Programa DevOps

UMSA
UNIVERSIDAD
DEL MUSEO SOCIAL ARGENTINO

Softtek®

DevOps e Infraestructura

2. Automatización, Herramientas e Infraestructura

En la era digital, este manual sobre Automatización, Herramientas e Infraestructura ofrece una guía para solucionar y enfrentar los desafíos DevSecOps. Desde sistemas operativos hasta scripting y redes avanzadas, exploramos herramientas clave como Jenkins, SonarQube y GitHub/GitLab en el ámbito de CI/CD. Diseñado para la eficiencia, este recurso proporciona las habilidades necesarias en un mundo tecnológico en constante cambio. Descubre cómo mejorar tu destreza y mantenerte al día con las últimas tendencias en esta exploración detallada de la automatización y la infraestructura digital.

¡Manos a la obra! 🚀

🎯Objetivos de aprendizaje

- Comprender cómo funcionan los principales Sistemas Operativos y la Virtualización de estos.
- Conocer los principios de las redes informáticas.
- Manejar los fundamentos del Scripting con diferentes lenguajes.
- Tener contacto con las herramientas de CI CD y ser capaz de generar Pipelines y Releases.

🗒️Índice

2.1 Sistemas Operativos y Virtualización

Un Sistema Operativo (SO) es un software fundamental que actúa como intermediario entre el hardware de una computadora y las aplicaciones que se ejecutan en ella. Su papel principal es gestionar los recursos del sistema y proporcionar una interfaz para que los usuarios interactúen con la máquina de manera eficiente.

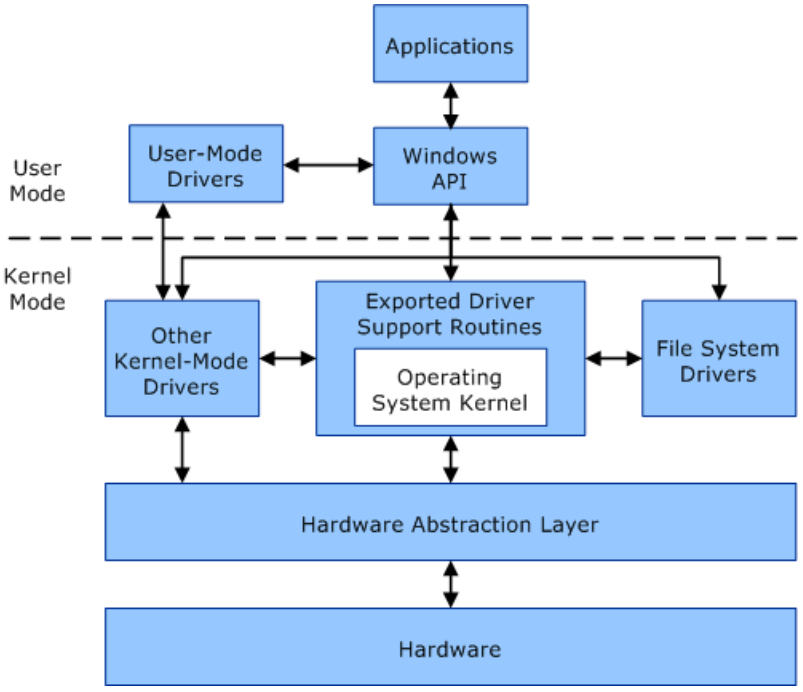
Dentro de los sistemas operativos más utilizados se encuentran
(En orden de cantidad de instalaciones):

- Android
- Microsoft Windows
- Linux

- macOS
- iOS

Nos vamos a enfocar en los sistemas operativos que, como un ingeniero que utiliza DevSecOps podría llegar a interactuar en entornos corporativos. Los más comunes son solo dos, con sus respectivas distribuciones y versiones que pueden ser variadas. Cabe destacar que esto es solo una apreciación basada en cantidad de Sistemas Operativos instalados en ambientes corporativos. Esto no indica que no existan casos particulares de empresas que utilicen macOs, Android u otros sistemas operativos.

2.1.1. Windows



Aspecto	Información
Historia breve	Familia de sistemas operativos desarrollada por Microsoft. La primera versión, fue lanzada en 1985. Que corría sobre DOS, desde entonces, ha experimentado varias actualizaciones y mejoras, siendo la versión más reciente Windows 11.
Arquitectura	Windows utiliza una arquitectura de sistema operativo híbrida que incluye un núcleo (kernel) monolítico y componentes adicionales en modo usuario y en modo kernel.
Sistema de archivos	Los sistemas de archivos principales utilizados por Windows son NTFS (New Technology File System) para las versiones más recientes y FAT32 (File Allocation Table) en versiones más antiguas.

Aspecto	Información
Manejo de Memoria	Windows gestiona la memoria mediante técnicas como paginación y segmentación. También soporta la memoria virtual para optimizar el uso de recursos.
Seguridad y protección	Windows cuenta con una variedad de características de seguridad, como cortafuegos, antivirus integrado (Windows Defender), control de cuentas de usuario (UAC), y actualizaciones regulares para abordar vulnerabilidades.
Línea de comandos	La línea de comandos en Windows se realiza a través del intérprete de comandos cmd.exe y, más recientemente, PowerShell, que es una interfaz de línea de comandos más avanzada y potente.
Ejecución multiplataforma	Para abordar algunas de estas limitaciones, Microsoft introdujo WSL, un subsistema que permite la ejecución de binarios de Linux directamente en Windows. WSL incluye un kernel de Linux, lo que facilita la ejecución de aplicaciones de Linux sin necesidad de una máquina virtual.

Estructura de Carpetas en Windows

La estructura de carpetas en Windows puede variar ligeramente dependiendo de la versión específica del sistema operativo, pero a grandes rasgos, sigue un patrón similar. Aquí hay una descripción general de la estructura de carpetas común en Windows:

Carpeta del Sistema (System Drive - C:):

La unidad principal donde se instala el sistema operativo. Contiene carpetas clave como "Windows" y "Archivos de Programa."

Windows:

La carpeta que alberga todos los archivos del sistema operativo Windows. Contiene subcarpetas como "System32" (archivos del sistema), "Program Files" (aplicaciones instaladas), y "Users" (perfiles de usuario).

Archivos de Programa (Program Files):

Aquí se instalan las aplicaciones y programas. En sistemas de 64 bits, también hay una carpeta "Program Files (x86)" para aplicaciones de 32 bits.

Usuarios (Users):

Contiene perfiles de usuario y carpetas personales para cada cuenta de usuario en el sistema. Dentro de esta carpeta, encontrarás carpetas como "Desktop," "Documents," "Downloads," y más para cada usuario.

Archivos de Programa Comunes (Common Files):

Contiene archivos compartidos utilizados por varias aplicaciones instaladas en el sistema.

AppData:

Almacena datos de configuración y archivos temporales para aplicaciones. Se encuentra dentro de la carpeta de cada usuario.

System32:

Contiene archivos del sistema esenciales. Es una carpeta crítica para el funcionamiento del sistema.

WinSxS (Windows Side-by-Side):

Almacena componentes de sistema en diferentes versiones, permitiendo la coexistencia de varias versiones de un mismo componente.

ProgramData:

Similar a "AppData," pero a nivel del sistema. Almacena datos compartidos por todas las cuentas de usuario.

Temp:

Una carpeta temporal que se utiliza para almacenar archivos temporales. Puede ser limpiada periódicamente.

Recycle Bin:

Almacena los archivos eliminados hasta que se vacía la papelera de reciclaje.

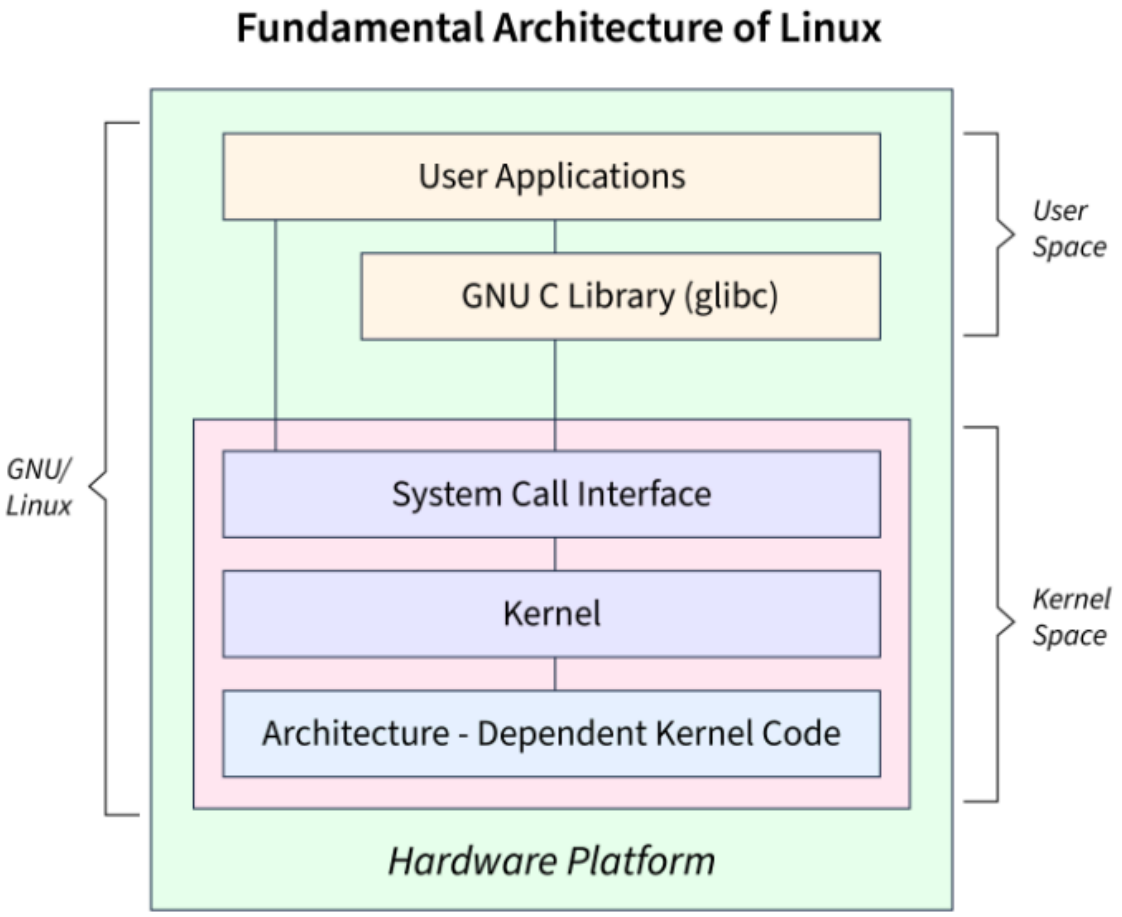
PerfLogs (Performance Logs):

Almacena datos de rendimiento y registros de monitoreo del sistema.

🏃 Quick Learning Arquitectura Windows

Visualiza el siguiente video en Youtube
https://www.youtube.com/watch?v=Mu_JHHvUOKk
Duración: 50:47

- ## 2.1.2. Linux



Aspecto	Información
Historia breve	Linux, desarrollado por Linus Torvalds en 1991, es un sistema operativo de código abierto basado en UNIX. Su desarrollo colaborativo ha llevado a la creación de numerosas distribuciones, como Ubuntu, Fedora y Debian.
Arquitectura	Linux es compatible con diversas arquitecturas, incluyendo x86, x86-64 (64 bits), ARM, MIPS, PowerPC, entre otras. La flexibilidad de Linux lo hace adecuado para una amplia gama de dispositivos y sistemas.
Sistema de archivos	Las distribuciones de Linux utilizan varios sistemas de archivos, siendo ext4 el más común en sistemas de escritorio y servidores. Otros incluyen Btrfs, XFS, y sistemas de archivos específicos para ciertos usos como ZFS.
Manejo de Memoria	Linux administra la memoria a través de técnicas como paginación y segmentación. El kernel de Linux también es compatible con la memoria virtual, lo que mejora la eficiencia en el uso de recursos.
Seguridad y protección	Linux es conocido por su robusta seguridad. Se basa en el modelo de permisos y control de acceso, donde cada usuario y proceso tiene sus propios permisos. El modelo de seguridad se apoya en principios como el mínimo privilegio.

Aspecto	Información
Línea de comandos	La interfaz de línea de comandos es una parte integral de Linux. Utiliza shells como Bash para la interacción del usuario. La línea de comandos permite realizar tareas administrativas, automatización y scripting.
Ejecución multiplataforma	Linux es capaz de ejecutar binarios desarrollados para diversas plataformas a través de emuladores y herramientas de compatibilidad. Como por ejemplo WINE También es compatible con máquinas virtuales y contenedores, facilitando la ejecución de software diseñado para diferentes entornos.

Estructura de Carpetas en Linux

La estructura de carpetas en Linux sigue una jerarquía del sistema de archivos estándar, que es una convención común para organizar archivos y directorios en sistemas basados en Unix. Aquí hay una descripción general de la estructura de carpetas en Linux:

- **Raíz del Sistema (/):**

La carpeta principal que contiene todos los demás archivos y directorios.

- **/bin (Binary):**

Contiene binarios (archivos ejecutables) esenciales para el inicio y operación del sistema.

- **/boot:**

Contiene archivos relacionados con el proceso de arranque del sistema, como el kernel del sistema y archivos de configuración del gestor de arranque.

- **/dev (Device):**

Contiene archivos de dispositivos, que representan dispositivos de hardware o pseudo-dispositivos.

- **/etc (Etcétera):**

Almacena archivos de configuración del sistema y de las aplicaciones.

- **/home:**

Contiene los directorios de inicio de los usuarios. Cada usuario tiene su propia carpeta dentro de /home.

- **/lib (Library):**

Contiene bibliotecas compartidas esenciales utilizadas por programas del sistema y aplicaciones.

- **/media:**

Punto de montaje para dispositivos de almacenamiento extraíbles, como CD-ROMs o unidades USB.

- **/mnt (Mount):**

Punto de montaje para sistemas de archivos temporales o adicionales.

- **/opt (Optional):**

Se utiliza para la instalación de paquetes de software adicionales.

- **/proc (Process):**

Contiene información sobre los procesos y el sistema, presentada como archivos virtuales.

- **/root:**

El directorio de inicio del usuario root, el superusuario del sistema.

- **/run:**

Contiene datos variables de ejecución utilizados por el sistema y las aplicaciones.

- **/sbin (System Binary):**

Similar a /bin, pero contiene binarios específicos del sistema utilizados para la administración del sistema.

- **/srv (Service):**

Contiene datos de servicios proporcionados por el sistema.

- **/tmp (Temporary):**

Almacena archivos temporales. El contenido de esta carpeta se borra regularmente.

🔸 **/usr (User):**

Contiene archivos y programas utilizados por usuarios del sistema, incluyendo binarios, bibliotecas y archivos de encabezado.

🔸 **/var (Variable):**

Contiene datos variables, como registros, archivos temporales y archivos de cola de correo.

🏃 **Quick Learning Comenzando con Linux**

Visualiza el siguiente video en Youtube
https://www.youtube.com/watch?v=knrc4q1S_q0
Duración: 43:50

● 2.1.3. Virtualización

La virtualización de sistemas operativos es una tecnología que permite la creación de entornos virtuales independientes dentro de un único sistema físico. Estos entornos virtuales, conocidos como máquinas virtuales (VM), pueden ejecutar sistemas operativos completos de forma aislada y simultánea en el mismo hardware físico.

Hipervisor

Un hypervisor (o VMM, Monitor de Máquina Virtual) es el componente central que permite la creación y gestión de máquinas virtuales. Hay dos tipos principales de hipervisores:

🟢**Tipo 1 (Bare Metal):** Se instalan directamente en el hardware físico y gestionan las VM sin necesidad de un sistema operativo anfitrión adicional. Ejemplos incluyen VMware ESXi y Microsoft Hyper-V.

🟢**Tipo 2 (Anfitrión):** Se ejecutan como aplicaciones dentro de un sistema operativo anfitrión existente. Ejemplos incluyen VMware Workstation y Oracle VirtualBox.

Máquinas Virtuales (VM)

Una máquina virtual es una instancia independiente de un sistema operativo que se ejecuta en una capa de virtualización. Cada VM tiene sus recursos virtuales asignados, como CPU, memoria RAM, almacenamiento y adaptadores de red. Pueden ejecutar sistemas operativos diferentes en el mismo hardware físico.

