación de ramas (branches) para el desarrollo paralelo era posible, pero a menudo complicada y propensa a conflictos. (Chacon & Straub, 2024)

Ejemplos:

CVS (Concurrent Versions System): Uno de los primeros sistemas de control de versiones ampliamente utilizados. Era popular en la década de 1990 y principios de la de 2000.

Subversion (SVN): Diseñado para ser un reemplazo de CVS, SVN ofrecía mejoras en rendimiento y funcionalidad. Fue ampliamente adoptado antes de la popularización de Git.

Otros Métodos y Prácticas

8.2.5 ***Archivos de parches:***

Antes de los CVCS, los desarrolladores a menudo intercambiaban cambios mediante archivos de parches (patches). Estos archivos contenían las diferencias entre versiones de archivos. Este método era engorroso y propenso a errores, especialmente para proyectos grandes.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Copias de seguridad locales:

Los desarrolladores a menudo mantenían copias de seguridad locales de sus archivos. Esto proporcionaba cierta protección contra la pérdida de datos, pero no facilitaba la colaboración ni el seguimiento de cambios.

Sistemas de control de versiones propietarios:

Algunas empresas utilizaban sistemas de control de versiones propietarios, como BitKeeper. Estos sistemas a menudo ofrecían características avanzadas, pero podían ser costosos y limitados en su disponibilidad.

El Nacimiento de Git: La Necesidad Agudiza el Ingenio

En 2002, el desarrollo del kernel de Linux utilizaba BitKeeper, un sistema de control de versiones propietario. Sin embargo, en 2005, la licencia gratuita de BitKeeper fue revocada, dejando a la comunidad de Linux en una situación precaria.

Figura 8
Linus Torvalds, creador de GIT y LINUX S.O

Nota: *Creador de varias contribuciones de código abierto.* ***Fuente:*** *Autor*

Linus Torvalds, el creador de Linux, tomó la iniciativa y comenzó a desarrollar Git.

Los objetivos de Torvalds eran claros:

- Velocidad: Git debía ser extremadamente rápido, incluso para proyectos grandes.
- Simplicidad: Git debía ser fácil de usar y comprender.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- Soporte para flujos de trabajo distribuidos: Git debía permitir a los desarrolladores trabajar de forma independiente y sincronizar sus cambios fácilmente.

Hitos Importantes en la creación de Git

- Abril de 2005: Linus Torvalds comienza el desarrollo de Git.
- Junio de 2005: Se lanza la primera versión de Git.
- 2008: Se lanza GitHub, una plataforma de alojamiento de código que utiliza Git.
- 2010: Git se convierte en el sistema de control de versiones más popular del mundo.

8.2.6 ***La Filosofía de Git: Un Enfoque Distribuido***

Git revolucionó el control de versiones al adoptar un enfoque distribuido. En lugar de un servidor central, cada desarrollador tiene una copia completa del historial de versiones en su máquina local. Esto ofrece varias ventajas: (Chacon & Straub, 2024)

Rendimiento excepcional: Las operaciones locales son extremadamente rápidas, ya que no requieren comunicación con un servidor central.

Flujos de trabajo flexibles: Git permite flujos de trabajo de ramificación y fusión muy flexibles, lo que facilita la colaboración y la experimentación.

Resiliencia: Si un desarrollador pierde su copia local del historial de versiones, puede recuperarla de otro desarrollador.

Los Componentes Clave de Git:

- Repositorio: Un repositorio de Git es un directorio que contiene el historial de versiones de un proyecto.
- Confirmación (Commit): Una confirmación es una instantánea de los cambios realizados en un repositorio.
- Ramificación (Branch): Una ramificación es una línea de desarrollo independiente.
- Fusión (Merge): La fusión combina los cambios de dos ramificaciones.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

El lanzamiento de GitHub en 2008 impulsó aún más la adopción de Git. GitHub proporcionó una plataforma centralizada para alojar repositorios de Git, lo que facilitó la colaboración entre desarrolladores. Hoy en día, Git es el sistema de control de versiones más utilizado del mundo. Se utiliza en proyectos de todos los tamaños, desde pequeños proyectos personales hasta grandes proyectos empresariales. (Chacon & Straub, 2024)

El Impacto de Git: Transformando el Desarrollo de Software

Git ha tenido un impacto profundo en el desarrollo de software. Ha facilitado la colaboración, la experimentación y la innovación. Git ha permitido a los desarrolladores trabajar de forma más eficiente y ha mejorado la calidad del software.

Git, el Legado de Linus Torvalds

Git fue creado por Linus Torvalds en 2005, en respuesta a la revocación de la licencia gratuita de BitKeeper, que se utilizaba para el desarrollo del kernel de Linux. La historia de Git es un testimonio del poder de la innovación y la colaboración. Linus Torvalds creó una herramienta que transformó el desarrollo de software y que seguirá siendo relevante durante muchos años. Torvalds necesitaba un sistema de control de versiones que fuera rápido, distribuido y capaz de manejar proyectos grandes y complejos. (Chacon & Straub, 2024)

Características Clave de Git:

- Distribuido: Cada desarrollador tiene una copia completa del repositorio, lo que permite trabajar sin conexión y facilita la creación de ramas.
- Rendimiento: Git está diseñado para ser rápido y eficiente, incluso con proyectos grandes.
- Ramificación flexible: Git facilita la creación y fusión de ramas, lo que permite el desarrollo paralelo y la experimentación.
- Integridad: Git utiliza sumas de comprobación para garantizar la integridad de los datos.

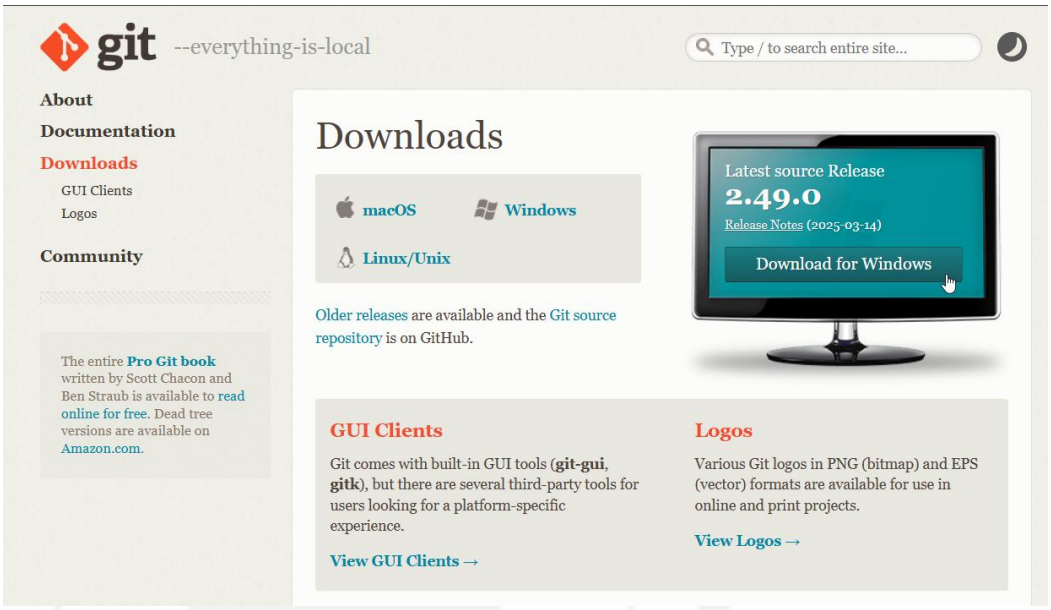
En resumen, antes de Git, el control de versiones era a menudo un proceso centralizado, lento y complicado. Git revolucionó el desarrollo de software al proporcionar un sistema distribuido, rápido y flexible que facilitó la colaboración y el seguimiento de cambios.

A continuación, revisaremos como se maneja esta herramienta y parte del proceso será referenciado en el sistema operativo de Windows para su utilización y posterior administración de proyectos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 9

Instalación de GIT en Windows parte 1

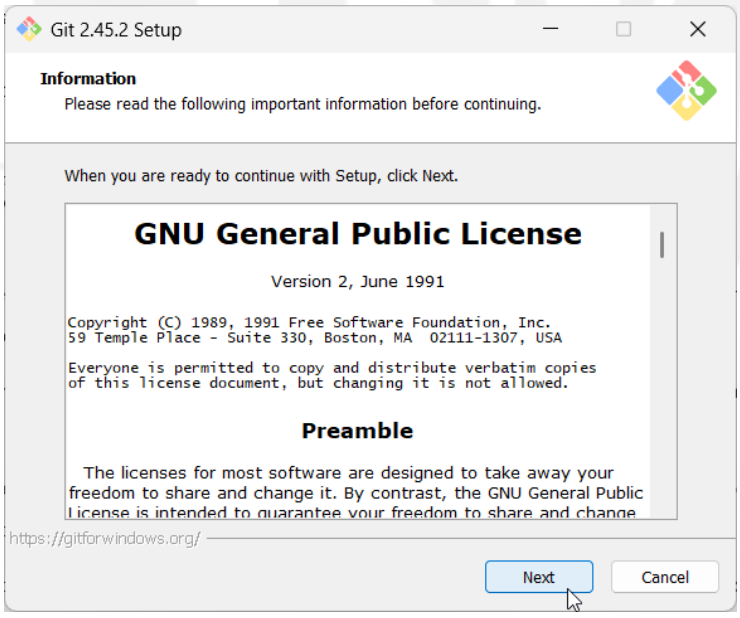


Nota: *Se representa la instalación dentro de Windows como entorno de cambios.* ***Fuente:*** *Autor*

Para poder realizar la instalación debemos descargar el ejecutable de Windows que nos permita instalar esta herramienta, así como podemos visualizar en la Figura 9.

Figura 10

Instalación Git en Windows parte 2



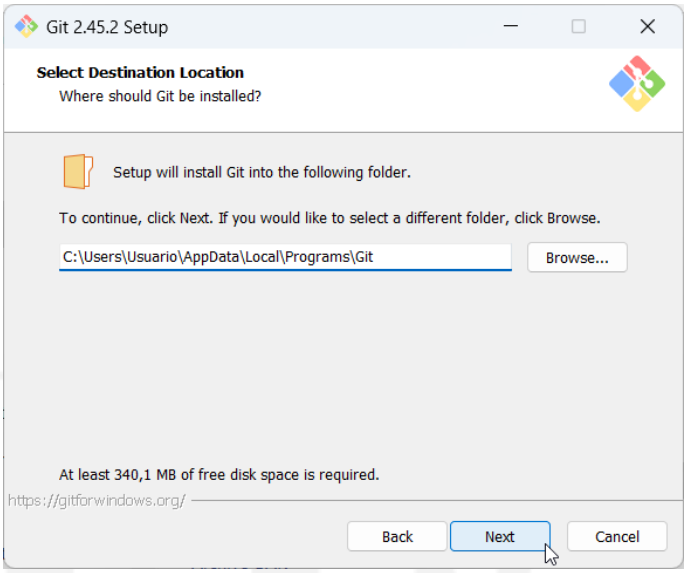
Nota*: Se considera el proceso de instalación recomendado por GIT y su documentación.* ***Fuente:*** *(git-scm, s.f.)*

Para poder realizar la instalación debemos dar doble clic en ejecutable de GIT Windows que nos e instalar como se indica en la en la Figura 10.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Una vez demos clic en siguiente tendremos que seleccionar la ruta donde se va instalar por defecto GIT, como se indica en la Figura 11.

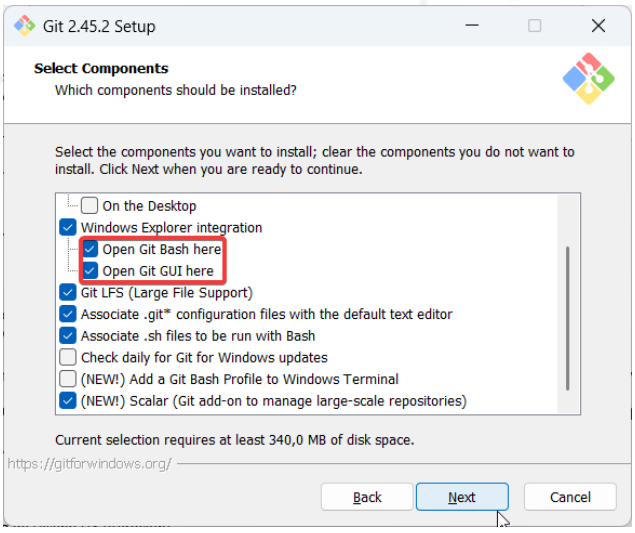
Figura 11
Instalación de Windows parte 3



Nota: *Selección de ruta a instalar.* ***Fuente:*** *(git-scm, s.f.)*

En la Figura 12 podemos divisar las características generales de GIT y tendremos que seleccionar las herramientas que por defecto vienen para instalarlos durante este proceso.

Figura 12
Modelo de Trabajo sobre la Pirámide de UX



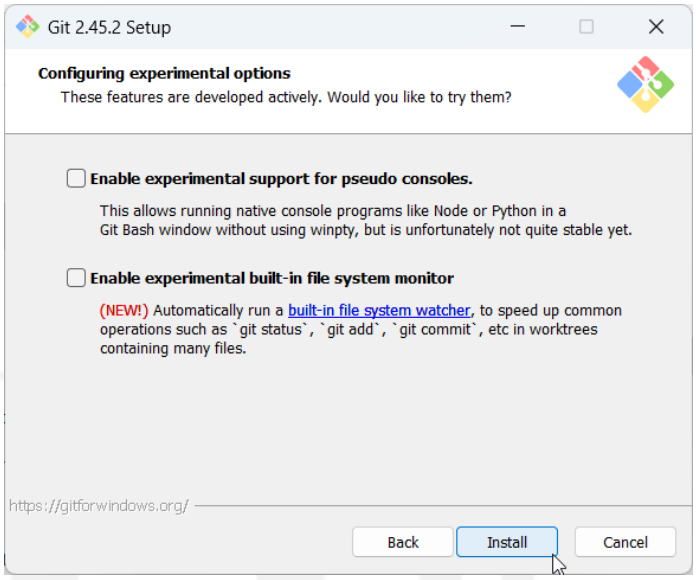
Nota: *Selección de elementos a instalar.* ***Fuente:*** *(git-scm, s.f.)*

En la Figura 13 se puede evidenciar ya el proceso de instalación previo a iniciarse cabe recalcar que hemos dejado todo por defecto para que pueda ser eficiente la instalación por ello

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

deben tener en cuenta que casi todas las configuraciones se realizan por defecto y hemos dado clic en siguiente hasta llegar a la figura indicada.

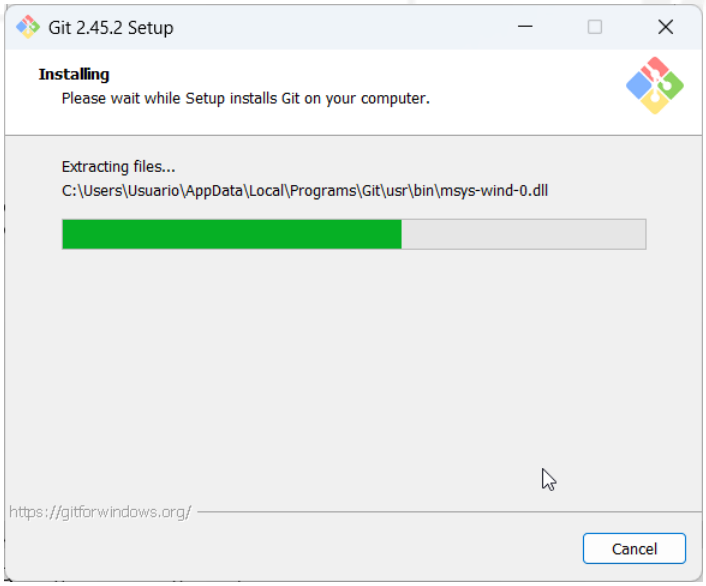
Figura 13
Instalación de Git parte 4



Nota: *Se representa proceso de instalación de GIT.* ***Fuente:*** *(git-scm, s.f.)*

En la Figura 14 se puede evidenciar ya el proceso de instalación va avanzando este paso no dura más de unos 5 minutos, una vez instalado tendremos que verificar por comandos la versión.