Hay varias opciones de software de virtualización de máquinas que son ampliamente utilizadas y reconocidas en la industria. La elección del software dependerá de tus necesidades específicas y de los requisitos del sistema. Aquí tienes algunas opciones populares:

- **VMware:**

VMware vSphere/ESXi: es uno de los líderes en virtualización. vSphere es la plataforma de virtualización integral que incluye ESXi como hipervisor tipo 1. VMware ofrece características avanzadas y herramientas de gestión potentes, pero algunas de sus soluciones pueden tener costos asociados.

- **Microsoft:**

Hyper-V es el hipervisor de Microsoft incluido en Windows Server. Ofrece una buena integración con el ecosistema de Microsoft y es una opción sólida si estás utilizando entornos basados en Windows. También hay una versión gratuita llamada Hyper-V Server.

- **Oracle:**

Oracle VM VirtualBox: Es un hipervisor tipo 2 que es gratuito y de código abierto. Es adecuado para entornos de desarrollo y pruebas y es compatible con una amplia variedad de sistemas operativos invitados.

- **Citrix:**

Hypervisor (anteriormente XenServer): Ofrece características de virtualización robustas y es conocido por su rendimiento. Citrix Hypervisor es gratuito y de código abierto con características avanzadas disponibles en su versión comercial.

- **KVM/QEMU:**

Kernel-based Virtual Machine (KVM) con QEMU: KVM es un módulo del kernel de Linux que permite la virtualización de hardware. QEMU es un emulador de procesador que

funciona junto con KVM. Juntos, proporcionan una solución de virtualización potente y de código abierto.

- **Proxmox Virtual Environment:**

Proxmox VE: Es una plataforma de virtualización de código abierto que combina KVM para la virtualización y LXC para la contenerización. Ofrece una interfaz web de gestión fácil de usar.

- **Xen Project:**

Es un hipervisor de código abierto que se utiliza en entornos empresariales y de nube. Ofrece para virtualización y virtualización completa.

- **Vagrant:**

Si te centras en entornos de desarrollo y pruebas, Vagrant puede ser útil. No es un hipervisor en sí mismo, sino una herramienta que facilita la creación y gestión de entornos de desarrollo virtualizados.

🏃 Quick Learning ¿Qué es la Virtualización?

Visualiza el siguiente video en Youtube
https://www.youtube.com/watch?v=K1vPbQtOlSc
Duración: 12:34

🏃 Quick Learning Aprende a utilizar Máquinas Virtuales

Visualiza el siguiente video en Youtube
https://www.youtube.com/watch?v=uiFZUfmFAus
Duración: 28:39

2.2 Introducción a Redes

2.2.1. Conceptos Básicos

Definición de Networking:

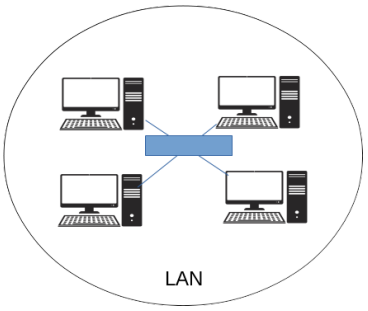
Se refiere a la práctica de conectar dispositivos y sistemas para compartir recursos y facilitar la comunicación. Es la disciplina que se ocupa del diseño, implementación, gestión y mantenimiento de sistemas de comunicación de datos.

Las redes de computadoras se pueden clasificar en función de diferentes atributos como tamaño físico, modo de conexión, tipos de dispositivos, etc. Para una introducción sencilla, veamos su clasificación en términos de tamaño físico. A menudo escuchará lo siguiente.

- Red de área local (LAN)
- Red de área metropolitana (MAN)
- Red de área amplia (WAN)

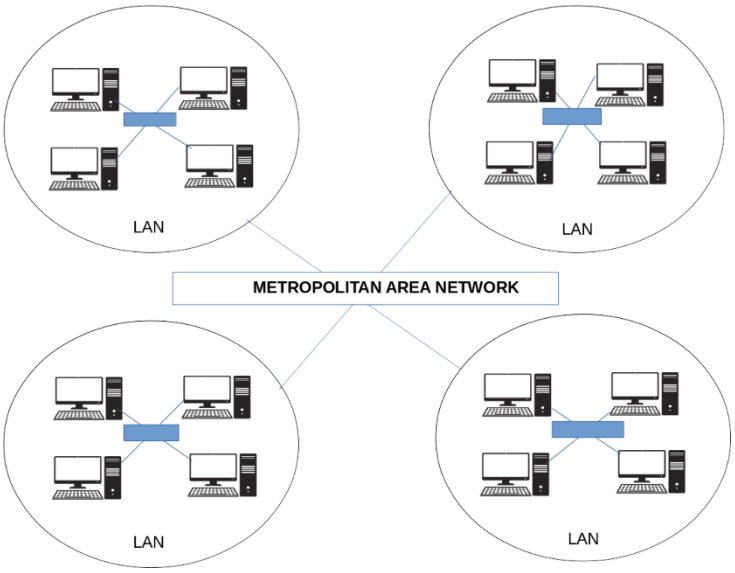
Red de área local (LAN)

LAN un tipo de red informática que se ocupa usualmente de un área pequeña y mayoritariamente restringida a un solo lugar; por ejemplo, la conexión en red de un cibercafé, un laboratorio de informática o simplemente el edificio de una organización. La conexión en red de una casa o un edificio pequeño se conoce como SOHO LAN (que significa red de área local de oficinas domésticas pequeñas). En este tipo de redes, generalmente hay pocas computadoras y otros dispositivos de red, por ejemplo, conmutadores y enrutadores involucrados



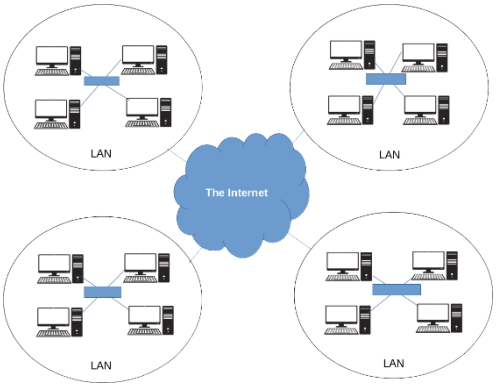
Red de área metropolitana (MAN)

MAN es un tipo de red informática que es más grande que una LAN pero que no llega a una WAN (Red de área amplia). Es un intermediario entre LAN y WAN. Se extiende por más áreas que en LAN y tiene una mayor cantidad de computadoras; también puede verse como una combinación de más de una LAN.
En este tipo, el uso de enrutadores se emplea bien para la segregación e integración de redes, y requiere más controles administrativos debido a su tamaño.



Red de área amplia (WAN)

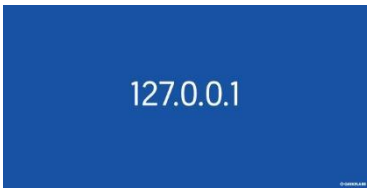
Esta es la forma más grande de red informática que abarca miles de hectáreas de superficie terrestre. Puede cruzar regiones y países; No hay límite para la cantidad de dispositivos o recursos informáticos y de red involucrados. WAN emplea el uso de todos los dispositivos de red necesarios como conmutadores, enrutadores, concentradores, puentes, repetidores, etc. Este tipo de redes suele requerir monitoreo administrativo a tiempo completo, ya que su falla puede provocar una pérdida o robo masivo de datos. Un ejemplo fácilmente familiar es Internet.



Direcciones IP

IP significa Protocolo de Internet.

Las direcciones IP (Internet Protocol) son identificadores numéricos únicos asignados a cada dispositivo conectado a una red que utiliza el protocolo de Internet para la comunicación. Estas direcciones son esenciales para el enrutamiento de datos en redes, permitiendo que los dispositivos se comuniquen entre sí en el mundo digital.



IPv4 e IPv6

IPv4 (Protocolo de Internet versión 4):

- **Formato:** Se compone de cuatro bloques de números decimales separados por puntos (por ejemplo, 192.168.1.1).

- **Cantidad de Direcciones:** IPv4 proporciona alrededor de 4.3 mil millones de direcciones únicas, pero la creciente demanda ha llevado a la adopción de IPv6.

IPv6 (Protocolo de Internet versión 6):

- **Formato:** Utiliza una notación hexadecimal y tiene una estructura más compleja (por ejemplo, 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

- **Cantidad de Direcciones:** IPv6 ofrece una cantidad virtualmente ilimitada de direcciones únicas, diseñadas para abordar la escasez de direcciones IPv4.

Tipos de Direcciones IP

1. Direcciones IP Públicas:

- **Únicas en Internet:** Son asignadas de manera global y son únicas en la Internet pública.
- **Accesibles desde la Internet:** Los dispositivos con direcciones IP públicas son accesibles directamente desde la red global.

2. Direcciones IP Privadas:

- **Utilizadas en Redes Privadas:** Reservadas para su uso en redes locales y no son accesibles directamente desde Internet.

- **Ejemplos:** Rangos de direcciones IP privadas comunes son 192.168.x.x, 172.16.x.x - 172.31.x.x y 10.x.x.x.

Direcciones IP Estáticas y Dinámicas:

- **Estáticas:** Asignadas manualmente y permanecen constantes. Suelen utilizarse para servidores o dispositivos que necesitan una identificación constante.
- **Dinámicas:** Asignadas automáticamente por un servidor DHCP (Protocolo de Configuración Dinámica de Host). Cambian periódicamente.

Estructura de una Dirección IP:

- **Octetos:** Las direcciones IP se dividen en octetos, cada uno representado por un número decimal.

- **Máscara de Subred:** Define la red y la parte del host en una dirección IP. Ayuda en la segmentación de redes.

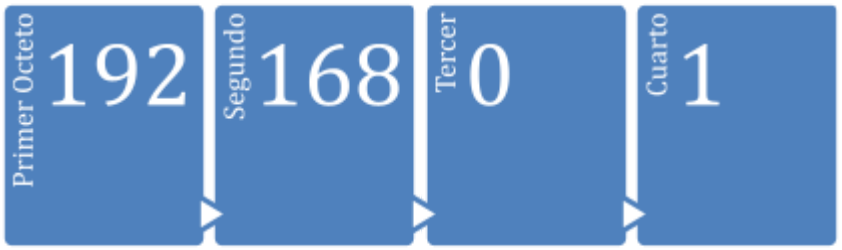
- **Ejemplo de IPv4:** 192.168.1.1 (cuatro octetos separados por puntos).

- **Ejemplo de IPv6:** 2001:0db8:85a3:0000:0000:8a2e:0370:7334 (notación hexadecimal).

Direcciones IPv4

Una dirección IPv4 está formada por 32 bits, que se representan comúnmente en formato decimal puntado (dotted-decimal). Los 32 bits se dividen en cuatro octetos, y cada octeto se representa como un número decimal separado por puntos. Cada octeto puede tener un valor entre 0 y 255.

Ejemplo de una dirección IPv4 en formato decimal puntado: 192.168.0.1



Estructura:

Cada uno de los cuatro octetos se representa en binario (0 o 1).
Ejemplo de dirección IP en binario: 11000000.10101000.00000000.00000001
Formato Decimal Puntado:

Cada octeto se convierte a decimal y se separa por puntos.
En el ejemplo: 192.168.0.1
Además, es importante destacar que cada dirección IP consta de dos partes principales: la red y el host. La máscara de subred se utiliza para determinar la división entre estas dos partes.

Cada octeto en una dirección IP tiene un propósito específico y se utiliza para identificar la red y el host en esa red. La manera en que se asignan los octetos a la red y al host depende del tipo de dirección IP: A, B o C.

Direcciones IP Clase A:

- El primer octeto (N) identifica la red, y los tres octetos restantes (H) identifican hosts en esa red.
- Rango del primer octeto: 1 a 126.
- Ejemplo: **10.0.0.1**

Direcciones IP Clase B:

- Los dos primeros octetos (N) identifican la red, y los dos octetos restantes (H) identifican hosts en esa red.
- Rango del primer octeto: 128 a 191.
- Ejemplo: **172.16.0.1**

Direcciones IP Clase C:

- Los tres primeros octetos (N) identifican la red, y el último octeto (H) identifica hosts en esa red.
- Rango del primer octeto: 192 a 223.
- Ejemplo: **192.168.0.1**
- **Uso de los Octetos:**
- **Red (N):** Los primeros octetos identifican la red a la que pertenece el dispositivo.
- **Host (H):** Los últimos octetos identifican el host único dentro de esa red.

Direcciones IP Privadas y Públicas:

- **Privadas:** Son direcciones IP reservadas para uso interno en redes privadas. Ejemplos son las direcciones IP que comienzan con **10.x.x.x**, **172.16.x.x** hasta **172.31.x.x** y **192.168.x.x**. Son comúnmente utilizadas en redes locales.
- **Públicas:** Son direcciones IP únicas en la Internet global. Están asignadas por la autoridad de asignación de direcciones IP. Las direcciones IP que no pertenecen a los rangos privados son públicas.

Otros tipos de Direcciones

Además de las clases A, B y C, existen otras clases de direcciones IP, pero son menos comunes y no se utilizan tan ampliamente en la actualidad. Estas clases incluyen las clases D y E.

Direcciones IP Clase D:

- **Formato: N.N.N.N**
- Rango del primer octeto: 224 a 239.
- Utilizadas para multicast, que es la transmisión de datos a múltiples receptores simultáneamente.
- Ejemplo: **239.1.1.1**

Direcciones IP Clase E:

- **Formato: N.N.N.N**
- Rango del primer octeto: 240 a 255.
- Reservadas para uso experimental y no se utilizan en la práctica general.
- No se asignan para su uso en redes públicas o privadas.

Es importante destacar que, en la práctica, las clases A, B y C son las más relevantes y utilizadas en la mayoría de las redes. Las direcciones de clase D se reservan para multicast, y las direcciones de clase E son reservadas para propósitos experimentales y no se utilizan en situaciones normales de redes comerciales.

Además, con la introducción de CIDR (Classless Inter-Domain Routing) y la creciente adopción de IPv6, que utiliza un formato completamente diferente de direcciones, la importancia de las clases de direcciones IP ha disminuido. CIDR permite una asignación de direcciones IP más flexible y eficiente al no depender estrictamente de las clases A, B o C. IPv6, por otro lado, proporciona un espacio de direcciones mucho más grande y no utiliza la clasificación de clases.

Subredes

Una subred (subnetwork o subred en inglés) es una porción de una red IP que ha sido dividida en segmentos más pequeños. La subdivisión de una red en subredes se realiza mediante el uso de máscaras de subred, lo cual permite la creación de redes más eficientes y la optimización en el uso de direcciones IP.

Máscara de Subred:

Es una combinación de bits que se utiliza para dividir la dirección IP en dos partes: la parte de la red y la parte del host. La máscara de subred se aplica mediante la operación lógica AND a la dirección IP.

Componentes de una máscara de subred:

- Bits de Red: Representan la parte de la dirección IP que identifica la red.
- Bits de Host: Representan la parte de la dirección IP que identifica los dispositivos (hosts) dentro de esa red.
- Valor de los Bits: Pueden ser 0 o 1, donde 1 indica que ese bit se utiliza para la red y 0 indica que se utiliza para el host.
- Ejemplos de máscaras de subred:
  255.255.255.0 (o /24 en notación CIDR): Reserva los primeros 24 bits para la red y deja 8 bits para hosts, permitiendo hasta 256 direcciones en la red.
  255.255.255.128 (o /25 en notación CIDR): Reserva los primeros 25 bits para la red y deja 7 bits para hosts, permitiendo hasta 128 direcciones en la red.

División en Subredes:

- Permite dividir una red en segmentos más pequeños para mejorar la eficiencia y la gestión de direcciones IP.
- Cada subred tiene su propia gama de direcciones IP disponibles.

Ventajas:

- **Optimización de Recursos:** Permite un uso más eficiente de las direcciones IP disponibles.
- **Segmentación de Red:** Facilita la administración y el rendimiento de la red al segmentarla en partes más pequeñas.
- **Mejora del Rendimiento:** Reduce el tráfico de broadcast al limitar su alcance a una subred específica.

Ejemplo:

- Si tienes una red con la dirección IP **192.168.0.0** y aplicas una máscara de subred de **255.255.255.128**, estarás dividiendo la red en dos subredes (**192.168.0.0** y **192.168.0.128**), cada una con su propio rango de direcciones IP disponibles.