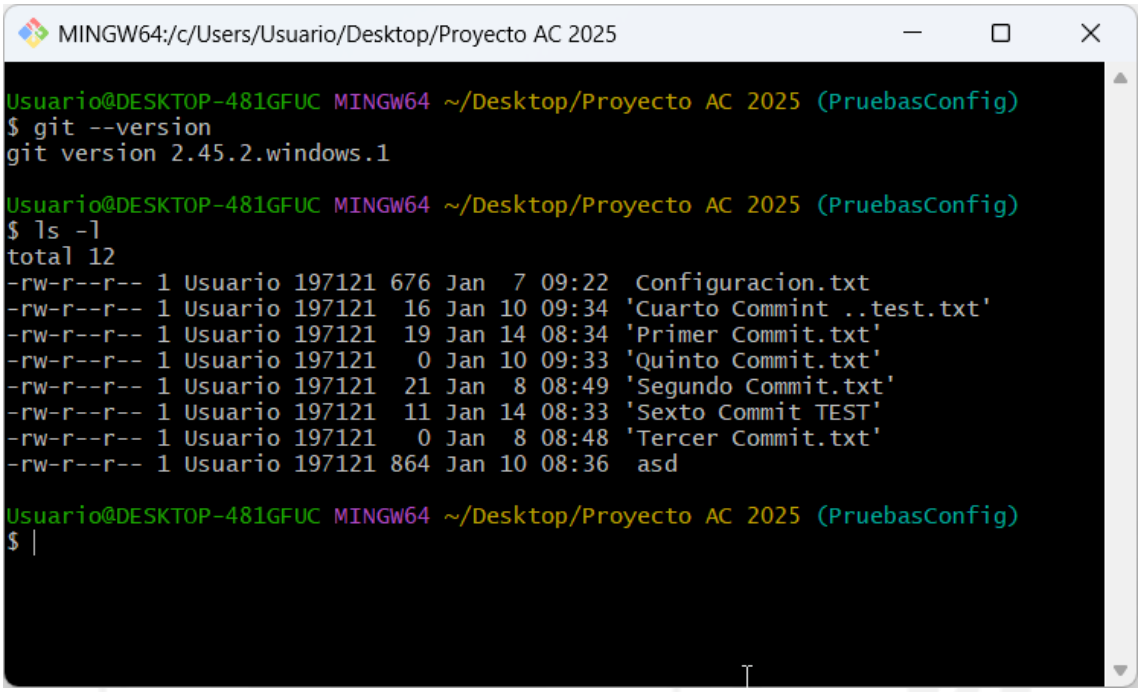
Figura 14
Instalación de GIT en Windows



Nota: *Se representa proceso de instalación de GIT.* ***Fuente:*** *(git-scm, s.f.)*

En la Figura 15 se puede evidenciar la versión instalada de GIT, así como las herramientas que esta facilita para la creación de comandos que permitan la administración de proyectos en la web. Esta herramienta se instaló por defecto y una vez demos clic derecho sobre el documento o la carpeta principal vamos a encontrar el nombre: "Open Git Bash Here" que nos permite obtener un terminal en Windows que se torna en un intérprete de comandos de GIT y Linux, pero en Windows. (Chacon & Straub, 2024)

Figura 15
Verificación de Git Instalado en Windows.



Nota: *Se verifica la instalación de GIT.* ***Fuente:*** *(git-scm, s.f.)*

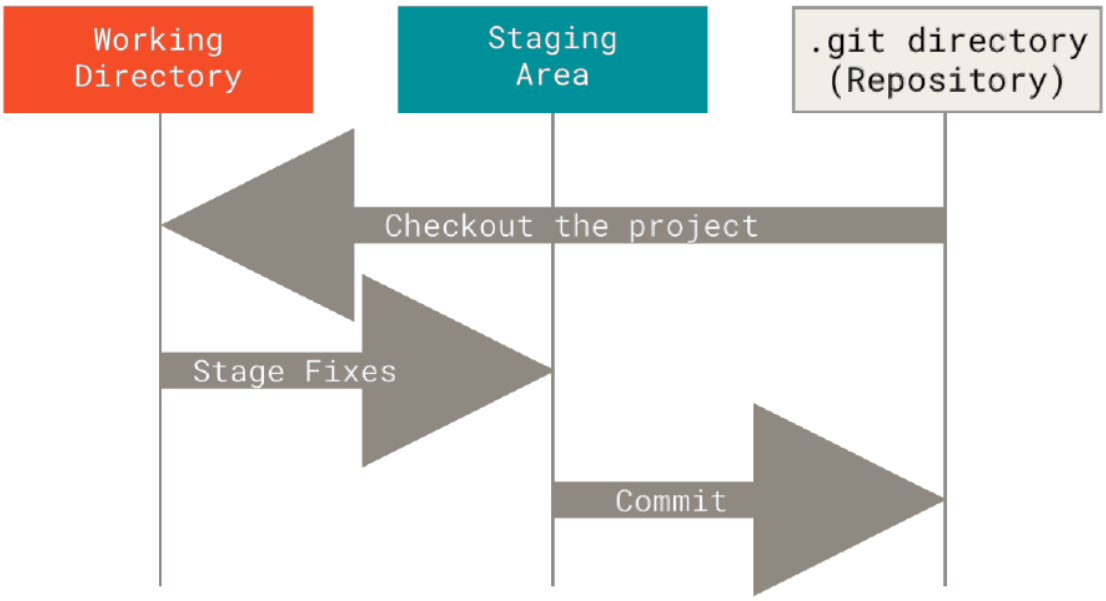
8.2.7 El Árbol de Estados de GIT

En este apartado debemos prestar más atención ya que esto es lo principal que se debe recordar sobre Git, si quieres que el resto de tu proceso de aprendizaje vaya por buen camino. Git tiene tres estados principales en los que pueden residir tus archivos: modificados, por etapas y confirmados.

- Modificado: significa que has cambiado el archivo, pero aún no lo has confirmado en tu base de datos.
- Staged: significa que has marcado un archivo modificado en su versión actual para que entre en tu próxima instantánea de confirmación.
- Confirmado significa que los datos están almacenados de forma segura en tu base de datos local.

Esto nos lleva a las tres secciones principales de un proyecto Git: denominado como el árbol de trabajo, el área de staging (área de espera), y el directorio principal de Git.

Figura 16
Árbol de trabajo de GIT.



Nota: *Se visualizar el área de trabajo de GIT.* ***Fuente:*** *(git-scm, s.f.)*

En la Figura 16 podemos visualizar el Árbol de trabajo, área de preparación y directorio Git donde el árbol de trabajo es la una única comprobación de una versión del proyecto. Estos archivos se extraen de la base de datos comprimida en el directorio Git y se colocan en el disco para que puedas utilizarlos o modificarlos. El área de preparación (staging) es un archivo, generalmente contenido en tu directorio Git, que almacena información sobre lo que se incluirá en tu próxima confirmación. Su nombre técnico en la jerga de Git es “índice”, pero la frase “área de preparación” es también valida. El directorio Git es donde esta tecnología almacenara los metadatos y la base de datos de objetos de tu proyecto. Esta es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador. (Chacon & Straub, 2024)

8.2.8 ***El flujo de trabajo básico de Git***

Tiene una relación que se da de la siguiente forma así:

Modificas, editas o cambias de alguna forma los archivos en tu árbol de trabajo.

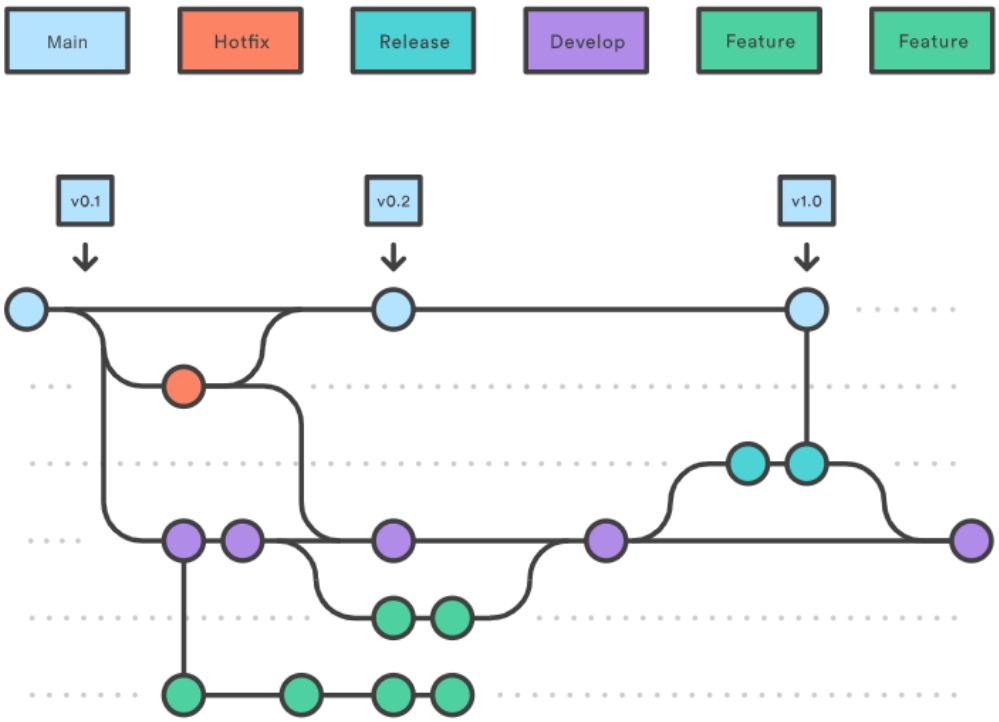
Seleccionas los cambios que quieres que formen parte de tu próxima confirmación, que añade sólo esos cambios al área de preparación sólo esos cambios al área de preparación.

Realizas una confirmación, que toma los archivos tal y como están en el área de preparación y almacena esa instantánea de forma permanentemente en tu directorio Git.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Si una versión concreta de un archivo está en el directorio Git, se considera confirmada. Si se ha modificado y se ha añadido al área de preparación, se considera preparado. Y si ha sido modificado desde que fue editada pero no ha sido preparado, se considera modificado. En Conceptos básicos de Git, aprenderás más sobre estos estados y cómo puedes aprovecharte de ellos o saltarte la parte de preparación por completo.

Figura 17
Flujo de trabajo de GIT



Nota: *Flujo de trabajo de GIT.* ***Fuente:*** *(atlassian, s.f.)*

8.2.9 ***Autoevaluación 2***

- ¿Cuál es el origen del nombre "Git" según lo explicado en el documento?

Es un acrónimo de “Global Information Tracker”

Fue elegido por su simplicidad y su connotación coloquial

Hace referencia a una tecnología específica en redes

Es una abreviación de “Great Innovation Tool”

- ¿Cuál fue uno de los principales problemas de los sistemas de control de versiones centralizados antes de Git?

La imposibilidad de trabajar de forma colaborativa

La falta de herramientas gráficas

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

El punto único de fallo en el servidor central

La incapacidad de gestionar proyectos pequeños

- ¿Qué evento impulsó a Linus Torvalds a desarrollar Git?

La creación de nuevos lenguajes de programación

La revocación de la licencia gratuita de BitKeeper

La demanda de una interfaz gráfica avanzada

La necesidad de un sistema centralizado más robusto

- ¿Cuál de las siguientes características NO corresponde a Git?

Enfoque distribuido

Operaciones que dependen siempre de un servidor central

Velocidad en el manejo de versiones

Soporte para ramificación flexible

- ¿Qué representa el "repositorio" en Git?

Un archivo temporal para guardar cambios

Una interfaz gráfica para editar código

Un directorio que contiene el historial de versiones de un proyecto

Un servidor remoto para la colaboración

- ¿Cuáles son los tres estados en los que pueden residir los archivos en Git?

Nuevo, modificado y respaldado

Activo, pendiente y archivado

Modificado, staged y confirmado

Editado, revisado y publicado

- ¿Qué ventaja principal ofrece el enfoque distribuido de Git?

Permite ejecutar comandos únicamente en la nube

Hace innecesaria la utilización de herramientas de fusión

Ofrece operaciones locales rápidas sin depender de un servidor central

Elimina la necesidad de crear ramas para el desarrollo

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- ¿Cuál es la función de la "área de staging" en Git?

Almacenar permanentemente los cambios en el repositorio

Guardar una copia de seguridad del proyecto en la nube

Mantener un registro temporal de los cambios que se incluirán en la próxima confirmación

Actuar como la interfaz gráfica del sistema

- ¿Qué es Git Bash y cuál es su función en Windows?

Es una herramienta para la edición de imágenes

Es una interfaz de línea de comandos que permite ejecutar comandos de Git

Es un entorno de desarrollo integrado exclusivo para Linux

Es una aplicación gráfica para gestionar repositorios

- ¿Cuál de los siguientes NO es uno de los componentes clave de Git mencionados en el documento?

Repositorio

Confirmación (Commit)

Fusión (Merge)

Servidor Central

8.2.10 ***Resumen de la Unidad 2***

En la segunda unidad hemos abordado como se da el origen y la historia de Git, destacando la importancia de contar con un sistema de control de versiones que permita manejar de forma ordenada y coherente los cambios en el desarrollo de software. Antes de Git, los sistemas de control de versiones estaban basados en un enfoque centralizado, representado por herramientas como CVS y Subversion (SVN). Estos sistemas utilizaban un servidor central para almacenar todo el historial de versiones, lo que implicaba importantes inconvenientes, tales como:

Punto único de fallo: Si el servidor central fallaba, se perdía el acceso al historial completo del proyecto.

Rendimiento lento: La dependencia de la comunicación constante con un servidor central hacía que las operaciones fueran lentas, especialmente en proyectos de gran escala.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Flujos de trabajo limitados: La gestión de ramas y fusiones era compleja y propensa a errores, lo cual dificultaba el trabajo colaborativo y la experimentación en el desarrollo de software.

Además, antes de la consolidación de sistemas centralizados, los desarrolladores solían intercambiar cambios mediante archivos de parches y mantener copias de seguridad locales, métodos que, aunque útiles en su momento, presentaban limitaciones para proyectos colaborativos y de gran envergadura. Algunos entornos incluso utilizaban sistemas propietarios, como BitKeeper, que aunque ofrecían características avanzadas, presentaban restricciones en cuanto a disponibilidad y costo.

El Nacimiento de Git y la Innovación de Linus Torvalds

La necesidad de contar con una herramienta más robusta y flexible se hizo patente cuando, en 2002, el kernel de Linux utilizaba BitKeeper. Sin embargo, en 2005 la licencia gratuita de BitKeeper fue revocada, lo que dejó a la comunidad del desarrollo del kernel en una situación crítica. Fue en ese contexto que Linus Torvalds, el creador de Linux, decidió desarrollar Git.

Torvalds planteó objetivos muy claros para la nueva herramienta:

Velocidad: Git debía ser extremadamente rápido, incluso para proyectos de gran tamaño.

Simplicidad: Se buscaba que fuera fácil de usar y de comprender por los desarrolladores.

Soporte para flujos de trabajo distribuidos: Git debía permitir a los desarrolladores trabajar de forma independiente, manteniendo una copia completa del historial en cada máquina local y facilitando la sincronización de cambios.

Entre los hitos importantes se destacan:

Abril de 2005: Inicio del desarrollo de Git.

Junio de 2005: Lanzamiento de la primera versión.

2008: Se lanza GitHub, plataforma que impulsó aún más la adopción de Git.

2010: Git se consolida como el sistema de control de versiones más popular a nivel mundial.

El nombre “Git” fue escogido por Torvalds en parte por su simplicidad y también por su connotación coloquial en inglés (donde "git" se asocia a “idiota” o “inútil”). Con este nombre, Torvalds mostró su sentido del humor y restó importancia a la herramienta, enfatizando en cambio sus robustas capacidades y su carácter revolucionario en el desarrollo de software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Filosofía y Características Clave de Git

Git introdujo un enfoque distribuido que transformó radicalmente la forma en que se manejaba el control de versiones. A diferencia de los sistemas centralizados, Git permite que cada desarrollador posea una copia completa del repositorio, lo que ofrece diversas ventajas:

Operaciones locales rápidas: Al no depender de un servidor central para realizar operaciones, Git permite trabajar de manera ágil y con alto rendimiento.

Flexibilidad en el flujo de trabajo: La posibilidad de crear y fusionar ramas de forma sencilla fomenta la colaboración y la experimentación en el desarrollo.

Resiliencia y seguridad: Si un desarrollador pierde su copia local, puede recuperarla fácilmente de otro repositorio, asegurando la integridad del historial.

Las características fundamentales de Git se resumen en:

Repositorio: Es el directorio que almacena el historial completo del proyecto, incluyendo metadatos y objetos.

Confirmación (Commit): Representa una instantánea de los cambios realizados, que se almacena de forma permanente en el repositorio.

Ramificación (Branch): Permite crear líneas de desarrollo independientes, facilitando la experimentación y el trabajo en paralelo.

Fusión (Merge): Es el proceso mediante el cual se combinan los cambios de distintas ramas en una única línea de desarrollo.