CIDR (Classless Inter-Domain Routing):

- "Classless Inter-Domain Routing" (Enrutamiento Interdominio sin Clases), es una notación que se utiliza para representar bloques de direcciones IP y sus correspondientes máscaras de subred de manera más flexible y eficiente que el sistema de clases tradicional (Clase A, B, C).

- En lugar de depender de las divisiones rígidas en clases (A, B, C), CIDR utiliza una notación que combina la dirección IP y el número de bits utilizados para la máscara de subred. La notación CIDR toma la forma de direcciónIP/máscara.

- Por ejemplo, **192.168.0.0/24** representa la red **192.168.0.0** con una máscara de subred de **255.255.255.0**.

Calcular número de Hosts de una red

Se basa en el concepto de contar el número de direcciones IP disponibles para dispositivos dentro de esa red.

Prefijo (CIDR)	Máscara de Subred	Hosts Disponibles	Número total de Direcciones IP	Número total de Subredes

/24	255.255.255.0	254	256	1
/25	255.255.255.128	126	128	2
/26	255.255.255.192	62	64	4
/27	255.255.255.224	30	32	8
/28	255.255.255.240	14	16	16
/29	255.255.255.248	6	8	32

🔵Paso 1: Entender la Máscara de Subred

La máscara de subred es clave para determinar la estructura de la red y cuántos host puede haber. La máscara de subred se representa en formato decimal puntado (por ejemplo, **255.255.255.0**) o en notación CIDR (por ejemplo, **/24**). La notación CIDR especifica cuántos bits de la dirección IP están reservados para la red.

🔵Paso 2: Calcular el Número de Bits para Hosts

Resta el número de bits reservados para la red de la longitud total de la dirección IP (32 bits en el caso de IPv4). Los bits restantes se utilizan para identificar los hosts.
Por ejemplo, si tienes una máscara de subred de **/24** (o **255.255.255.0**), entonces hay 32 - 24 = 8 bits disponibles para hosts.

🔵Paso 3: Calcular el Número de Direcciones para Hosts

El número de direcciones para hosts es 2^n, donde "n" es el número de bits para hosts. Esto se debe a que cada bit puede tener dos valores (0 o 1).
En el ejemplo de una máscara de subred **/24**, tendrías 2^8 = 256 direcciones para hosts.

🔵Paso 4: Restar las Direcciones de Red y de Broadcast

En una red, las direcciones de red y de broadcast no se asignan a hosts, así que estas deben restarse del total calculado en el paso anterior. Así obtienes el número real de direcciones IP utilizables para hosts.
Por ejemplo, si tienes 256 direcciones para hosts, pero una se usa para la dirección de red y otra para la dirección de broadcast, entonces tendrías 256 - 2 = 254 direcciones IP utilizables para hosts.

Ejemplo:

Supongamos que tienes la siguiente máscara de subred: **255.255.255.128** o **/25**. Calculamos:

- Bits para hosts: 32 - 25 = 7 bits.
- Direcciones para hosts: 2^7 = 128 direcciones.
- Direcciones utilizables para hosts: 128 - 2 (dirección de red y de broadcast) = 126 direcciones.

Entonces, en esta red, habría 126 direcciones IP utilizables para dispositivos (hosts). Este proceso se aplica a cualquier máscara de subred, ya sea que esté representada en formato decimal punteado o en notación CIDR.

Claro está, también es mucho más fácil utilizar una herramienta online para calcular los mismos 😉 👉🏽IP Subnet Calculator | IPTP Networks

Direcciones Reservadas

Existen ciertos rangos de direcciones IP que están reservados para usos especiales y no deben ser utilizados en redes públicas de Internet. Estas direcciones IP están destinadas para casos específicos, como redes privadas, pruebas, multicast, y otros propósitos especiales. Algunos de los rangos de direcciones IP reservadas son:

🔹 Direcciones IP Privadas (No Ruteables en Internet):

- **Clase A:** 10.0.0.0 a 10.255.255.255
- **Clase B:** 172.16.0.0 a 172.31.255.255
- **Clase C:** 192.168.0.0 a 192.168.255.255

Estas direcciones IP están reservadas para su uso en redes privadas y no deben ser enrutadas a través de Internet.

🔹 Direcciones de Loopback:

- **Loopback:** 127.0.0.0 a 127.255.255.255
- Ejemplo: 127.0.0.1 (localhost)

Estas direcciones están reservadas para el loopback o comunicación interna de un dispositivo consigo mismo.

- **Direcciones IP de Documentación (Uso en Documentación y Pruebas):**
    - **IPv4 Documentación:** 192.0.2.0 a 192.0.2.255
    - **Test-Net:** 198.51.100.0 a 198.51.100.255
    - **IPv6 Documentación:** 2001:DB8:: a 2001:DB8:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF

Estas direcciones están destinadas para documentación y pruebas y no deben ser utilizadas en redes reales.

- **Direcciones IP Multicast:**
    - **Multicast:** Rango específico, como 224.0.0.0 a 239.255.255.255

Estas direcciones se utilizan para transmitir datos a múltiples receptores simultáneamente.

- **Dirección de Broadcast para Redes IPv4:**
    - **Broadcast IPv4:** 255.255.255.255
    - Se utiliza para transmitir datos a todos los dispositivos en la red.

Estas direcciones IP reservadas ayudan a evitar conflictos y confusiones en la asignación de direcciones en Internet y se utilizan para propósitos específicos según las normativas y estándares de la industria.

2.2.2. Modelo OSI

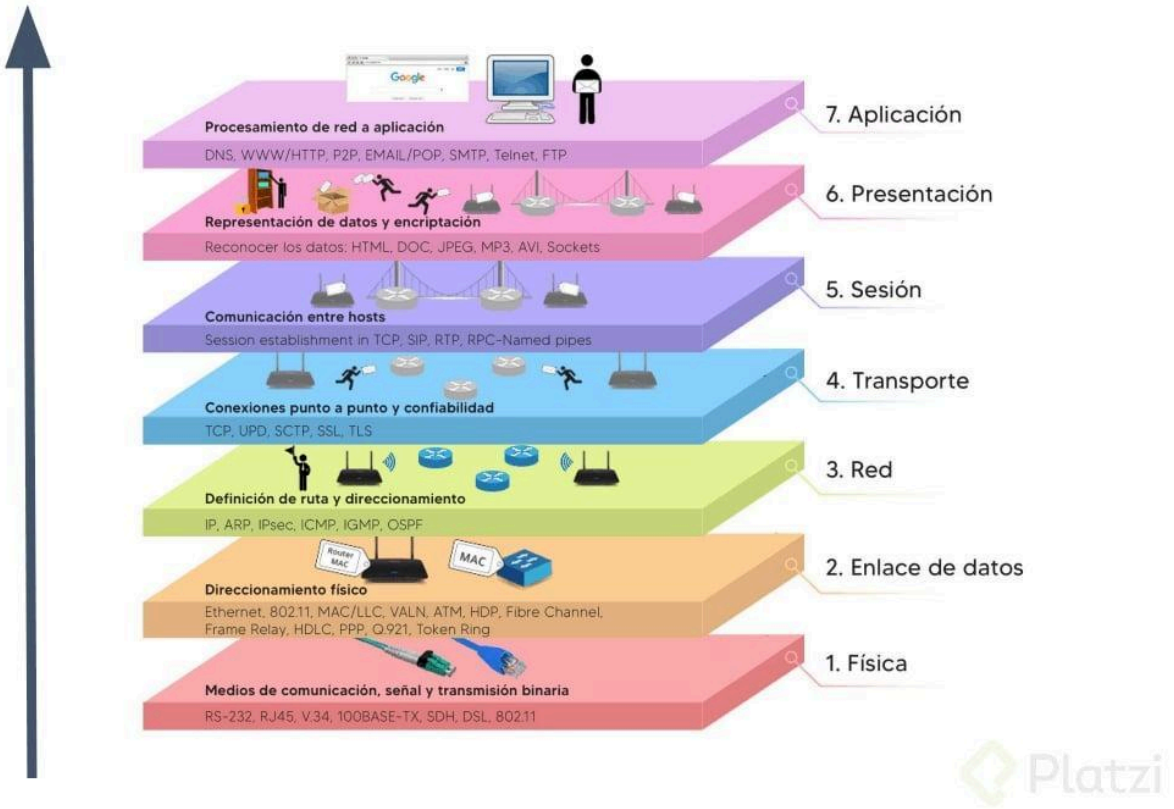
El **modelo OSI** es un marco utilizado para comprender cómo funcionan juntos los diferentes sistemas de comunicación. Divide el proceso de envío y recepción de datos a través de una red en siete capas, cada una de las cuales realiza una función específica.
Este fue desarrollado por la **Organización Internacional para la Estandarización** (ISO), a fines de la década de **1970**, como una manera de estandarizar el trabajo de distintos sistemas de comunicación y proporcionar una estructura lógica y organizada para el intercambio de datos.

¿Cómo funciona el modelo OSI?

El modelo cuenta con 7 capas que enumeraremos de abajo hacia arriba.

- Capa física
- Capa de enlace de datos

- Capa de red
- Capa de transporte
- Capa de sesión
- Capa de presentación
- Capa de aplicación



¿Cuáles son las siete capas del modelo OSI?

Este estándar de redes para computadoras se maneja por capas porque cada una involucra una tarea específica y se comunica con la capa inmediatamente inferior o superior para ejecutar su función y por eso a continuación **te explicaremos cada una de ellas de abajo hacia arriba.**

1. Capa física

La capa física se ocupa de la conexión física entre dispositivos. Por ejemplo, como la transmisión de datos a través de un cable de cobre o fibra óptica (electricidad, cables, hardware). Algunas de las funciones que tiene la capa física son:

- Establecer y mantener la conexión física entre dispositivos
- Especificar el tipo de medio de transmisión
- Controlar el flujo de datos
- Detectar y corregir errores
- Convertir datos en señales físicas

2. Capa de enlace de datos

Esta capa de enlace de datos es responsable de establecer y mantener un enlace entre dispositivos, asegurando que los datos se transmitan de manera precisa y eficiente. También realiza la verificación y corrección de errores para garantizar que los datos se reciban correctamente.

En esta capa:

Se envían los datos que se convirtieron a bits
Se le añade información sobre el direccionamiento físico
Se llega a la capa de red

3. Capa de red

De otro modo, la capa de red se encarga de enrutar datos entre dispositivos en una red, también determina la ruta más eficiente para que viajen los datos y garantiza que lleguen a su destino.
Algunas de las funciones de la capa de red son:

- Enrutamiento de datos
- Direccionamiento de dispositivos
- Fragmentación y reensamblaje de paquetes

- Control de congestión

4. Capa de transporte

Esta capa de transporte tiene el objetivo de garantizar que los datos se entreguen de manera confiable y en el orden correcto. Agrega control de flujo y comprobación de errores para garantizar que los datos no se pierdan ni se corrompan durante la transmisión. Se encuentran protocolos como TCP.
Funciones:

- Control de flujo
- Control de errores
- División y reensamblaje de paquetes (datos más pequeños)

5. Capa de sesión

La capa de sesión tiene como fin establecer, mantener y terminar las conexiones entre dispositivos. Permite que los dispositivos se comuniquen entre sí y coordinen sus actividades. Entre sus funciones está:

- Establecer conexiones
- Mantener esos enlaces
- Dar fin a esas conexiones

6. Capa de presentación

Esta capa de presentación es responsable de convertir los datos a un formato que pueda ser entendido por el dispositivo receptor. También maneja el cifrado y la compresión de

datos para garantizar que se transmitan de manera segura y eficiente. Formatea los datos para transferirlos a la siguiente capa.

Entre las funciones de la capa de presentación está:

- Codificación y decodificación de datos
- Compresión y descompresión de datos

7. Capa de aplicación

Esta capa de aplicación es el nivel más alto del modelo OSI y es responsable de brindar servicios al usuario. Permite a los usuarios interactuar con la red y acceder a los recursos que necesitan.

2.2.3. Protocolos de Red

Los protocolos de red son reglas y convenciones establecidas para la comunicación entre dispositivos en una red de computadoras. Estos protocolos definen cómo los datos se transmiten, reciben y procesan en una red, garantizando la interoperabilidad y la comunicación efectiva entre sistemas heterogéneos.

Algunos ejemplos de protocolos de red son:

- DNS (Domain Name System): Resuelve nombres de dominio a direcciones IP.
- HTTP/HTTPS (Hypertext Transfer Protocol/Secure): Utilizado para la transferencia de datos en la World Wide Web.
- TCP (Protocolo de Control de Transmisión).
- IP (Protocolo de Internet).
- UDP (Protocolo de Datagramas de Usuario).
- ICMP (Protocolo de Mensajes de Control de Internet), entre otros.

Pero no vamos a ver en detalle todos los protocolos, en cambio en cada capa del Modelo TCP/IP que explicaremos a continuación vamos a nombrar algunos de ellos.

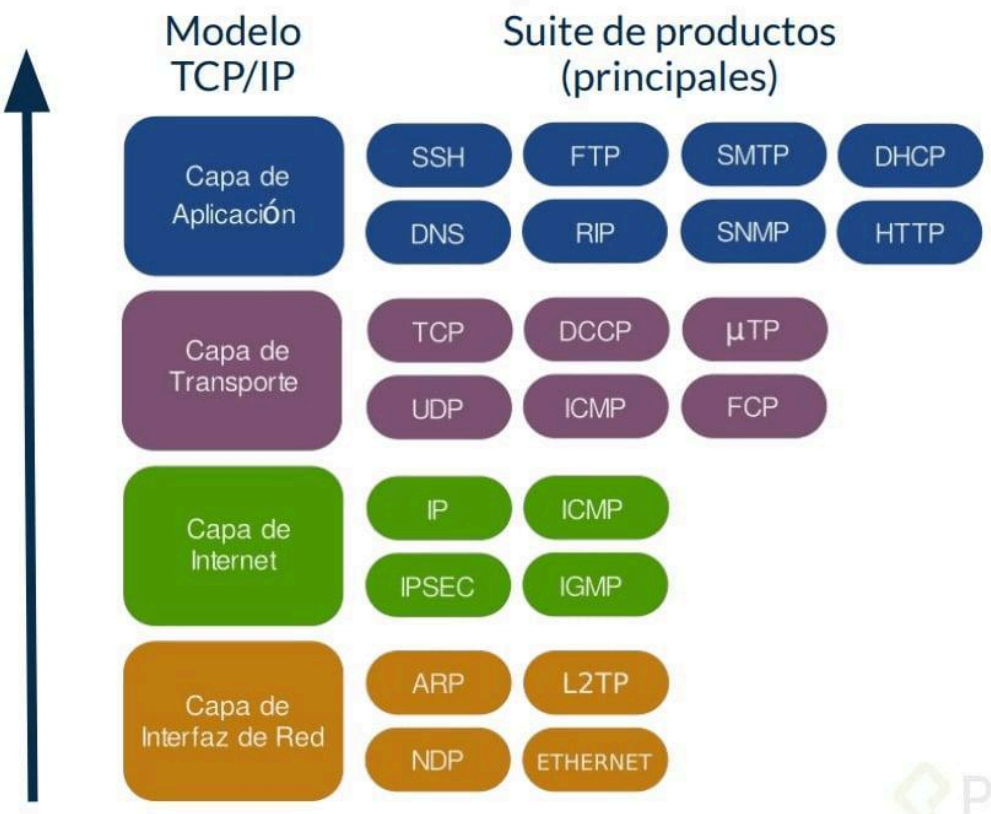
● 2.2.4. TCP/IP

El **modelo TCP** (Protocolo de control de transmisión) proporciona pautas para la comunicación entre dispositivos y programas, asegurando la interpretación precisa de los mensajes. Ha existido por más de 30 años.
Adicionalmente, consta de cuatro niveles: interfaz de red, capa de internet, capa de transporte y capa de aplicación, cada uno desempeñando una función específica para una comunicación adecuada. Ahora examinaremos las cuatro capas desde la base.

¿Cómo funciona el modelo TCP/IP?

Este modelo cuenta con 4 capas. Te las enumeramos de abajo hacia arriba:

- Capa 1 (Interfaz de red)
- Capa 2 (internet)
- Capa 3 (Transporte)
- Capa 4 (Aplicación)



El modelo TCP/IP consta de cuatro capas para redes de comunicación. Cada capa describe un conjunto de directrices generales.

🟢Capa 1: interfaz de red

La primera capa es el nivel más bajo de comunicación compuesto por pulsaciones eléctricas o transporte físico. Este protocolo no establece comunicación entre dos puntos, solo transmite en el medio físico.
Tiene protocolos como:

1. **ARP:** Address Resolution Protocol. Encuentra la MAC de las IP.
2. **Ethernet:** tecnología tradicional para conectar dispositivos en una red LAN.
3. **NDP:** Neighbor Discovery Protocol. Resolución de direcciones y permite que un dispositivo se integre al entorno local (una red física).
4. **L2TP:** Layer 2 Tunneling Protocol. Corrige deficiencias de otros protocolos.

🟢Capa 2: internet

Se encarga de comunicar dos hosts para transmitir los datos y establecer la ruta para que la información llegue al destino deseado.
Tiene protocolos como:

1. **IP:** Internet Protocol. Para identificación y comunicación.
2. **ICMP:** Internet Control Message Protocol.
3. **IPSEC:** Internet Protocol security. Son varios protocolos que autentican la IP y cifran los paquetes.
4. **IGMP:** Internet Group Management Protocol. Multidifusión.

🟢Capa 3: transporte

Se encarga de transferir archivos entre los dos hosts. Para esto, hay varios protocolos, como el protocolo “TCP”, que prioriza la fiabilidad de los datos transmitidos (sin pérdida de información en el camino).
Por otro lado, el protocolo “UDP” se enfoca en la velocidad de transmisión de datos en lugar de la fiabilidad (prefiere enviar los datos lo más rápido posible, incluso si se pierde algo de información en el camino).
Sus protocolos principales son: TCP y UDP.

- **TCP:** Transmission Control Protocol. Controla la transmisión de la información.
- **DCCP:** Datagram Congestion Control Protocol. Transporte de mensajes.
- **MTP:** Micro Transport Protocol. Conexiones peer to peer.
- **UDP:** User Datagram Protocol. Transporte mínimo de mensajes.
- **ICMP:** Internet Control Message Protocol. Mensajes de error e información operativa.
- **FCP:** Fibre Channel protocol. Carga el S.O y verificación.

🟢Capa 4: aplicación

Ofrece a las aplicaciones, tanto de usuario como no, la capacidad de acceder a los servicios de las demás capas y define los protocolos utilizados para el intercambio de datos, como el correo electrónico (POP y SMTP), gestores de bases de datos y protocolos de transferencia de archivos (FTP).
Tiene protocolos como:

- **FTP:** File Transfer Protocol. Transferencia de archivos.
- **SSH:** Secure Shell. Para conexiones seguras.
- **SMTP:** Simple Mail Transfer Protocol. Asigna dinámicamente la IP y otros parámetros.
- **DHCP:** Dynamic Host Configuration Protocol. Establece una conexión y verifica que todo se haya realizado correctamente.
- **DNS:** Domain Name System. Asigna nombres a las IP de los dominios y se encarga de su traducción.
- **RIP:** Routing Information Protocol.
- **SNMP:** Simple Network Management Protocol.
- **HTTP:** Hypertext Transfer Protocol.

Diferencias entre TCP y UDP

TCP:

- Es confiable.
- Está orientado a la conexión.
- Se espera que exista una conexión constante entre 2 computadoras.
- Prioriza la confiabilidad sobre la rapidez.

Un ejemplo claro es la transferencia de archivos.

UDP:

- No es confiable.
- No está orientado a la conexión.
- No se preocupa si los datos llegan o no.

Un ejemplo que ilustra la diferencia entre TCP y UDP es el siguiente: TCP es utilizado en el correo electrónico, donde es necesario que el mensaje completo llegue; en cambio, UDP se utiliza en juegos en línea, donde la información debe llegar lo más rápido posible y no importa si se pierde parte de los datos.

Diferencias entre modelo OSI y modelo TCP/IP

A diferencia del Modelo OSI, el Modelo TCP/IP no se divide en capas separadas y definidas con la misma rigurosidad. En lugar de eso, consta de cuatro capas interconectadas que abordan diferentes aspectos de la comunicación en redes.
Si bien el Modelo OSI puede considerarse desactualizado en términos de implementación práctica, sigue siendo una herramienta valiosa para comprender los principios fundamentales de la comunicación en redes.

● 2.2.5. Routing

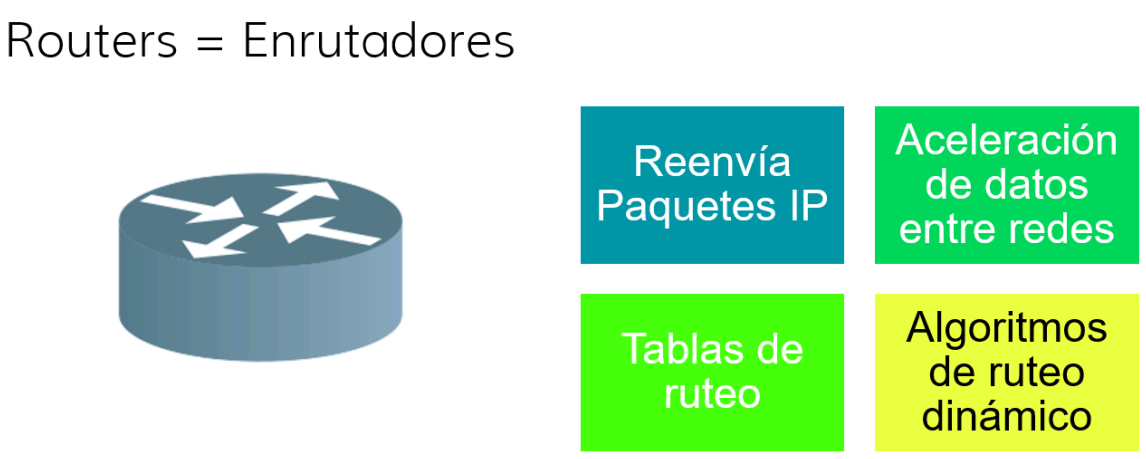
El enrutamiento (routing) es un concepto fundamental en redes de computadoras que se refiere al proceso de determinar la mejor ruta para que los datos viajen desde el origen hasta el destino a través de una red. El enrutamiento es esencial para garantizar que los datos se entreguen de manera eficiente y efectiva a través de redes complejas.

Definición: Es el proceso de selección de la ruta más adecuada para el envío de datos desde un dispositivo de origen a un dispositivo de destino a través de una red.

Componentes Principales:

Routers: Dispositivos dedicados que toman decisiones de enrutamiento basadas en la información de las tablas de enrutamiento.

Tablas de Enrutamiento: Almacenan información sobre la topología de la red y las rutas disponibles.

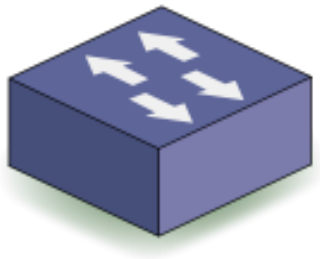


● 2.2.6. Switching

Es el proceso de reenvío de paquetes de datos dentro de una red local, basado en las direcciones MAC (Media Access Control) de los dispositivos conectados.

Funcionamiento Básico:
Los switches determinan el destino de un paquete de datos examinando la dirección MAC de destino en la trama Ethernet.

Utilizan una tabla de direcciones MAC (tabla CAM - Content Addressable Memory) para asociar direcciones MAC con los puertos del switch.



● 2.2.7. Load Balancing

El balanceo de carga (Load Balancing) es una técnica que distribuye el tráfico de red o las solicitudes de servicio entre múltiples servidores para garantizar un uso eficiente de los recursos, mejorar la velocidad de respuesta y garantizar la disponibilidad y confiabilidad del sistema. Esta práctica es comúnmente utilizada en entornos de servidores web, aplicaciones y redes para optimizar el rendimiento y prevenir la congestión en servidores individuales.

Objetivos:

- **Mejorar Rendimiento:** Distribuir la carga entre múltiples servidores para evitar cuellos de botella y mejorar el tiempo de respuesta.
- **Garantizar Disponibilidad:** Asegurar la disponibilidad del servicio al distribuir las solicitudes entre servidores saludables.
- **Escalabilidad:** Facilitar la adición de nuevos servidores para manejar un aumento en la carga.

Métodos de Balanceo de Carga:

- **Round Robin:** Asigna cada solicitud al siguiente servidor en la lista, de manera circular.
- **Balanceo Ponderado:** Asigna pesos a los servidores según su capacidad, distribuyendo más tráfico a servidores más poderosos.
- **Menos Conexiones Actuales:** Dirige el tráfico al servidor con menos conexiones activas en ese momento.

- **Algoritmos de Balanceo Dinámico:** Ajustan la distribución basándose en la carga actual de los servidores.

2.2.8. Firewalls

Los firewalls son dispositivos o programas diseñados para proteger una red o sistema informático al monitorear y controlar el tráfico de red basándose en un conjunto de reglas de seguridad predefinidas. Su objetivo principal es prevenir o limitar el acceso no autorizado y proteger contra amenazas potenciales, como ataques maliciosos, malware y violaciones de seguridad.

Definición

Firewall: Un dispositivo o programa diseñado para filtrar y controlar el tráfico de red según reglas de seguridad definidas, con el objetivo de proteger una red o sistema informático.

Funciones Principales

- **Filtrado de Paquetes:** Examina y decide si permitir o bloquear paquetes de datos basándose en reglas predefinidas.
- **Estado de la Conexión:** Realiza un seguimiento del estado de las conexiones para permitir o bloquear el tráfico en función de la información de la sesión.
- **Inspección de Contenido:** Examina el contenido de los paquetes para detectar patrones maliciosos o firmas de malware.
- **Proxy y Traducción de Direcciones de Red (NAT):** Actúa como intermediario entre los usuarios y los servicios en línea, ocultando las direcciones IP internas.

Tipos de Firewalls

- **Firewalls de Hardware:** Dispositivos físicos dedicados para la protección de red.
- **Firewalls de Software:** Aplicaciones o programas instalados en sistemas operativos para proteger un host o una red.
- **Firewalls de Próxima Generación (NGFW):** Combina funciones de filtrado de paquetes con inspección profunda de contenido y otros mecanismos avanzados.

Ubicaciones Comunes

- **Firewalls de Borde:** Colocados en el perímetro de la red para protegerla contra amenazas externas.

- **Firewalls Internos:** Utilizados dentro de la red para segmentar y proteger secciones específicas.

- **Firewalls Personales:** Software instalado en dispositivos individuales para protegerlos contra amenazas externas.

Reglas de Firewall

- **Permitir o Denegar:** Define qué tipos de tráfico se permiten o se bloquean.
- **Puertos y Protocolos:** Especifica los puertos y protocolos permitidos para el tráfico.
- **Direcciones IP:** Controla el acceso basándose en direcciones IP de origen y destino.

Tipos de Reglas

- **Reglas de Entrada:** Controlan el tráfico que ingresa a la red.
- **Reglas de Salida:** Controlan el tráfico que sale de la red.

Firewalls de Estado y Sin Estado

- **Firewalls de Estado:** Hacen un seguimiento del estado de las conexiones y toman decisiones basándose en el contexto de la sesión.

- **Firewalls Sin Estado:** Evalúan cada paquete individualmente sin considerar el estado de la conexión.

Inspección Profunda de Paquetes (DPI)

- Analiza el contenido de los paquetes para identificar y bloquear amenazas específicas.

2.2.9. VPN

Una Red Privada Virtual (VPN) es una tecnología que permite establecer una conexión segura y cifrada entre dispositivos a través de una red pública, como Internet. Su principal objetivo es proporcionar privacidad y seguridad a la comunicación en línea, permitiendo a los usuarios acceder a recursos de red de forma remota como si estuvieran físicamente conectados a la misma red local.

Protocolos VPN

- **IPsec (Protocolo de Seguridad de Internet):** Comúnmente utilizado para implementar VPNs, proporciona un conjunto de protocolos para la seguridad de la comunicación.
- **OpenVPN:** Un protocolo de código abierto que utiliza tecnologías SSL/TLS para la creación de túneles seguros.
- **PPTP (Protocolo de túneles punto a punto):** Menos común hoy en día debido a vulnerabilidades conocidas, pero aún en uso en algunos casos.
- **L2TP/IPsec (Layer 2 Tunneling Protocol):** Combina las ventajas de L2TP y IPsec para proporcionar una conexión segura.

Aplicaciones Comunes de VPN

- **Acceso Remoto a la Empresa:** Permite a empleados acceder a recursos de la empresa desde ubicaciones remotas de manera segura.
- **Conexiones Interempresariales:** Facilita la conexión segura entre diferentes sucursales de una empresa.
- **Acceso Seguro a Internet:** Protege la conexión a Internet en redes públicas, como en cafeterías o aeropuertos.

Ventajas de las VPN

- **Privacidad:** Cifra la comunicación para proteger la privacidad y la confidencialidad.
- **Seguridad:** Proporciona una capa adicional de seguridad al transmitir datos a través de redes no confiables.
- **Acceso Remoto:** Facilita el acceso seguro a recursos de red desde cualquier ubicación.
- **Bypass de Restricciones Geográficas:** Permite el acceso a servicios en línea que pueden estar restringidos por ubicación geográfica.

2.2.10. Troubleshooting y Monitoreo

Troubleshooting

El Troubleshooting en redes se refiere al proceso de identificar, diagnosticar y resolver problemas o fallas en una red de computadoras. Este proceso es esencial para mantener un rendimiento óptimo y una disponibilidad continua de los servicios de red.

Pero... ¿Qué deberíamos chequear al hacer un Troubleshooting o solución de problemas en una red?

Identificación de Problemas

- **Recolección de Información:** Obtener detalles sobre el problema, como síntomas, mensajes de error y cambios recientes.
- **Documentación de la Red:** Contar con documentación actualizada de la configuración de red, topología y políticas facilita la identificación de problemas.

Análisis de Topología

- **Mapa de Red:** Tener un mapa de la topología de la red ayuda a visualizar cómo están interconectados los dispositivos.
- **Rastreo de Rutas:** Utilizar herramientas de rastreo de rutas (traceroute) para identificar la ruta que toman los paquetes a través de la red.

Herramientas de Diagnóstico

Ping: Verifica la conectividad básica con un host.

```
ping <dirección_ip_o_nombre_de_host>
```

Traceroute (Windows) / Tracepath (Linux y macOS): Muestra la ruta que toma un paquete para llegar a su destino.

```
tracert <dirección_ip_o_nombre_de_host>  en Windows
```

```
tracepath <dirección_ip_o_nombre_de_host>  en Linux y macOS
```

Ipconfig (Windows) / ifconfig (Linux) / ip (Linux y macOS): Muestra la configuración de red de la interfaz de red.

```
ipconfig                (en Windows)
ifconfig                (en Linux)
ip a                    (en Linux y macOS)
```

Dig (Linux) / nslookup (Windows) / host (Linux y macOS): Realiza consultas de DNS para obtener información sobre un dominio o dirección IP.

```
dig <nombre_de_host_o_dominio>    (en Linux)
nslookup <nombre_de_host_o_dominio> (en Windows)
host <nombre_de_host_o_dominio>    (en Linux y macOS)
```

Netstat: Muestra conexiones de red, tablas de enrutamiento y estadísticas de la interfaz.

```
netstat -an
```

Nmap (Requiere instalación): Escanea puertos en un host para identificar servicios y determinar su estado.

```
nmap <dirección_ip_o_nombre_de_host>
```

Quick Learning NMAP

https://www.youtube.com/watch?v=qRE0AmeYFi4
Duración: 16:50

Wireshark (Requiere instalación): Permite analizar el tráfico de red y capturar paquetes para diagnosticar problemas.

Quick Learning Wireshark

https://www.youtube.com/watch?v=L8kSqSSZ0Ug&t=24s
Duración: 22:49

Iperf (requiere instalación): Mide el rendimiento de la red mediante la transferencia de datos entre dos sistemas.

```
iperf -s          (en el servidor)
iperf -c <ip>     (en el cliente)
```

NSLookup: realiza consultas DNS (Sistema de Nombres de Dominio). Su propósito principal es obtener información relacionada con la resolución de nombres de dominio, como la dirección IP asociada a un nombre de host o viceversa.

1. **Obtener la dirección IP de un nombre de host:**

```
nslookup <nombre_de_host>
```

```
nslookup www.ejemplo.com
```

2. **Obtener el nombre de host asociado a una dirección IP:**

```
nslookup <dirección_ip>
```

```
nslookup 8.8.8.8
```

3. **Cambiar el servidor DNS utilizado:**

```
nslookup > server <dirección_ip_del_servidor_dns>
```
Esto te permite cambiar el servidor DNS antes de realizar consultas.