Git utiliza sumas de comprobación para garantizar la integridad de los datos, asegurando que los cambios no se hayan corrompido. Además, su estructura distribuida y la capacidad para trabajar sin conexión lo convierten en una herramienta esencial para proyectos tanto pequeños como de gran escala.

Implementación y Proceso de Instalación en Windows

El documento también detalla el proceso de instalación de Git en el sistema operativo Windows, resaltando la importancia de seguir las configuraciones recomendadas para una instalación eficiente. Se explica paso a paso cómo descargar el ejecutable de Git, proceder con la instalación y seleccionar la ruta por defecto. Durante el proceso, se menciona la importancia de mantener la configuración por defecto para garantizar la compatibilidad y el correcto funcionamiento de la herramienta. Al finalizar la instalación, se verifica la versión de Git utilizando la terminal. En Windows, se puede acceder a Git Bash, una interfaz de línea de

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

comandos que permite ejecutar comandos de Git y emular un entorno similar al de Linux, facilitando la administración y manejo del repositorio. Con Git Bash, al hacer clic derecho en el directorio o archivo principal, se muestra la opción “Open Git Bash Here”, lo que permite iniciar rápidamente una terminal para gestionar el proyecto.

El Árbol de Estados y el Flujo de Trabajo Básico de Git

Una parte crucial del aprendizaje de Git es comprender los tres estados principales en los que pueden residir los archivos:

- Modificado: El archivo ha sido alterado, pero los cambios aún no han sido preparados.
- Staged (Preparado): El archivo modificado ha sido marcado para ser incluido en la siguiente confirmación.
- Confirmado (Committed): Los cambios se han almacenado de forma permanente en el repositorio.

El documento explica que estos tres estados forman parte del “árbol de trabajo” de Git, que se compone de:

- El árbol de trabajo: Donde se encuentran los archivos actuales del proyecto.
- El área de staging (índice): Donde se almacenan temporalmente los cambios que serán confirmados.
- El directorio Git: Que contiene todos los metadatos y el historial completo del proyecto.

El flujo de trabajo básico de Git se desarrolla de la siguiente manera:

- Se modifican los archivos en el árbol de trabajo.
- Se seleccionan y añaden los cambios al área de staging.
- Se realiza una confirmación que toma los archivos del área de staging y los almacena en el repositorio, registrando una instantánea del estado del proyecto.

Este proceso permite un control minucioso de cada cambio, facilitando la identificación y corrección de errores, y promoviendo un manejo eficiente y seguro del código.

Impacto y Legado de Git en el Desarrollo de Software

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

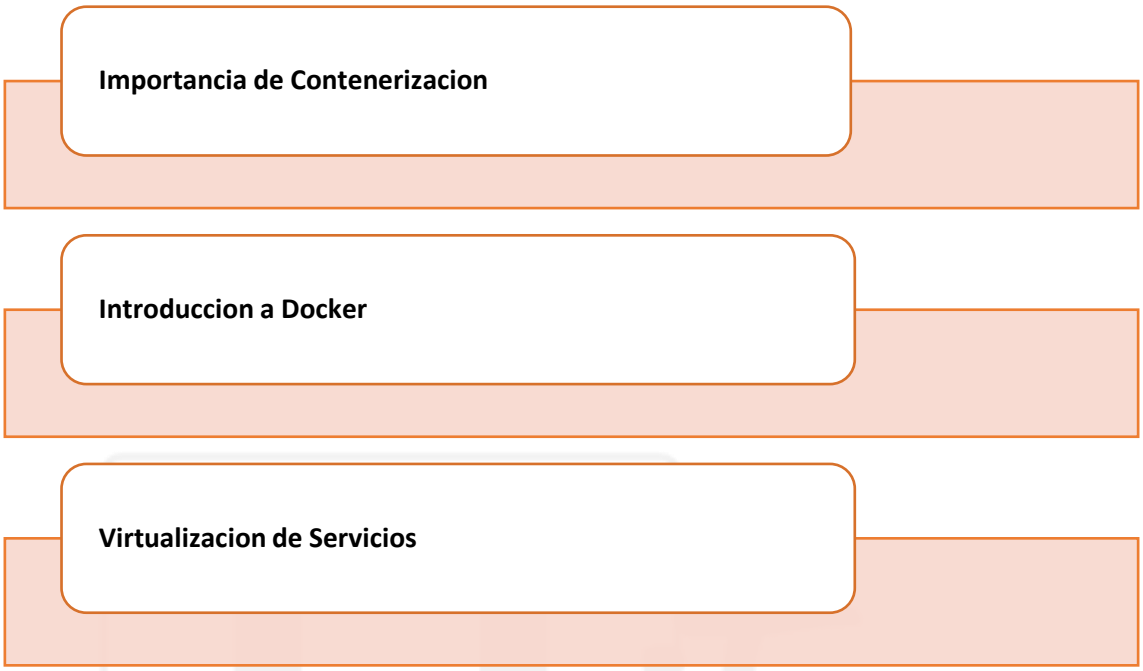
Git ha transformado radicalmente el desarrollo de software al fomentar la colaboración, la experimentación y la innovación. Gracias a su enfoque distribuido, Git no solo ha permitido un manejo más eficiente de los proyectos, sino que también ha contribuido a mejorar la calidad del software, facilitando el trabajo en equipo y la integración continua. El impacto de Git se refleja en su adopción masiva en proyectos de todo tipo, desde desarrollos personales hasta grandes iniciativas empresariales. La aparición de plataformas como GitHub, que se basan en Git, ha potenciado aún más la colaboración entre desarrolladores a nivel global, creando comunidades y ecosistemas en torno a la herramienta. Linus Torvalds, al crear Git, dejó un legado que no solo revolucionó el control de versiones, sino que también sentó las bases para nuevas metodologías de trabajo colaborativo y ágil.

Git se ha convertido en un elemento esencial en el ciclo de desarrollo de software moderno, siendo reconocido por su velocidad, flexibilidad e integridad de datos. En general en la unidad 2 se ofrece una visión completa sobre la evolución del control de versiones, destacando las limitaciones de los sistemas centralizados y presentando Git como la solución innovadora y robusta que transformó el panorama del desarrollo de software. Se explican tanto los orígenes históricos y la filosofía detrás de Git como sus componentes clave, su flujo de trabajo básico y el proceso de instalación en Windows, lo que lo hace accesible para desarrolladores de cualquier nivel. Para docentes y profesionales este contenido representa una base sólida para enseñar y aplicar buenas prácticas en el manejo de versiones. Comprender cómo Git gestiona los cambios, la importancia de su enfoque distribuido y el valor de herramientas como Git Bash en Windows permite estructurar procesos académicos y talleres prácticos que preparen a la nueva generación de desarrolladores para enfrentar retos en proyectos reales. La capacidad de Git para facilitar el trabajo colaborativo, garantizar la integridad del código y permitir la experimentación sin comprometer el estado estable del proyecto lo convierte en una herramienta indispensable en la industria del software. Su legado y la visión de Linus Torvalds continúan siendo relevantes y se reflejan en la evolución constante de metodologías de desarrollo, integración y despliegue continuo.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

8.3 UNIDAD 3 (HTML, CSS y JS)

*8.3.1 **Temas y Subtemas***



*8.3.2 **Importancia de Contenerización***

La contenerización es una tecnología que permite empaquetar una o varias aplicaciones y sus dependencias en una unidad portátil llamada contenedor. Un contenedor incluye todo lo necesario para ejecutar la aplicación: código, bibliotecas, configuraciones y dependencias del sistema, permitiendo tener un ambiente prácticamente listo en segundos si sabemos manejar las correctas configuraciones del caso, este trabaja con imágenes o archivos que permitan exponer y palpar estas configuraciones (Iradier & Martínez, 2021).

*8.3.3 **¿Cómo Funciona la Contenerización?***

Aislamiento: Los contenedores proporcionan un aislamiento de procesos, lo que significa que cada contenedor se ejecuta en su propio espacio aislado. Esto evita conflictos entre aplicaciones y garantiza que cada aplicación tenga su propio entorno.

Portabilidad: Los contenedores son portátiles y pueden ejecutarse en cualquier sistema operativo o infraestructura que admita la tecnología de contenedores. Esto facilita el despliegue de aplicaciones en diferentes entornos.