Quick Learning NSLookup

https://www.youtube.com/watch?v=JdWgcBrpkE8&t=10s
Duración: 3:45

Logs y Registros

- **Registro de Eventos:** Revisar registros y eventos en routers, switches y servidores para identificar posibles problemas.
- **Logs del Sistema:** Analizar los logs del sistema en servidores y dispositivos de red para detectar eventos anómalos.

Dividir y Conquistar:

- **Segmentación de la Red:** Dividir la red en segmentos más pequeños para identificar en qué parte se encuentra el problema.
- **Prueba por Capas:** Diagnosticar problemas de red siguiendo el modelo OSI y probando cada capa por separado.

Actualizaciones y Cambios

- **Verificación de Cambios Recientes:** Identificar cualquier cambio en la configuración o actualización de software que pueda haber causado el problema.
- **Backups:** Mantener copias de seguridad de configuraciones y registros antes de realizar cambios.

Colaboración y Documentación

- **Colaboración en Equipo:** Trabajar con otros miembros del equipo de soporte para resolver problemas de manera eficiente.
- **Documentación de Soluciones:** Registrar soluciones y procedimientos para problemas recurrentes para referencia futura.

Efectuar Mantenimiento Preventivo

- **Actualizaciones y Parches:** Mantener actualizado el software y firmware para evitar problemas de seguridad y rendimiento.
- **Auditorías Regulares:** Realizar auditorías periódicas de seguridad y rendimiento en la red.

Análisis de Rendimiento

- **Herramientas de Monitoreo de Rendimiento:** Utilizar herramientas para monitorear el rendimiento de la red y detectar posibles cuellos de botella.

Monitoreo de Redes

El monitoreo de redes implica la observación continua y el análisis del tráfico, rendimiento y disponibilidad de una red para garantizar un funcionamiento óptimo y resolver problemas de manera proactiva.

1. **Herramientas de Monitoreo:**

- **SNMP (Simple Network Management Protocol):** Protocolo estándar para monitorear y gestionar dispositivos de red.
- **Paquetes de Monitoreo:** Utilizar herramientas como Nagios, Zabbix, PRTG, entre otras, para monitorear dispositivos y servicios.

🏃 **Quick Learning** Monitoreo con Nagios
https://www.youtube.com/watch?v=bzIzTShR0C4
Duración: 29:00

2. **Métricas de Rendimiento:**

- **Ancho de Banda:** Medir la utilización del ancho de banda para identificar posibles cuellos de botella.
- **Latencia y Pérdida de Paquetes:** Evaluar la calidad de la conexión y la latencia de la red.
- **Uso de Recursos:** Monitorear la carga de CPU, memoria y almacenamiento en dispositivos de red.

3. **Alertas y Notificaciones:**

- **Configuración de Umbrales:** Establecer umbrales para métricas críticas y recibir alertas cuando se superan.
- **Notificaciones en Tiempo Real:** Ser notificado inmediatamente cuando se detecta un problema.

4. **Mapas de Red y Topología:**
- **Visualización de la Topología:** Crear mapas visuales de la topología de la red para entender su estructura.
- **Rastreo de Conexiones:** Seguir conexiones entre dispositivos para identificar patrones de tráfico.

5. **Historial de Datos:**
- **Almacenamiento de Datos:** Mantener un historial de datos para realizar análisis a largo plazo y detectar patrones.
- **Tendencias y Estadísticas:** Identificar tendencias en el rendimiento y el tráfico de la red.

6. **Monitoreo de Seguridad:**
- **Detección de Anomalías:** Utilizar herramientas que puedan detectar comportamientos anómalos que podrían indicar ataques o intrusiones.
- **Registro de Eventos de Seguridad:** Monitorear logs de seguridad para detectar eventos sospechosos.

7. **Gestión de Configuración:**
- **Control de Cambios:** Registrar y monitorear cambios en la configuración de dispositivos de red.
- **Comparación de Configuraciones:** Comparar configuraciones actuales con versiones anteriores para identificar discrepancias.

8. **Informes y Análisis:**
- **Generación de Informes:** Crear informes regulares sobre el rendimiento y la disponibilidad de la red.
- **Análisis Post-incidente:** Analizar eventos y problemas pasados para mejorar la preparación y la respuesta futura.

9. **Monitoreo de Aplicaciones:**
- **Monitoreo de Servicios:** Vigilar la disponibilidad y el rendimiento de servicios y aplicaciones específicos.
- **Transacciones de Aplicaciones:** Analizar el rendimiento de transacciones críticas.

10. **Escalabilidad y Automatización:**
- **Escalabilidad:** Utilizar soluciones que se adapten a entornos en crecimiento sin perder eficiencia.

- **Automatización de Tareas:** Implementar automat

2.2.11. Automatización de Networking

Vamos a enfocarnos en este aspecto más adelante cuando hablemos de la Automatización de la Infraestructura.

Solo para aclarar la idea de automatizar la gestión de redes tiene grandes beneficios, así como también agrega complejidad, dado que se requieren conocimientos de scripting y programación y el uso de algunas herramientas adicionales.

Herramientas y Tecnologías de Automatización

- **Ansible:** Un popular framework de automatización que utiliza playbooks escritos en YAML para describir configuraciones y tareas.
- **Chef y Puppet:** Herramientas de automatización de configuración que permiten definir y gestionar la infraestructura como código.
- **SaltStack:** Plataforma de automatización que utiliza un enfoque de ejecución remota para la gestión de la configuración.
- **Python y Scripts:** Muchos profesionales de networking utilizan scripts personalizados en Python u otros lenguajes para automatizar tareas específicas.

2.2.12. Arquitectura de Red en la Nube

Podremos encontrar diferentes arquitecturas de Red en la nube, sobre todo si cambiamos de proveedor. No vamos a entrar en detalle en este aspecto en este momento, pero vamos a comentar algunos de los servicios que brindan proveedores tales como AWS, Amazon o GCP.

Amazon Virtual Private Cloud (VPC)

VPC es un servicio que permite lanzar recursos de AWS en una red virtual aislada de forma lógica que usted defina. Puede controlar todos los aspectos del entorno de red virtual, como la selección de su propio rango de direcciones IP, la creación de subredes y la configuración de tablas de enrutamiento y gateways de red. Puede utilizar tanto IPv4 como IPv6 para la mayoría de los recursos de la VPC, lo que ayuda a garantizar el acceso seguro y fácil a los recursos y las aplicaciones.

Como uno de los servicios esenciales de AWS, Amazon VPC facilita la personalización de la configuración de red de la VPC. Puede crear una subred con acceso público para los servidores web que tengan acceso a Internet. También permite colocar los sistemas backend, como los servidores de aplicaciones o las bases de datos, en una subred con acceso privado sin acceso a Internet. Amazon VPC le permite utilizar varias capas de seguridad, incluidos los grupos de seguridad y las listas de control de acceso a la red, para ayudar a controlar el acceso a las instancias de Amazon Elastic Compute Cloud (Amazon EC2) en cada subred.

Azure Virtual Network (VNET)

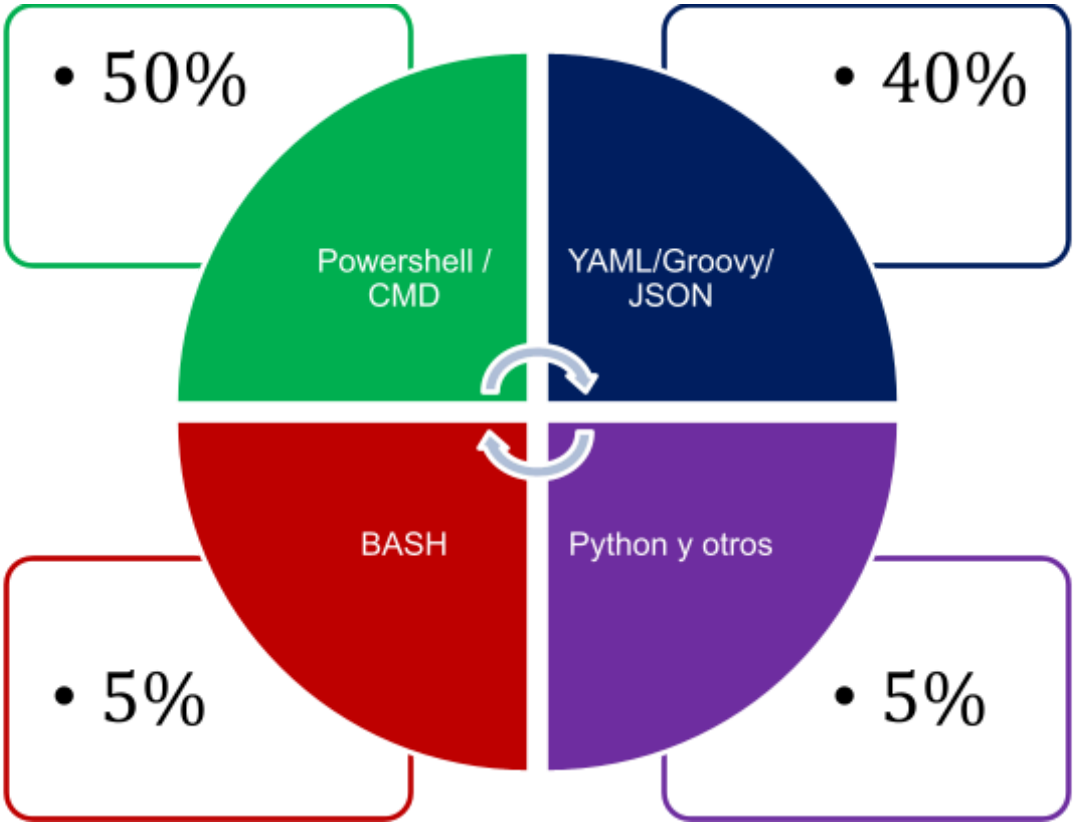
VNET es un servicio que proporciona el bloque de compilación fundamental para su red privada en Azure. Una instancia del servicio (una red virtual) permite que muchos tipos de recursos de Azure se comuniquen de forma segura entre sí, Internet y redes locales. Estos recursos de Azure incluyen máquinas virtuales (VM).

Una red virtual es similar a una red tradicional con la que trabajaría en su propio centro de datos. Pero aporta ventajas adicionales de la infraestructura Azure, como escala, disponibilidad y aislamiento.

2.3 Fundamentos de Scripting

Un ingeniero de DevOps es un generalista de TI que debe tener un amplio conocimiento tanto del desarrollo como de las operaciones, lo que incluye programación, gestión de infraestructuras, administración de sistemas y cadenas de herramientas de DevOps. Los

ingenieros de DevOps también deben tener habilidades interpersonales, ya que trabajan con los distintos grupos aislados de la empresa para crear un entorno más colaborativo. Los ingenieros de DevOps necesitan sólidos conocimientos de la arquitectura común del sistema, el aprovisionamiento y la administración, pero también deben tener experiencia con el conjunto de herramientas y prácticas tradicionales de los desarrolladores, como el control del código fuente, la entrega y recepción de revisiones de código, la escritura de pruebas unitarias y la familiaridad con los principios de la metodología ágil.



• 2.3.1. CMD (Windows)

CMD, abreviatura de Command Prompt, es una interfaz de línea de comandos (CLI) en sistemas operativos basados en DOS y en versiones de Windows. Ofrece una forma de interactuar con el sistema operativo utilizando comandos de texto en lugar de una interfaz gráfica.

Características Principales

1. **Interfaz de Línea de Comandos (CLI):** CMD proporciona una interfaz basada en texto donde los usuarios ingresan comandos para realizar tareas específicas. Los comandos se escriben en la ventana del símbolo del sistema.

2. **Comandos de DOS:** CMD incluye una serie de comandos propios de DOS, como **dir** para listar archivos y directorios, **copy** para copiar archivos, **del** para eliminar archivos, entre otros.

3. **Ruta de Comandos:** CMD utiliza una variable de entorno llamada "PATH" para buscar los ejecutables de los comandos. La ruta especificada determina dónde CMD buscará los programas para ejecutar.

4. **Scripting y Archivos por Lotes:** Puedes crear scripts o archivos por lotes que contienen una secuencia de comandos. Estos archivos pueden ejecutarse en CMD para automatizar tareas repetitivas.

5. **Acceso a Herramientas del Sistema:** CMD permite el acceso a diversas herramientas y utilidades del sistema operativo, proporcionando a los usuarios un mayor control sobre el sistema.

Ejemplos de Comandos

- **dir**: Muestra el contenido del directorio actual.
- **cd**: Cambia el directorio actual.
- **copy**: Copia archivos de un lugar a otro.
- **del**: Elimina archivos.
- **type**: Muestra el contenido de un archivo de texto.
- **cls**: Limpia la pantalla de la ventana del símbolo del sistema.

Uso Básico

Abrir CMD: Puedes abrir la ventana del símbolo del sistema escribiendo "cmd" en la barra de búsqueda de Windows y presionando Enter. Recuerda que si quieres usar el modo elevado de privilegios de administrador deberás ejecutar en Windows seleccionando dicha opción o presionando CTRL+Shift+Enter. En ambos casos se abrirá UAC y solicitará el permiso correspondiente.

Navegar por Directorios:

```
cd \          # Cambia al directorio raíz.
cd C:\Users # Cambia al directorio "Users" en la unidad C.
```

Listar Archivos y Directorios:

```
dir           # Muestra el contenido del directorio actual.
```

```
dir /w      # Muestra una lista más corta y ancha.
```

Ejecutar Programas:

```
notepad     # Abre el Bloc de notas.
calc        # Abre la calculadora.
```

Crear y Ejecutar un Archivo por Lotes:

```
echo echo Hola, Mundo! > HolaMundo.bat # Crea un archivo por lotes.
HolaMundo.bat                          # Ejecuta el archivo por lotes.
```

● 2.3.2. Powershell

PowerShell es un entorno de línea de comandos y scripting desarrollado por Microsoft para la administración y automatización de tareas en sistemas Windows. También se conoce como "Windows PowerShell" o "PowerShell Core" (la versión multiplataforma basada en .NET Core).

Origen y Propósito de PowerShell

- Fue lanzado por primera vez en 2006 y se basa en el marco .NET. Su objetivo es proporcionar una interfaz unificada y consistente para la administración del sistema y la automatización de tareas en entornos Windows.

- PowerShell reemplaza y mejora el antiguo cmd.exe, ofreciendo una sintaxis más rica y capacidades de scripting avanzadas.

Importancia en la Administración y Automatización

- PowerShell es esencial para la administración eficiente de sistemas Windows y entornos de servidores.

- Facilita la automatización de tareas rutinarias, la gestión de configuraciones, la supervisión de servicios y más.

- Se utiliza ampliamente en entornos empresariales para la administración de servidores, la configuración de Active Directory, la gestión de Exchange, entre otros.

Principales Características

- **Interfaz de Línea de Comandos (CLI) y Scripting:** PowerShell proporciona una interfaz de línea de comandos para la ejecución de comandos inmediatos, así como un entorno de scripting para la creación de scripts.

- **Cmdlets:** Los cmdlets son pequeñas funciones especializadas que realizan operaciones específicas. Siguen una convención de nomenclatura verbo-sustantivo (por ejemplo, **Get-Process**, **Stop-Service**).

- **Pipes y Redirecciones de Salida:** Permite encadenar comandos utilizando tuberías (pipes) y redireccionar la salida de un comando a otro o a un archivo.

Ejemplo de Comandos Básicos

```
# Mostrar directorio actual
Get-Location

# Listar procesos en ejecución
Get-Process

# Detener un servicio
Stop-Service -Name "NombreDelServicio"
```

Interfaz de Línea de Comandos (CLI) y Scripting

- **Interactividad:** PowerShell permite ejecutar comandos de forma interactiva en la línea de comandos, proporcionando una experiencia similar a la de la consola tradicional. Los comandos se ejecutan inmediatamente, y los resultados se muestran en la consola.

- **Ambiente de Scripting:** Además de la interfaz interactiva, PowerShell es un entorno de scripting poderoso. Los usuarios pueden escribir scripts para realizar tareas más complejas y automatizar flujos de trabajo.

Cmdlets

- **Definición:** Los cmdlets (command-lets) son pequeñas funciones especializadas que realizan operaciones específicas en el sistema. Se caracterizan por seguir una convención de nomenclatura verbo-sustantivo.

Ejemplo: **Get-Process** (obtiene información sobre los procesos en ejecución).

- **Funcionalidad Abundante:** PowerShell incluye una amplia variedad de cmdlets incorporados para tareas comunes como administrar servicios, archivos, procesos, usuarios, etc.

Pipes y Redirecciones de Salida

- **Pipes:** Permite encadenar la salida de un cmdlet como entrada para otro, facilitando la realización de operaciones complejas en una sola línea.

Ejemplo: **Get-Process | Where-Object { $_.WorkingSet -gt 100MB } | Stop-Process**

- **Redirecciones de Salida:** Se puede redirigir la salida de un cmdlet a un archivo o a otro cmdlet.

Ejemplo: **Get-Process | Out-File -FilePath "procesos.txt"**

Documentación Oficial👉🏽Documentación de PowerShell

🏃🏽‍♀️Quick Learning Powershell

https://www.youtube.com/watch?v=YwGlXXqLDkM&t=12s
Duración: 25 minutos

Pasar Parámetros / Argumentos

En PowerShell, puedes pasar parámetros a un script utilizando la declaración param o simplemente definiendo las variables al principio del script. Aquí hay ejemplos de ambas formas:

- **Usando la Declaración param:**

```
param (
    [string]$nombre,
    [int]$edad
)

Write-Host "Nombre: $nombre"
Write-Host "Edad: $edad años"
```

En este ejemplo, el script espera dos parámetros: $nombre (una cadena) y $edad (un entero). Puedes ejecutar el script y pasar los parámetros de la siguiente manera:

```
.\TuScript.ps1 -nombre "John" -edad 30
```

- **Definiendo Variables al Principio del Script:**

```
# Definir variables al principio del script
$nombre = $args[0]
```

```
$edad = $args[1]

Write-Host "Nombre: $nombre"
Write-Host "Edad: $edad años"
```

En este caso, $args es una variable que contiene todos los argumentos pasados al script. Puedes ejecutar el script y pasar los parámetros así:

```
.\TuScript.ps1 "John" 30
```

Ambas formas son válidas, pero el uso de param es más explícito y permite especificar tipos de datos para los parámetros. El segundo método con $args es más sencillo, pero puede ser menos legible y puede llevar a errores si no se maneja correctamente.

Elige la opción que se ajuste mejor a tus necesidades y preferencias. Ten en cuenta que al usar param, los nombres de los parámetros están precedidos por un guion, mientras que con $args simplemente proporcionan los valores en el orden correcto.

Estructuras de control

🔹 Condicionales - if, else, elseif

Las estructuras condicionales permiten ejecutar bloques de código basándose en evaluaciones de condiciones.

```
$numero = 10

if ($numero -eq 10) {
    Write-Host "El número es igual a 10."
} elseif ($numero -lt 10) {
    Write-Host "El número es menor que 10."
} else {
    Write-Host "El número es mayor que 10."
}
```

🔹 Switch

La estructura switch permite comparar una expresión con varios valores posibles y ejecutar bloques de código según la coincidencia.

```
$opcion = "B"
```

```
switch ($opcion) {
    "A" { Write-Host "Opción A seleccionada." }
    "B" { Write-Host "Opción B seleccionada." }
    "C" { Write-Host "Opción C seleccionada." }
    default { Write-Host "Opción no reconocida." }
}
```

- **Bucles**

Foreach

El bucle foreach permite recorrer elementos de una colección, como una matriz o una lista.

```
$nombres = "Juan", "María", "Carlos"

foreach ($nombre in $nombres) {
    Write-Host "Hola, $nombre."
}
```

For:

El bucle for permite ejecutar un bloque de código un número específico de veces.

```
for ($i = 1; $i -le 5; $i++) {
    Write-Host "Iteración $i."
}
```

While

El bucle while ejecuta un bloque de código mientras una condición sea verdadera.

```
$contador = 1

while ($contador -le 3) {
    Write-Host "Iteración $contador."
    $contador++
}
```

Do-while

El bucle do-while es similar a while, pero garantiza que el bloque de código se ejecute al menos una vez antes de evaluar la condición.

```
$contador = 1
```

```
do {
    Write-Host "Iteración $contador."
    $contador++
} while ($contador -le 3)
```

• 2.3.3. Lenguajes para intercambio de Objetos

JSON – JavaScript Object Notation

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y de fácil lectura que se utiliza para transmitir y almacenar datos estructurados. Aunque su nombre incluye "JavaScript", JSON es un formato de datos independiente del lenguaje y se utiliza ampliamente en diversos entornos y tecnologías.

1. **Sintaxis**

   JSON utiliza una sintaxis sencilla y fácil de entender, que se compone principalmente de pares clave-valor. Los datos se representan como objetos (conjuntos no ordenados de pares clave-valor), matrices (listas ordenadas de valores) y valores escalares (cadenas, números, booleanos, nulos).

   Ejemplo de un objeto JSON:

   ```
   {
     "nombre": "Juan",
     "edad": 25,
     "ciudad": "Ejemploville",
     "activo": true,
     "hobbies": ["lectura", "cine", "deporte"]
   }
   ```

2. **Tipos de Datos**

   JSON admite los siguientes tipos de datos:

   Objeto: Un conjunto no ordenado de pares clave-valor, delimitado por llaves {}.
   Matriz: Una lista ordenada de valores, delimitada por corchetes [].
   Cadena: Una secuencia de caracteres entre comillas (").
   Número: Puede ser entero o de punto flotante.
   Booleano: `true` o `false`.
   Nulo: `null`.

3. **Uso en la Web**

   JSON es ampliamente utilizado en el desarrollo web para intercambiar datos entre el servidor y el cliente. Se utiliza comúnmente en combinación con tecnologías como AJAX (Asynchronous JavaScript and XML) para realizar solicitudes asíncronas a servidores y recibir datos en formato JSON.

4. **Formato Ligero**

   Una de las ventajas clave de JSON es su formato ligero. La información se puede representar de manera compacta, lo que facilita su transmisión a través de redes y su almacenamiento eficiente.

5. **Formato de Configuración**

   Debido a su simplicidad y legibilidad, JSON se utiliza a menudo como formato de configuración para aplicaciones y servicios. Muchas aplicaciones y servicios web utilizan archivos JSON para almacenar configuraciones y preferencias.

6. **Herramientas y Lenguajes**

   JSON es compatible con una amplia variedad de lenguajes de programación. La mayoría de los lenguajes de programación modernos tienen bibliotecas o funciones integradas para trabajar con JSON.

7. **Uso en API REST**

   En el contexto de servicios web, las API REST a menudo transmiten datos en formato JSON. Una solicitud a una API REST podría enviar datos al servidor en formato JSON, y el servidor podría responder con datos también en formato JSON.

8. **Validación de Esquema**

   JSON Schema es un estándar que define la estructura y la validación de los datos JSON. Proporciona un conjunto de reglas para describir la estructura esperada de los datos y garantizar su validez.

YAML = Yet Another Markup Language

Acrónimo recursivo de "YAML Ain't Markup Language" o "Yet Another Markup Language" es un formato de serialización de datos legible por humanos y fácil de escribir. A diferencia de JSON, que utiliza símbolos como **`{}`** y **`[]`**, YAML se basa en la identación y utiliza espacios para estructurar los datos. YAML es comúnmente utilizado para la configuración de aplicaciones, archivos de definición de tareas, y en general, para representar datos estructurados de manera clara y legible.

1. Sintaxis Basada en Identación

La estructura en YAML se define mediante la indentación en lugar de utilizar símbolos como corchetes o llaves. Esto facilita la lectura y escritura de documentos YAML.

Ejemplo de un objeto YAML:

```yaml
nombre: Juan
edad: 25
ciudad: Ejemploville
activo: true
hobbies:
  - lectura
  - cine
  - deporte
```

2. Tipos de Datos

YAML admite tipos de datos como objetos, matrices, cadenas, números, booleanos y nulos, similar a JSON.

3. Facilidad de Lectura y Escritura

La sintaxis de YAML es diseñada para ser fácilmente legible por humanos, lo que facilita su uso en archivos de configuración y documentos donde la legibilidad es crucial.

4. Comentarios

YAML permite comentarios para agregar anotaciones o explicaciones en el documento. Los comentarios comienzan con el carácter **#**.

```yaml
# Esto es un comentario en YAML
```

5. Herencia y Anclaje

YAML permite la herencia de estructuras y el anclaje de nodos, lo que puede ser útil para reutilizar partes de la estructura en diferentes lugares del documento YAML.

6. Uso en Configuración

YAML es comúnmente utilizado en archivos de configuración de aplicaciones y servicios. Muchos sistemas de configuración y orquestación, como Ansible y Kubernetes, utilizan YAML para describir la configuración de tareas y recursos.

7. Documentos YAML Multilínea

YAML facilita la escritura de cadenas multilínea utilizando **|** o **>** para conservar saltos de línea y espaciado.

```yaml
descripcion: |
Este es un ejemplo |
de una cadena multilínea |
```

en YAML.

8. Compatibilidad con JSON

Debido a sus similitudes, YAML y JSON son a menudo interoperables. Puedes convertir fácilmente un documento YAML a JSON y viceversa.

9. YAML en Lenguajes de Programación

Muchos lenguajes de programación modernos cuentan con bibliotecas o módulos para trabajar con datos en formato YAML. Por ejemplo, en Python, puedes usar la biblioteca PyYAML.

● 2.3.4. Groovy

Groovy es un lenguaje de programación de alto nivel, dinámico y orientado a objetos que se ejecuta en la plataforma Java Virtual Machine (JVM). Fue diseñado para ser compatible con Java y aprovecha la infraestructura de la JVM, lo que facilita la interoperabilidad con bibliotecas Java existentes. Groovy combina características de varios lenguajes, incluidos Java, Python y Ruby, proporcionando una sintaxis concisa y expresiva.

1. Sintaxis Concisa

Groovy se diseñó para tener una sintaxis más concisa y fácil de leer en comparación con Java. Esto incluye características como la omisión de puntos y comas, lo que reduce la necesidad de escribir caracteres adicionales.

¡Ejemplo de un programa “Hola, Mundo!" en Groovy:

```
println "¡Hola, Mundo!"
```

2. Orientado a Objetos

Groovy es un lenguaje totalmente orientado a objetos, donde todo es un objeto. Incluso los tipos primitivos se mapean a clases.

3. Dinamismo

Groovy es un lenguaje dinámico, lo que significa que las variables y tipos de datos pueden ser determinados en tiempo de ejecución. Esto proporciona una mayor flexibilidad y expresividad en el código.

4. **Cierre de Bloques (Closures)**

Groovy incluye cerraduras (closures) que son bloques de código que pueden ser asignados a variables y pasados como argumentos a funciones. Los cierres son una característica poderosa y versátil.

Ejemplo de un cierre en Groovy:

```
def saludar = { nombre ->
   println "¡Hola, $nombre!"
}
saludar("Daniel")
```

5. **Interoperabilidad con Java**

Groovy puede interoperar directamente con código Java. Puedes utilizar bibliotecas Java existentes y también puedes llamar a código Groovy desde Java.

6. **DSL (Domain-Specific Language) Amigable**

La sintaxis de Groovy facilita la creación de DSLs específicos del dominio, lo que permite escribir código que se asemeje al lenguaje del dominio específico que estás abordando.

7. **Manejo Mejorado de Colecciones**

Groovy ofrece métodos mejorados para el manejo de colecciones, lo que facilita las operaciones de filtrado, mapeo y reducción.

Ejemplo de manipulación de colecciones en Groovy:

```
def numeros = [1, 2, 3, 4, 5]
def cuadrados = numeros.collect { it * it }

println cuadrados
```

- ## 2.3.5. BASH

Acrónimo de "Bourne Again SHell," es un intérprete de comandos y un lenguaje de scripting ampliamente utilizado en sistemas operativos basados en Unix y Linux. Es la shell predeterminada en la mayoría de las distribuciones de Linux y también está disponible en sistemas macOS.

Es muy similar a CMD en algunos aspectos y en otros se parece a PowerShell, sin embargo, Bash fue primero, recordemos que PowerShell existe particularmente en

entornos Windows, aunque se puede utilizar también en Linux comúnmente no excede esta plataforma.

Estructuras de Control

Bash admite estructuras de control como `if`, `else`, `for`, `while`, etc.

```
if [ "$edad" -ge 18 ]; then
    echo "Eres mayor de edad."
else
    echo "Eres menor de edad."
fi
```

Redirección y Tuberías

Puedes redirigir la entrada/salida estándar y combinar comandos mediante tuberías.

```
ls > lista_archivos.txt
cat lista_archivos.txt | grep "txt"
```

Expansión de Comandos

Bash realiza la expansión de comandos, permitiendo la ejecución de comandos dentro de una cadena.

```
fecha_actual=$(date)
echo "La fecha actual es: $fecha_actual"
```

Variables de Entorno:

Bash utiliza variables de entorno para almacenar información que puede ser utilizada por los programas en ejecución.

```
export NOMBRE_USUARIO="Dan"
```

Temas Avanzados

- **Funciones en Bash:**

Puedes definir y utilizar funciones para modularizar y reutilizar código.

🔸 **Expresiones Regulares:**
Bash admite el uso de expresiones regulares para buscar y manipular patrones en texto.

🔸 **Gestión de Procesos:**
Puedes manejar procesos en segundo plano, procesos en primer plano, y enviar señales a procesos.

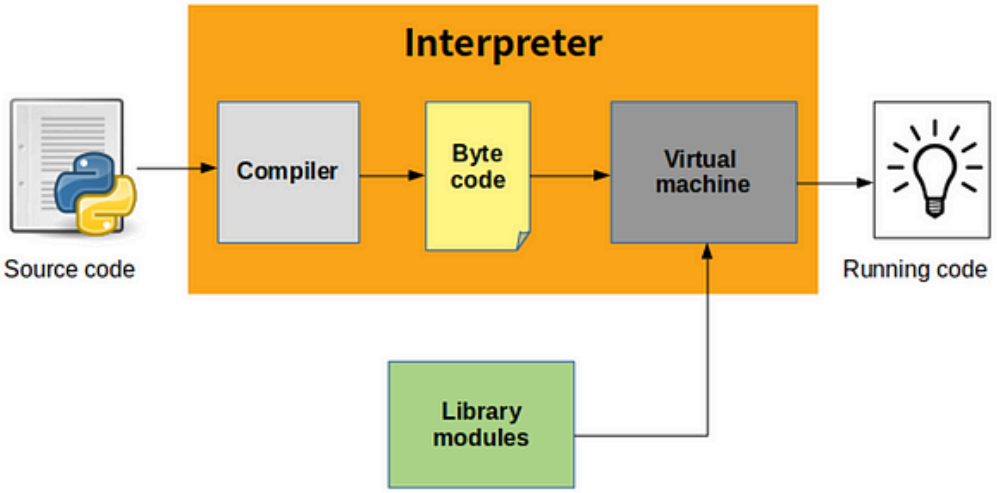
🔸 **Control de Usuarios y Permisos:**
Bash se utiliza comúnmente para la administración de usuarios y permisos en sistemas Unix/Linux.

Comparación entre Bash y CMD

Funcionalidad	CMD (Windows)	Bash (Unix/Linux)
Mostrar Directorio Actual	cd o chdir	pwd
Listar Archivos y Directorios	dir	ls
Cambiar de Directorio	cd	cd
Crear un Directorio	mkdir	mkdir
Copiar un Archivo o Directorio	copy o xcopy	cp
Mover o Renombrar un Archivo o Directorio	move	mv
Eliminar un Archivo	del o erase	rm
Eliminar un Directorio	rmdir	rmdir o rm -r
Ver el Contenido de un Archivo	type o more	cat o less
Crear un Archivo Vacío	type nul > archivo.txt	touch archivo.txt
Conectar a un Servidor Remoto (SSH)	No aplicable	ssh usuario@servidor
Variables de Entorno	%VAR%	$VAR
Ejecutar un Script o Comando	script.bat o comando	./script.sh o comando
Comentarios en Script	REM	#
Mostrar Ayuda	help o /?	man o --help

● 2.3.6. Python

Python es un lenguaje de programación de alto nivel, interpretado y generalista. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. A lo largo de los años, Python se ha convertido en uno de los lenguajes de programación más populares y ampliamente utilizados en la industria y en la comunidad de desarrollo.



1. **Sintaxis Clara y Concisa:**

- Python se distingue por su sintaxis clara y legible, lo que facilita la escritura y lectura del código. Utiliza espacios en blanco (identación) para definir bloques de código en lugar de llaves o palabras clave.