Ligereza: Los contenedores son ligeros y eficientes, ya que comparten el kernel del sistema operativo host. Esto permite ejecutar más contenedores en un solo servidor en comparación con las máquinas virtuales.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

8.3.4 *Gestión de Ambientes de Desarrollo con Contenerización*

El proceso de contenerización simplifica la gestión de ambientes de desarrollo de varias maneras:

- Consistencia: Los contenedores garantizan que todos los desarrolladores trabajen en el mismo entorno, lo que reduce los errores y conflictos causados por diferencias en las configuraciones locales.
- Aislamiento de Dependencias: Cada proyecto puede tener sus propias dependencias aisladas en contenedores separados, lo que evita conflictos entre versiones de bibliotecas.
- Despliegue Simplificado: Los contenedores facilitan el despliegue de aplicaciones en diferentes entornos, ya que el mismo contenedor se puede ejecutar en desarrollo, pruebas y producción.
- Escalabilidad: Los contenedores permiten escalar aplicaciones fácilmente mediante la creación de múltiples instancias de un contenedor.
- Reproducibilidad: Los contenedores garantizan que los ambientes de desarrollo sean reproducibles, lo que facilita la incorporación de nuevos desarrolladores al equipo.

Herramientas de Contenerización

Docker: Es la plataforma de contenedores más popular y ampliamente utilizada. Docker facilita la creación, el despliegue y la gestión de contenedores.

Kubernetes: Es un sistema de orquestación de contenedores que automatiza el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores.

La contenerización es una tecnología que ha permitido una gran mejora en el desarrollo de software moderno ya que simplifica la gestión de ambientes de desarrollo, mejora la consistencia y facilita el despliegue de aplicaciones. Docker y Kubernetes son herramientas clave en el ecosistema de la contenerización que permiten desplegar un ambiente computacional completo para el desarrollo de aplicaciones. (Iradier & Martínez, 2021)

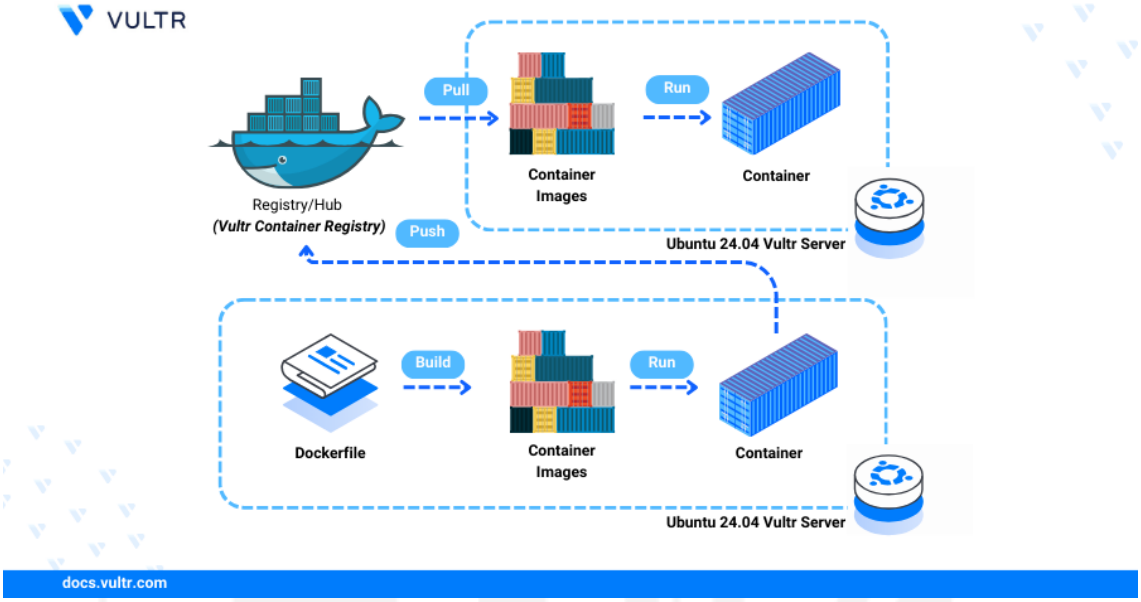
8.3.5 *Introducción a Docker: una plataforma integradora*

Docker es una plataforma de código abierto que permite a los desarrolladores automatizar el despliegue de aplicaciones dentro de "contenedores". Un contenedor es un entorno de software estandarizado que empaqueta todo lo que una aplicación necesita para

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ejecutarse: código, bibliotecas, configuraciones y dependencias. Esto garantiza que la aplicación se ejecute de manera consistente en cualquier entorno, ya sea en la máquina de un desarrollador, en un servidor de pruebas o en un entorno de producción. (Iradier & Martínez, 2021)

Figura 18
Proceso de Docker



***Nota**: Flujo de trabajo de Docke. **Fuente**: (vultr, 2024)*

8.3.6 ***Historia de Docker***

Orígenes en dotCloud:

La historia de Docker se remonta a 2008, cuando Solomon Hykes fundó dotCloud, una empresa que ofrecía una plataforma de servicios en la nube (PaaS). Dentro de dotCloud, se desarrolló una tecnología interna para facilitar el despliegue de aplicaciones. Esta tecnología se convertiría en la base de lo que hoy conocemos como Docker. (IONOS, 2023)

Nacimiento de Docker:

En 2013, Solomon Hykes presentó Docker al público en la PyCon de Santa Clara. Docker se lanzó como un proyecto de código abierto en marzo de 2013. Rápidamente, Docker ganó popularidad entre los desarrolladores debido a su facilidad de uso y su capacidad para simplificar el despliegue de aplicaciones. DotCloud, Inc. se transformó en Docker, Inc, que hoy es la firma que promueve el proyecto y ofrece a la venta la versión comercial de la plataforma. (DataScientest.com, 2008)

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Impacto y Adopción:

Docker democratizó la tecnología de contenedores, haciéndola accesible para desarrolladores de todos los niveles, gracias a su fácil acceso ya que su plataforma es independiente al S.O que se esté utilizando y su capacidad para simplificar el despliegue y la gestión de aplicaciones ha sido fundamental para la adopción de la computación en la nube y la arquitectura de microservicios. (IONOS, 2023)

Significado del Nombre "Docker"

Inspiración marítima: El nombre "Docker" proviene del término inglés "docker", que se refiere a los trabajadores portuarios que cargan y descargan contenedores de carga en los barcos. Esta analogía es muy apropiada, ya que Docker permite empaquetar aplicaciones en "contenedores" que pueden ser transportados y desplegados fácilmente en diferentes entornos. En esencia, Docker facilita el "empaquetado" y "envío" de aplicaciones, al igual que los trabajadores portuarios hacen con los contenedores de carga. (Docker, 2021)

Conceptos Clave de Docker

Imágenes: Una imagen de Docker es una plantilla de solo lectura con instrucciones para crear un contenedor. Las imágenes contienen todo lo necesario para ejecutar una aplicación. (IBM, 2023)

Contenedores: Un contenedor es una instancia en ejecución de una imagen de Docker. Los contenedores son entornos aislados que permiten ejecutar aplicaciones de manera consistente. (IBM, 2023)

Docker Hub: Es un registro en línea donde se pueden almacenar y compartir imágenes de Docker. Es un recurso de gran baluarte ya que puedes encontrar imágenes preconstruidas para diversas aplicaciones. (IBM, 2023)

Beneficios de Docker

Consistencia: Las aplicaciones se ejecutan de manera consistente en cualquier entorno. (Amazon Web Services, 2021)

Portabilidad: Las aplicaciones se pueden desplegar fácilmente en diferentes plataformas. (Amazon Web Services, 2021)

Eficiencia: Los contenedores utilizan menos recursos que las máquinas virtuales. (Amazon Web Services, 2021)

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Escalabilidad: Las aplicaciones se pueden escalar fácilmente para adaptarse a la demanda. (Amazon Web Services, 2021)

Agilidad: Docker facilita la integración y el despliegue continuos (CI/CD).

Docker ha transformado la forma en que desarrollamos y desplegamos aplicaciones, y su impacto seguirá siendo significativo en el futuro de la industria del desarrollo.

Estructura de Funcionamiento de Docker

Docker se basa en una arquitectura cliente-servidor

Cliente Docker:

Es la interfaz de línea de comandos (CLI) que utilizas para interactuar con el demonio Docker. Envía comandos al demonio para construir, ejecutar y gestionar contenedores. (DataScientest.com, 2008)

Demonio Docker (dockerd):

Es el componente principal de Docker nos permite ejecutar en segundo plano y gestionar las imágenes y los contenedores. Se encargarán de construir imágenes, ejecutar contenedores y gestionar el almacenamiento y las redes. (DataScientest.com, 2008)

Imágenes Docker:

Son plantillas de solo lectura que contienen las instrucciones para crear un contenedor. Definen el sistema operativo, las bibliotecas, las aplicaciones y las configuraciones necesarias.