2. **Interpretado e Interpretación Dinámica:**

- Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea por un intérprete en lugar de ser compilado antes de la ejecución. Además, es de tipado dinámico, permitiendo asignar variables sin declarar su tipo.

3. **Multiplataforma:**

- Python es compatible con una amplia gama de plataformas, lo que significa que un programa escrito en Python puede ejecutarse en diversas plataformas sin necesidad de modificaciones.

4. **Amplia Biblioteca Estándar:**

- Python cuenta con una biblioteca estándar muy completa que abarca desde operaciones básicas hasta funcionalidades avanzadas. Esto facilita el desarrollo de aplicaciones sin tener que escribir todo desde cero.

5. **Comunidad Activa y Documentación Extensa:**

- Python tiene una comunidad grande y activa de desarrolladores que contribuyen a su desarrollo y mejora continua. La documentación oficial de Python es extensa y fácil de seguir.

6. **Versatilidad:**

- Python se utiliza en diversos contextos, como desarrollo web, ciencia de datos, inteligencia artificial, automatización, desarrollo de juegos, scripting, y más. Esta versatilidad lo hace atractivo para una amplia audiencia de programadores.

7. **Frameworks y Bibliotecas Populares:**

- Hay muchos frameworks y bibliotecas populares en Python que facilitan el desarrollo en áreas específicas. Algunos ejemplos son Django y Flask para desarrollo web, NumPy y Pandas para ciencia de datos, TensorFlow y PyTorch para aprendizaje profundo, entre otros.

8. **Aprendizaje Fácil para Principiantes:**

- Python es a menudo recomendado como un lenguaje de programación para principiantes debido a su sintaxis clara y su enfoque en la legibilidad del código. Es un buen lenguaje para aprender los conceptos fundamentales de programación.

9. **Soporte para Programación Orientada a Objetos:**

- Python es un lenguaje que soporta programación orientada a objetos, lo que permite la encapsulación, herencia y polimorfismo.

🏃 Quick Learning Fundamentos de Python

https://www.youtube.com/watch?v=Nvm7JzhUpc4

Duración: 1 hora 36 minutos

2.4 Herramientas de CI/CD

● 2.4.0. Importancia de Control de Cambios en CI/CD

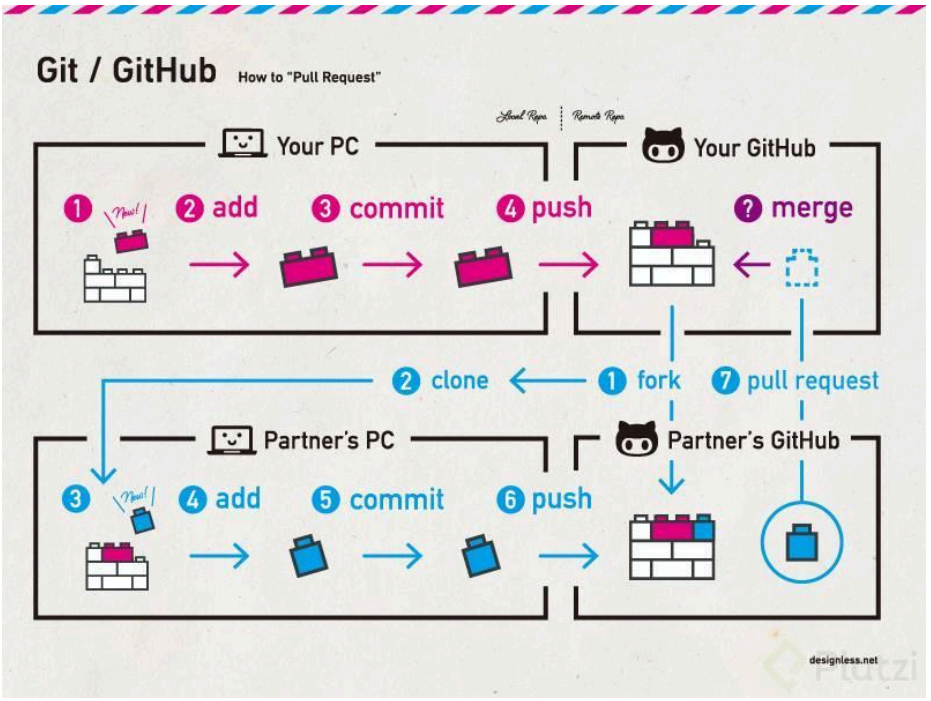
Pull Requests

Un **Pull Request** es una función de Git que permite a tu equipo solicitar la revisión y aprobación de sus cambios antes de fusionarlos en la rama principal de desarrollo, denominada “master” o “main”.

Al crear un PR se genera una conversación en la que los demás miembros pueden seguir y comentar los cambios propuestos en el código del repositorio. Esto permite detectar cualquier error o problema potencial antes de integrarlos en la rama principal, lo que ayuda a mantener el proyecto más limpio y estable.

Estructura de la incorporación de cambios

El proceso para hacer un pull request consiste en indicar la rama y repositorio de origen y destino. De esta forma, un desarrollador podrá incluir tus cambios en su proyecto.



Cómo hacer un pull request

Un **PR** es un proceso crucial para facilitar la revisión y la integración efectiva del código. A continuación, veremos el paso a paso.

Solicitando un pull request

1. **Crea una rama paralela**: Antes de hacer cambios en el código, utiliza el comando **git checkout -b <rama>** para crear una nueva rama. Así, podrás hacer tus modificaciones sin afectar la rama principal (por ejemplo, **master**).

2. **Realiza commits**: Después de hacer cambios en los archivos, usa **git commit -am '<Comentario>'** para hacer un commit con un mensaje descriptivo.

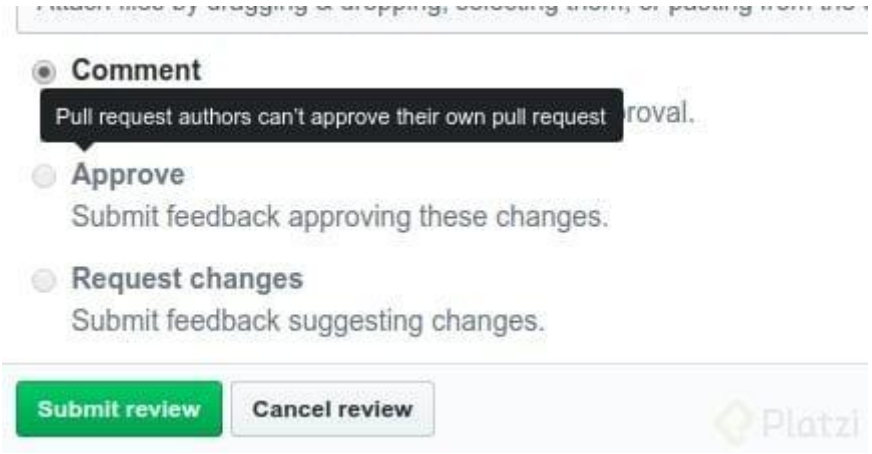
3. **Sube los cambios**: Usa **git push origin <rama>** para subir tus cambios de la rama local al repositorio remoto. Reemplaza **<rama>** con el nombre de tu rama.

4. **Crea un pull request**: En el repositorio remoto (como GitHub), crea un nuevo pull request. Selecciona la rama principal como destino y tu rama con los cambios como comparación.

5. **Feedback**: Los revisores examinarán los cambios. Usa la sección de comentarios del pull request para discutir los cambios y proporcionar feedback adicional.

6. **Realiza los cambios solicitados**: Si se solicitan cambios, regresa a tu rama local y haz las modificaciones necesarias. Luego, sube los cambios al repositorio remoto usando **git push origin <rama>**.

Aceptando un pull request



1. **Acepta los cambios en Git**: Si estás satisfecho con los cambios propuestos en el pull request y consideras que están listos para ser fusionados con la rama principal, acepta el pull request en Git. De esta forma, los cambios se fusionarán en la rama principal del repositorio.

2. **Realiza el merge en la rama principal**: Después de aceptar el pull request, selecciona la opción para realizar el merge en Git. Esto combinará los cambios de la rama con los cambios existentes en la rama principal (**master**).

¿Cómo corregir un Pull Request?

Al revisar un pull request, es posible encontrar problemas o áreas que necesiten corrección antes de que los cambios se puedan unir con la rama principal. Aquí se explica cómo corregir un pull request de forma efectiva.

1. **Lee los comentarios y feedback**: Comprueba cuidadosamente todos los comentarios y feedback proporcionados por los revisores en el pull request. Toma nota de los problemas señalados y las sugerencias de mejora.

2. **Regresa a tu rama local**: Utiliza el comando **git checkout <rama>** para volver a la rama en la que realizaste los cambios y necesitas revisar.

3. **Realiza las modificaciones**: Basándote en los comentarios y feedback recibidos, efectúa las revisiones necesarias en los archivos relevantes. Asegúrate de abordar todos los problemas señalados y seguir las sugerencias de mejora proporcionadas.

4. **Realiza un nuevo commit**: Después de efectuar las correcciones, utiliza el comando **git commit -am '<Comentario>'** para crear un nuevo commit que refleje

las correcciones realizadas. Asegúrate de proporcionar un mensaje claro y descriptivo para indicar las modificaciones realizadas.

5. **Sube los cambios al repositorio remoto**: Utiliza el comando **git push origin <rama>** para subir los cambios corregidos a la rama correspondiente en el repositorio remoto.

6. **Actualiza el pull request**: Una vez que los cambios corregidos se hayan subido al repositorio remoto, el pull request se actualizará automáticamente para reflejar las modificaciones procedidas en la rama. Los revisores podrán ver los nuevos cambios y comentarios.

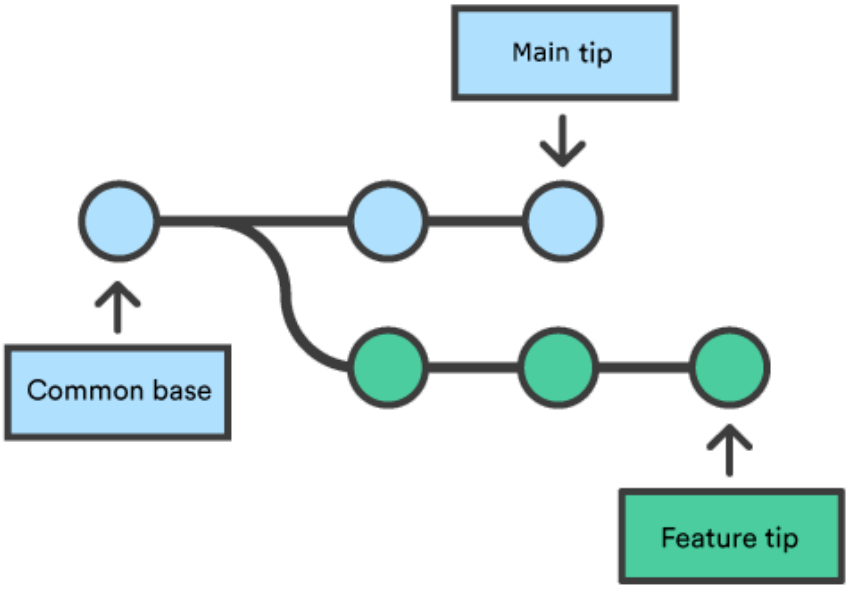
7. **Comunica las correcciones realizadas**: Si consideras que has abordado adecuadamente los problemas señalados y las sugerencias de mejora, puedes comunicar a los revisores que has realizado las correcciones necesarias y que el pull request está listo para ser revisado nuevamente.

Merge

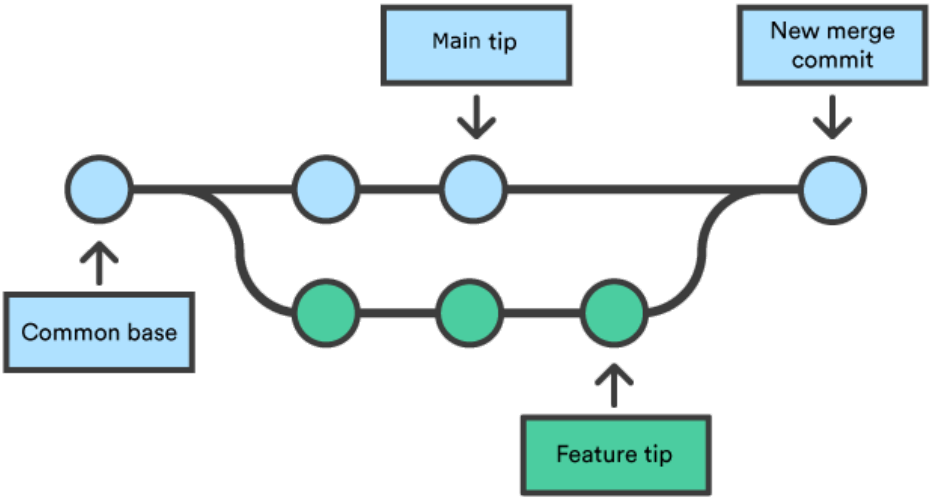
O también llamado Fusion. La fusión es la forma que tiene Git de volver a unir un historial bifurcado. El comando git merge permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama.

Ten en cuenta que todos los comandos presentados a continuación se fusionan en la rama actual. La rama actual se actualizará para reflejar la fusión, pero la rama de destino no se verá afectada en absoluto. Una vez más, esto significa que git merge se suele utilizar junto con git checkout para seleccionar la rama actual y git branch -d para eliminar la rama de destino obsoleta.

git merge combinará varias secuencias de confirmaciones en un historial unificado. En los casos de uso más frecuentes, git merge se utiliza para combinar dos ramas. Los ejemplos siguientes del presente documento se centrarán en este patrón de fusión de ramas. En estos casos, git merge toma dos punteros de confirmación, normalmente los extremos de la rama, y encuentra una confirmación base común entre ellos. Una vez que Git encuentra una confirmación base en común, crea una "confirmación de fusión" nueva que combina los cambios de cada secuencia de confirmación de fusión puesta en cola.
Supongamos que tenemos una rama de función nueva que se basa en la rama main. Ahora, queremos fusionar esa rama de función con la rama main.



Al invocar este comando, la rama de función especificada se fusionará con la rama actual, la cual asumimos que es la main. Git determinará el algoritmo de fusión automáticamente (véase más adelante).



Las confirmaciones de fusión son únicas con respecto a otras confirmaciones en el hecho de que tienen dos confirmaciones principales. Al crear una confirmación de fusión, Git tratará de fusionar automáticamente los historiales independientes. Sin embargo, si encuentra datos que se han cambiado en ambos historiales, no podrá combinarlos de ese modo. En ese caso, se crea un conflicto de control de versiones y Git solicitará la intervención del usuario para poder continuar.

● 2.4.1. Jenkins

Jenkins es un servidor open source para la integración continua. Es una herramienta que se utiliza para compilar y probar proyectos de software de forma continua, lo que facilita a los desarrolladores integrar cambios en un proyecto y entregar nuevas versiones a los usuarios. Escrito en Java, es multiplataforma y accesible mediante interfaz web. Es el software más utilizado en la actualidad para este propósito.

Con Jenkins, las organizaciones aceleran el proceso de desarrollo y entrega de software a través de la automatización. Mediante sus centenares de plugins, se puede implementar en diferentes etapas del ciclo de vida del desarrollo, como la compilación, la documentación, el testeo o el despliegue.

Origen de Jenkins

La primera versión de **Jenkins surgió en 2011**, pero su desarrollo se inició en 2004 como parte del **proyecto Hudson**. Kohsuke Kawaguchi, un desarrollador de Java que trabajaba en Sun Microsystems, creó un servidor de automatización para facilitar las tareas de compilación y de realización de pruebas.

En 2010, surgieron discrepancias relativas a la gestión del proyecto entre la comunidad y Oracle y, finalmente, se decidió el **cambio de denominación a "Jenkins"**. Desde entonces los dos proyectos continuaron desarrollándose independientemente. Hasta que finalmente Jenkins se impuso al ser utilizado en muchos más proyectos y contar con más contribuyentes.

¿Qué es la integración continua (CI)? (Repaso)

La **integración continua o Continuous Integration (CI)** es una práctica habitual en desarrollo de software que consiste en integrar frecuentemente mejoras en el código de un proyecto una vez han sido validadas, normalmente varias veces al día, con el objetivo de detectar errores lo antes posible.

Cada cambio que realiza un desarrollador (ya sea una resolución de un bug, la creación de una nueva funcionalidad, etc.) se comprueba **compilando el código fuente y obteniendo un ejecutable (llamado *build*).** Si es validado, será incorporado al código fuente y desplegado.

Realizar *builds* frecuentemente y comprobar su funcionamiento ayuda a conseguir un **producto final más fiable**, al prevenir fallos en producción. Gracias a la integración continua los desarrolladores pueden detectar y corregir errores constantemente, evitar despliegues traumáticos y reducir notablemente el *time to market*.

Además, gracias a herramientas como Jenkins, se puede **conocer el estado del software en cada momento**, monitorizar la calidad del código, la cobertura de las pruebas y reducir la deuda técnica y los costos asociados.

La integración continua es **parte integral de DevOps** y trabaja en la línea de entregar valor a los usuarios en el menor tiempo posible. Se vincula habitualmente con el Extreme Programming (XP) y las metodologías ágiles.

¿Por qué usar Jenkins para realizar CI?

Antes de disponer de herramientas como Jenkins para poder aplicar integración continua nos encontrábamos con un escenario en el que:

- **Todo el código fuente era desarrollado y luego testeado**, con lo que los despliegues y las pruebas eran muy poco habituales y localizar y corregir errores era muy laborioso. El tiempo de entrega del software se prolongaba.

- Los desarrolladores tenían que **esperar al desarrollo de todo el código** para poner a prueba sus mejoras.

- **El proceso de desarrollo y testeo eran manuales**, por lo que era más probable que se produjeran fallos.

Sin embargo, con Jenkins (y la integración continua que facilita) la situación es bien distinta:

- **Cada commit es desarrollado y verificado.** Con lo que, en lugar de comprobar todo el código, los desarrolladores sólo necesitan centrarse en un commit concreto para corregir bugs.

- Los desarrolladores **conocen los resultados de las pruebas de sus cambios** durante la ejecución.

- **Jenkins automatiza el despliegue y las pruebas**, lo que ahorra mucho tiempo y evita errores.

- **El ciclo de desarrollo es más rápido.** Se entregan más funcionalidades y más frecuentemente a los usuarios, con lo que los beneficios son mayores.

¿Qué se puede hacer con Jenkins?

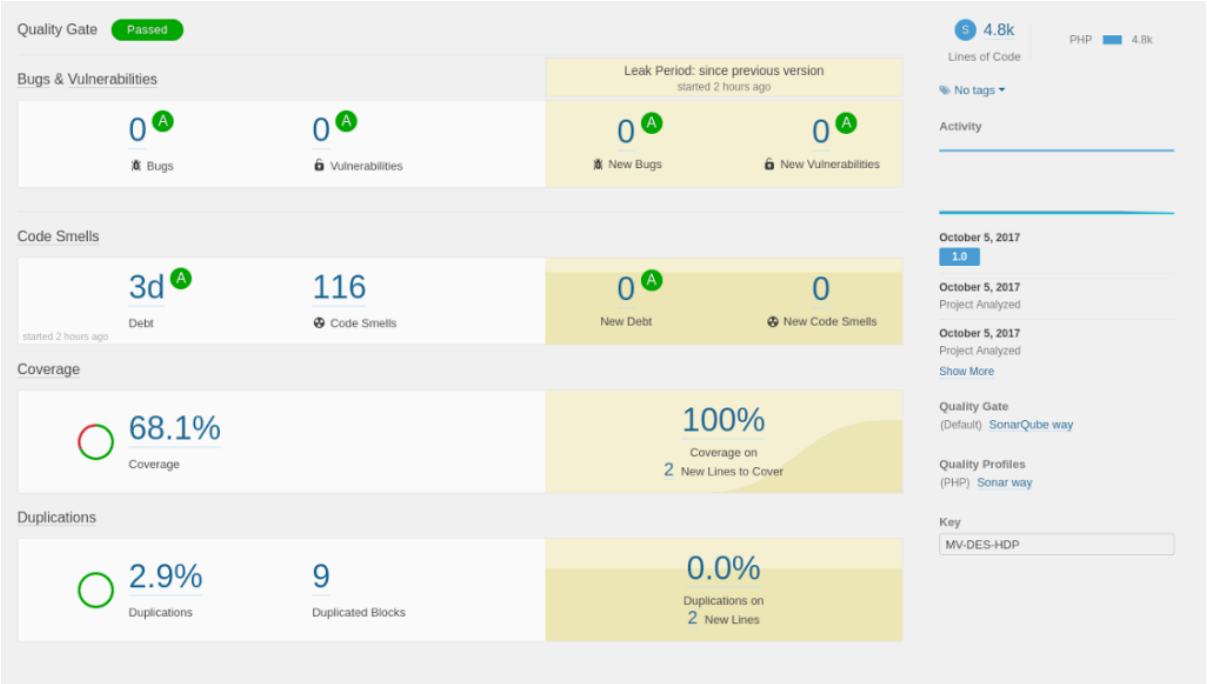
Con Jenkins podemos **automatizar multitud de tareas** que nos ayudarán a reducir el *time to market* (TTL) de nuestros productos digitales o de nuevas versiones de ellos. Concretamente, con esta herramienta podemos:

- **Automatizar** la compilación y testeo de software.
- **Notificar** a los equipos correspondientes la detección de errores.
- **Desplegar los cambios** en el código que hayan sido validados.
- **Hacer un seguimiento** de la calidad del código y de la cobertura de las pruebas.
- **Generar la documentación** de un proyecto.

Podemos ampliar las funcionalidades de Jenkins a través de **múltiples plugins creados por la comunidad**, diseñados para ayudarnos en centenares de tareas, a lo largo de las diferentes etapas del proceso de desarrollo

● 2.4.2. SonarQube

El análisis estático del código es el proceso de evaluar el software sin ejecutarlo. La idea de este análisis es que teniendo como entrada nuestro código fuente, podamos obtener información y métricas que nos permita mejorar la base de código detectando errores típicos de programación, bugs, code smells, etc. Esta herramienta nos hará sugerencias sobre que partes del código son mejorables. Una herramienta muy conocida utilizada para tal fin es SonarQube



¿Qué tipo de problemas podemos resolver?

Un análisis estático del código no es suficiente, necesitamos también otro tipo de herramientas como los tests unitarios para verificar el correcto uso y calidad de nuestro software, pero haciendo un análisis del código sin estar en ejecución podemos detectar elementos como:

- **Problemas de Diseño:** Podemos detectar problemas en el diseño y arquitectura del software analizando las dependencias entre las clases del proyecto. Esto nos permite actuar a tiempo frente a crear un enmarañado de clases totalmente acopladas y difícilmente reutilizables.

- **Duplicidad de Código:** Sonarqube nos proporciona métricas de código duplicado, pudiendo detectar partes de nuestro software se asemejan, pudiendo así tomar decisiones como desacoplar componentes o aplicar técnicas de refactoring utilizando el polimorfismo, la herencia y la reutilización de componentes. Recuerda: Don't Repeat Yourself!

- **Detección de Vulnerabilidades:** SonarQube cuenta con una base de datos de codesmells y errores típicos de programación que detectan si alguna línea de código puede estar cometiendo algún problema que pueda vulnerar la seguridad. Por ejemplo, a la hora de cómo recoger los parámetros o cómo usarlos en nuestras consultas para evitar SQL Injection.

- **Standard de codificación**: Avisándonos de partes del código que no cumplan con el PSR o incluyan malas prácticas a la hora de definir constantes, variables, llamadas a métodos estáticos.

- **Monitorización de Cobertura:** En nuestro caso también lo usamos para poder monitorizar si la cobertura de los tests es aceptable y de esta manera tener una visión global del estado de la cobertura de todos los proyectos e invertir en incrementar el volumen de tests del proyecto.

Una puerta de calidad es un indicador de calidad de código que se puede configurar para dar una señal de "listo/no listo" en la calidad actual del código.
Indica si el código está limpio y puede avanzar.

- **Una puerta de calidad que pasa (verde) significa que el código cumple con su estándar y está listo para fusionarse.**

- **Una puerta de calidad que falla (roja) significa que hay problemas que abordar.**

SonarQube proporciona retroalimentación a través de su interfaz de usuario, correo electrónico y en decoraciones en solicitudes de extracción o fusiones (en ediciones comerciales) para notificar al equipo que hay problemas que abordar.

También se pueden obtener comentarios en los IDE compatibles con SonarLint cuando se ejecuta en modo conectado. SonarQube también ofrece una guía detallada sobre los problemas, indicando por qué cada problema es un problema y cómo solucionarlo, añadiendo una valiosa capa de educación para desarrolladores de todos los niveles de experiencia. Los desarrolladores pueden abordar los problemas de manera efectiva, por lo que el código solo se promueve cuando está limpio y pasa la puerta de calidad.

¿Cómo uso SonarQube? ¿Es lo mismo SonarQube que Sonar?

Si bien se suele abreviar al hablar y se dice Sonar, Sonar es la solución completa de los creadores de SonarQube.

Puede ejecutar una instancia local no productiva de SonarQube y realizar un análisis inicial del proyecto. Instalar una instancia local permite ponerse en funcionamiento rápidamente, para experimentar SonarQube de primera mano. Luego, cuando esté listo para configurar SonarQube en producción, deberá instalar el servidor antes de configurar su primer análisis de código.

Para efectuar la instalación de SonarQube puede referirse a la documentación oficial:

https://docs.sonarsource.com/sonarqube/latest/setup-and-upgrade/overview/

🏃 Quick Learning Instalación y configuración de SonarQube

Duración 17:25
https://www.youtube.com/watch?v=uduDckVSILE

● 2.4.3. GitHub / GitLab

Introducción a GitHub

GitHub es una plataforma de desarrollo colaborativo basada en la nube que facilita la gestión y colaboración en proyectos de software. Fundada en 2008, GitHub se ha convertido en una herramienta esencial para desarrolladores y equipos de programación.

Como la plataforma esta basada en GIT, no hay mucho mas para aprender sobre como maneja la parte de repositorios.

En cambio, tiene una sección de CI/CD llamada GitHub Actions, que maneja pipelines y flujos de trabajo para efectuar los despliegues.

GitHub Actions

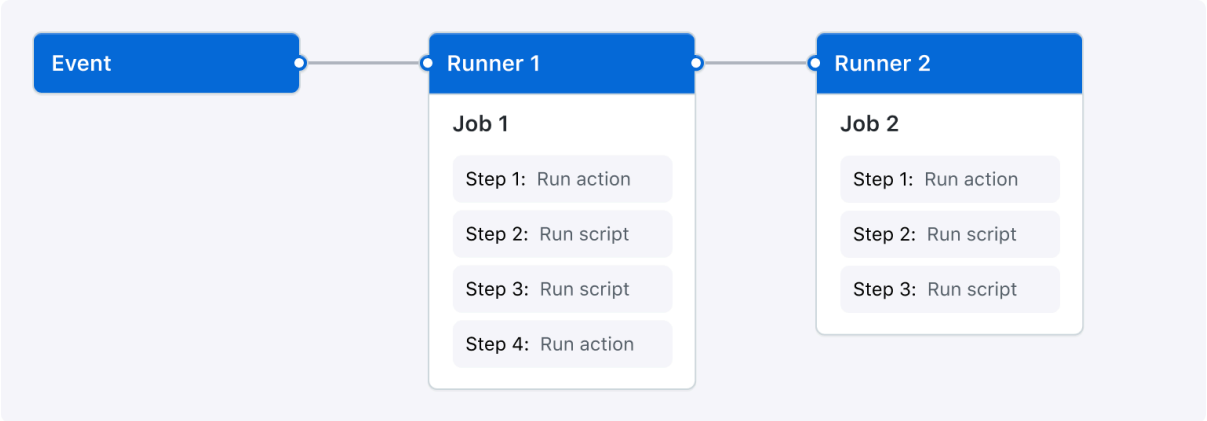
Es una plataforma de integración y despliegue continuos (IC/DC) que te permite automatizar tu mapa de compilación, pruebas y despliegue. Puedes crear flujos de trabajo y crear y probar cada solicitud de cambios en tu repositorio o desplegar solicitudes de cambios fusionadas a producción.

GitHub Actions va más allá de solo DevOps y te permite ejecutar flujos de trabajo cuando otros eventos suceden en tu repositorio. Por ejemplo, puedes ejecutar un flujo de trabajo para que agregue automáticamente las etiquetas adecuadas cada que alguien cree una propuesta nueva en tu repositorio.

GitHub proporciona máquinas virtuales Linux, Windows y macOS para que ejecutes tus flujos de trabajo o puedes hospedar tus propios ejecutores auto-hospedados en tu propio centro de datos o infraestructura en la nube.

Puede configurar un *flujo de trabajo* de GitHub Actions que se desencadene cuando se produzca un *evento* en el repositorio, por ejemplo, la apertura de una solicitud de incorporación de cambios o la creación de una incidencia. El flujo de trabajo contiene uno o varios *trabajos* que se pueden ejecutar en orden secuencial o en paralelo. Cada trabajo

se ejecutará dentro de su propio *ejecutor* de máquina virtual o dentro de un contenedor, y tendrá uno o varios *pasos* que pueden ejecutar un script que defina, o bien una *acción*, que es una extensión reutilizable que puede simplificar el flujo de trabajo.



Workflows o Flujos de Trabajo

Un flujo de trabajo es un proceso automatizado configurable que ejecutará uno o más jobs. Los flujos de trabajo se definen mediante un archivo de YAML que se verifica en tu repositorio y se ejecutará cuando lo active un evento dentro de este o puede activarse manualmente o en una programación definida.
Los flujos de trabajo se definen en el directorio `.github/workflows` de un repositorio y un repositorio puede tener varios flujos de trabajo, cada uno de los cuales puede realizar un conjunto diferente de tareas. Por ejemplo, puedes tener un flujo de trabajo para crear y probar las solicitudes de cambio, otro para desplegar tu aplicación cada que se cree un lanzamiento y todavía otro más que agregue una etiqueta cada que alguien abra una propuesta nueva.

Eventos

Un evento es una actividad específica en un repositorio, la cual activa una ejecución de flujo de trabajo. Por ejemplo, la actividad puede originarse desde GitHub cuando alguien crea una solicitud de cambios, abre una propuesta o sube una confirmación a un repositorio. También puedes desencadenar una ejecución de flujo de trabajo según una programación, mediante la publicación en una API de REST o manualmente.

Trabajos

Un trabajo es un conjunto de *pasos* de un flujo de trabajo que se ejecuta en el mismo ejecutor. Cada paso puede ser un script de shell o una *acción* que se ejecutarán. Los

pasos se ejecutarán en orden y serán dependientes uno del otro. Ya que cada paso se ejecuta en el mismo ejecutor, puedes compartir datos de un paso a otro. Por ejemplo, puedes tener un paso que compile tu aplicación, seguido de otro que pruebe la aplicación que se compiló.

Puedes configurar las dependencias de un job con otros jobs; predeterminadamente, los jobs no tienen dependencias y se ejecutan en paralelo entre ellos. Cuando un job lleva una dependencia a otro job, este esperará a que el job dependiente se complete antes de que pueda ejecutarse. Por ejemplo, puedes tener jobs de compilación múltiple para arquitecturas diferentes que no tengan dependencias y un job de empaquetado que sea dependiente de estos jobs. Los jobs de compilación se ejecutarán en paralelo y, cuando se hayan completado con éxito, se ejecutará el job de empaquetado.

Acciones

Una *acción* es una aplicación personalizada para la plataforma de GitHub Actions que realiza una tarea compleja pero que se repite frecuentemente. Utiliza una acción para ayudarte a reducir la cantidad de código repetitivo que escribes en tus archivos de flujo de trabajo. Una acción puede extraer tu repositorio de git desde GitHub, configurar la cadena de herramientas correcta para tu ambiente de compilación o configurar la autenticación en tu proveedor de servicios en la nube.
Puedes escribir tus propias acciones o puedes encontrar acciones para utilizar en tus flujos de trabajo dentro de GitHub Marketplace.

Ejecutores

Un ejecutor es un servidor que ejecuta tus flujos de trabajo cuando se activan. Cada ejecutor puede ejecutar un job individual a la vez. GitHub proporciona ejecutores de Ubuntu Linux, Microsoft Windows y macOS para ejecutar tus flujos de trabajo; cada flujo de trabajo se ejecuta en una máquina virtual nueva y recién aprovisionada. GitHub también ofrece ejecutor más grande, que están disponibles en configuraciones más grandes.

Flujo de Trabajo de Ejemplo en GitHub