Contenedores Docker:

Son instancias en ejecución de imágenes que proporcionan un entorno aislado para ejecutar aplicaciones las cuales comparten el núcleo del sistema operativo del host, lo que los hace más ligeros y eficientes que las máquinas virtuales, dado que no cuentan con esta parte física y pesada.

Registros Docker:

Son repositorios para almacenar y compartir imágenes Docker. Docker Hub es el registro público predeterminado. También puedes crear registros privados para almacenar imágenes internas.

Virtualización de Servicios con Docker

Docker facilita la virtualización de servicios al permitir empaquetar cada servicio en un contenedor independiente. Esto ofrece varias ventajas:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Aislamiento: Cada servicio se ejecuta en su propio entorno aislado, lo que evita conflictos de dependencias. Esto facilita la gestión de aplicaciones complejas con múltiples servicios. (Docker, 2021)

Portabilidad: Los contenedores se pueden ejecutar en cualquier entorno que tenga Docker instalado. Esto simplifica el despliegue de aplicaciones en diferentes plataformas. (Docker, 2021)

Escalabilidad: Los contenedores se pueden escalar fácilmente para adaptarse a la demanda. Esto permite gestionar picos de tráfico y garantizar la disponibilidad de los servicios. (Docker, 2021)

Eficiencia: Los contenedores utilizan menos recursos que las máquinas virtuales. Esto permite ejecutar más servicios en un mismo servidor. (Iradier & Martínez, 2021)

Despliegue rápido: Los contenedores permite un despliegue muy rápido, esto facilita la integración y el despliegue continuos (CI/CD). (Docker, 2021)

Ejemplo de Virtualización de Servicios con Docker

Imagina que tienes una aplicación web con los siguientes servicios:

- Servidor web (Nginx o Apache)
- Base de datos (MySQL o PostgreSQL)
- Aplicación PHP

Con Docker, puedes empaquetar cada uno de estos servicios en un contenedor independiente. Luego, puedes utilizar Docker Compose para orquestar la ejecución de estos contenedores y definir las dependencias entre ellos. Esto te permite:

- Desarrollar y probar cada servicio de forma independiente.
- Desplegar la aplicación en cualquier entorno con Docker.
- Escalar cada servicio de forma individual según la demanda.

La orquestación de contenedores para aplicaciones complejas con muchos contenedores se deben utilizar herramientas de orquestación como Docker o Kubernetes. Es importante implementar medidas de seguridad para proteger los contenedores y los datos. Donde Docker proporcionara herramientas para gestionar las redes y el almacenamiento de los contenedores.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

8.3.7 ***Autoevaluación 3***

- ¿Qué es la contenerización en el desarrollo de software?

Un proceso para compilar código más rápido

Un método para empaquetar aplicaciones y sus dependencias en contenedores

Un sistema de bases de datos distribuido

Una técnica de desarrollo en dispositivos móviles

- ¿Cuál de las siguientes es una ventaja clave de los contenedores?

Requieren un sistema operativo separado para cada aplicación

Son más pesados que las máquinas virtuales

Permiten ejecutar aplicaciones en entornos aislados

No requieren gestión de dependencias

- ¿Cuál es una característica fundamental de Docker?

Proporciona hardware adicional para ejecutar contenedores

Facilita la creación, despliegue y gestión de contenedores

Reemplaza la necesidad de sistemas operativos

Solo funciona en servidores físicos

- ¿Qué beneficio clave ofrece Docker en comparación con las máquinas virtuales?

Consume más recursos

Aísla completamente el sistema operativo

Permite compartir el kernel del sistema operativo

No necesita configuración

- ¿Cuál es el propósito principal de Kubernetes?

Compilar código más rápido

Administrar y orquestar múltiples contenedores

Servir como base de datos distribuida

Optimizar imágenes de Docker

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- ¿Cómo se asegura la portabilidad de los contenedores?

Gracias a la independencia del hardware y el sistema operativo

Porque incluyen un sistema operativo completo

Mediante la configuración manual de cada servidor

Con la instalación de múltiples versiones de software

- ¿Qué componente de Docker se encarga de ejecutar y gestionar contenedores?

Docker Hub

Docker Compose

Dockerd (Demonio de Docker)

Dockerfile

- ¿Cuál es el propósito del Docker Hub?

Administrar redes internas de Docker

Servir como registro público para almacenar y compartir imágenes de Docker

Gestionar la escalabilidad de contenedores

Crear imágenes personalizadas

- ¿Qué permite Docker Compose?

Crear y administrar múltiples contenedores con una sola configuración

Ejecutar contenedores sin necesidad de imágenes

Crear redes físicas para contenedores

Reemplazar la funcionalidad de Docker Hub

- ¿Cuál es la diferencia principal entre imágenes y contenedores en Docker?

Las imágenes son instancias en ejecución, mientras que los contenedores son solo plantillas

Las imágenes contienen la configuración y los contenedores son su ejecución

No hay diferencia; ambos son lo mismo

Los contenedores almacenan datos de forma permanente

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

8.3.8 ***Resumen de la Unidad 3***

En esta unidad se trataron los siguientes puntos a profundidad y aquí se va a proporcionar un pequeño detalle de estos: Funcionamiento de la Contenerización: La contenerización se basa en el aislamiento de procesos, lo que significa que cada contenedor opera de forma independiente sin interferir con otros. Esto evita conflictos entre aplicaciones y asegura que cada una tenga su propio entorno de ejecución. Sus principales ventajas incluyen:

Portabilidad: Los contenedores pueden ejecutarse en cualquier sistema que soporte esta tecnología, como servidores locales o en la nube.

Ligereza: A diferencia de las máquinas virtuales, los contenedores comparten el kernel del sistema operativo host, lo que reduce el consumo de recursos y permite ejecutar múltiples instancias en un solo servidor.

Consistencia y reproducibilidad: Garantiza que los desarrolladores trabajen en el mismo entorno sin problemas de compatibilidad.

Despliegue simplificado y escalabilidad: Facilita el lanzamiento de aplicaciones en diferentes etapas del desarrollo, incluyendo pruebas y producción, además de permitir la ejecución de múltiples instancias según la demanda.

Herramientas Clave en la Contenerización

Docker: Plataforma líder en la creación, despliegue y gestión de contenedores. Su facilidad de uso ha contribuido a la adopción masiva de la contenerización.

Kubernetes: Herramienta de orquestación que permite gestionar múltiples contenedores, automatizando el escalado y la administración de aplicaciones en entornos de producción.

Historia y Evolución de Docker

Docker nació en 2008 dentro de la empresa dotCloud, fundada por Solomon Hykes. Inicialmente, dotCloud desarrolló una tecnología interna para simplificar el despliegue de aplicaciones en la nube. En 2013, Hykes presentó Docker como un proyecto de código abierto en la PyCon de Santa Clara, lo que permitió su rápida adopción por la comunidad de desarrolladores. Gracias a su facilidad de uso y portabilidad, Docker democratizó la tecnología de contenedores y transformó la forma en que las aplicaciones son desplegadas en servidores y la nube. Su impacto ha sido fundamental en la adopción de la arquitectura de microservicios y en la computación en la nube.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

El nombre "Docker" proviene del término marítimo "docker", que hace referencia a los trabajadores portuarios que cargan y descargan contenedores en los barcos. Esta metáfora se alinea con la función de Docker de empaquetar y transportar aplicaciones de un entorno a otro sin problemas. La contenerización ha evolucionado la manera en que se desarrollan y despliegan aplicaciones al proporcionar entornos consistentes, portátiles y escalables.

Docker ha sido la pieza clave de esta transformación, facilitando la adopción de arquitecturas de microservicios y optimizando la gestión de entornos de desarrollo. La combinación de Docker y Kubernetes ha llevado la contenerización al siguiente nivel, permitiendo el despliegue y escalado automático de aplicaciones en la nube y en infraestructuras locales. Gracias a estas tecnologías, el desarrollo de software es más ágil, eficiente y seguro, asegurando la portabilidad y reproducibilidad de aplicaciones en cualquier entorno.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

9. ANEXOS

Solucionario 1.-

Tabla 1: Solucionario unidad 1

# pregunta	Respuesta	Retroalimentación
1	b	Un ambiente de desarrollo proporciona herramientas necesarias para la programación, depuración y prueba de código antes de implementarlo en producción. Su objetivo es asegurar que el software funcione correctamente antes de su despliegue.
2	b	Git es un sistema de control de versiones ampliamente utilizado que permite rastrear cambios en el código, colaborar con otros desarrolladores y administrar versiones del software de manera eficiente.
3	a	La computación en la nube ha permitido que los desarrolladores trabajen en entornos flexibles sin depender de una única máquina física, facilitando el acceso a recursos escalables según la demanda del proyecto.
4	c	Aunque muchos ambientes de desarrollo incluyen interfaces gráficas, no es un requisito obligatorio. Algunos desarrolladores prefieren usar herramientas basadas en línea de comandos como Vim o Emacs.
5	b	Un IDE (Entorno de Desarrollo Integrado) ofrece múltiples herramientas en un solo lugar, como depuradores, compiladores y autocompletado de código, lo que mejora la eficiencia del desarrollo.
6	a	Los ambientes remotos permiten que los desarrolladores trabajen desde cualquier ubicación con acceso a internet, usando herramientas en la nube como AWS Cloud9 o GitHub Codespaces.
7	a	Un entorno de ejecución es donde una aplicación se ejecuta y opera, e incluye bases de datos como MySQL,

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

		servidores web como Apache y sistemas operativos como Linux o Windows.
8	d	Docker permite empaquetar aplicaciones con todas sus dependencias en contenedores, asegurando que el software funcione de manera consistente en diferentes entornos.
9	d	DevOps busca integrar desarrollo y operaciones mediante la automatización y la colaboración continua, permitiendo ciclos de desarrollo más ágiles y eficientes.
10	f	Un entorno de pruebas permite evaluar la funcionalidad y estabilidad de una aplicación antes de que sea implementada en producción, reduciendo el riesgo de errores.

Solucionario 2.-

Tabla 2: Solucionario unidad 2

# pregunta	Respuesta	Retroalimentación
1	b	Linus Torvalds eligió el nombre "Git" por ser corto, fácil de recordar y, en tono humorístico, aprovechar la connotación coloquial en inglés que significa "idiota" o "inútil". No se utilizó como acrónimo ni para referirse a una tecnología en particular.
2	c	Los sistemas centralizados, como CVS o SVN, almacenaban el historial de versiones en un servidor central, lo que generaba un punto único de fallo. Si el servidor fallaba, se perdía el acceso al historial y se interrumpían las operaciones.
3	b	En 2005, la licencia gratuita de BitKeeper fue revocada, lo que dejó a la comunidad del kernel de Linux sin un sistema adecuado para el control de versiones, impulsando a Torvalds a crear Git.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

4	b	Git es un sistema distribuido, lo que significa que cada desarrollador posee una copia completa del historial, permitiendo operaciones locales rápidas sin depender siempre de un servidor central. Esta característica contrasta con los sistemas centralizados.
5	d	En Git, el repositorio es el directorio donde se almacena el historial de versiones y la base de datos de objetos del proyecto, permitiendo el seguimiento y recuperación de cambios.
6	c	Git clasifica los archivos en tres estados: modificado (cambios realizados en el árbol de trabajo), staged (preparados para ser confirmados) y confirmado (almacenados permanentemente en el repositorio).
7	c	La naturaleza distribuida de Git permite que las operaciones se realicen localmente, lo que mejora el rendimiento y evita retrasos asociados a la comunicación constante con un servidor central.
8	c	El área de staging (índice) es donde se guardan temporalmente los cambios seleccionados para que formen parte de la siguiente confirmación, permitiendo revisar y agrupar modificaciones antes de almacenarlas en el repositorio.
9	b	Git Bash es una terminal disponible en Windows que permite ejecutar comandos propios de Git y emular un entorno similar al de Linux, facilitando la administración y el manejo de repositorios desde la línea de comandos.
10	d	Los componentes clave de Git incluyen el repositorio, las confirmaciones, las ramas y la fusión. Git no depende de un servidor central, ya que su enfoque distribuido permite que cada copia local contenga todo el historial.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Solucionario 3.-

Tabla 3: Solucionario unidad 3

# pregunta	Respuesta	Retroalimentación
1	b	La contenerización permite empaquetar aplicaciones junto con todas sus dependencias en contenedores, asegurando que se ejecuten de manera consistente en diferentes entornos.
2	c	Los contenedores aíslan cada aplicación en su propio entorno sin necesidad de un sistema operativo completo, lo que reduce el consumo de recursos y evita conflictos de dependencias.
3	b	Docker es una plataforma de código abierto que permite gestionar contenedores de manera eficiente, facilitando el desarrollo y despliegue de aplicaciones en cualquier entorno.
4	c	A diferencia de las máquinas virtuales, Docker permite que múltiples contenedores compartan el mismo kernel del sistema operativo, reduciendo el consumo de recursos y mejorando el rendimiento.
5	b	Kubernetes es un sistema de orquestación que automatiza la gestión de contenedores, permitiendo escalabilidad, balanceo de carga y despliegues eficientes en entornos de producción.
6	a	Los contenedores incluyen todas las dependencias necesarias para ejecutar una aplicación, lo que los hace portátiles entre distintos entornos sin cambios en la configuración.
7	c	Dockerd es el demonio de Docker que gestiona la ejecución de contenedores, creando y supervisando instancias en segundo plano.
8	b	Docker Hub es un servicio en la nube que permite almacenar y distribuir imágenes de Docker, facilitando la reutilización y el despliegue de aplicaciones.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

9	a	Docker Compose permite definir y ejecutar múltiples contenedores con un solo archivo YAML, facilitando la administración de aplicaciones con varias dependencias.
10	b	Una imagen en Docker es una plantilla con las instrucciones necesarias para ejecutar una aplicación, mientras que un contenedor es una instancia activa basada en esa imagen.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

10. BIBLIOGRAFÍA

Amazon Web Services. (2021). *Contenedores de Docker | ¿Qué es Docker?* Obtenido de Contenedores de Docker | ¿Qué es Docker?: https://aws.amazon.com/es/docker/

Atlassian. (14 de 08 de 2024). *Las 5 mejores herramientas de CI/CD para tu equipo de DevOps*. Obtenido de Las 5 mejores herramientas de CI/CD para tu equipo de DevOps: https://www.atlassian.com/es/devops/devops-tools/cicd-tools

atlassian. (s.f.). *atlassian*. Obtenido de https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

Chacon, S., & Straub, B. (2024). *Pro Git.* USA: Apress.

DataScientest.com. (2008). Docker surgió de la empresa dotCloud. *Reclu IT* .

Docker. (2021). *Documentación de Docker.* https://docs.docker.com/.

Editorial Etecé. (19 de 03 de 2023). *Concepto*. Obtenido de Concepto: https://concepto.de/historia-de-la-computadora/

EDTeam. (2022). *Entornos de trabajo en el desarrollo de software*. Obtenido de https://ed.team/blog/entornos-de-trabajo-en-el-desarrollo-de-software-44268387-202d-4dec-98d4-b5d31c5aa95a

git-scm. (s.f.). *git-scm*. Obtenido de https://git-scm.com/book/en/v2

IBM. (2023). *Qué es Docker?* Obtenido de Qué es Docker?: https://www.ibm.com/mx-es/topics/docker

IONOS. (2023). *Tutorial de Docker: introducción a la popular plataforma de contenedores*. Obtenido de Tutorial de Docker: introducción a la popular plataforma de contenedores: https://www.ionos.com/es-us/digitalguide/servidores/configuracion/tutorial-docker-instalacion-y-primeros-pasos/

Iradier, Á., & Martínez, I. (2021). *Docker para DevOps: de noob a experto* . Madrid: TecnologiaEspublico.

Lascano, E. (2003). *Componentes y Operación de Sistemas.* . Quito : Escuela Politécnica Nacional. doi:https://doi.org/10.1016/B978-1-55860-808-5.X5000-X

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Martin, A. (2021). *Metodologías ágiles para el desarrollo de software.* Repositorio UCLA.

vultr. (2024). *vultr*. Obtenido de https://docs.vultr.com/how-to-install-docker-on-ubuntu-24-04

Yummerlys, L. (2022). *ed.team*. Obtenido de ed.team: https://ed.team/blog/entornos-de-trabajo-en-el-desarrollo-de-software-44268387-202d-4dec-98d4-b5d31c5aa95a

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC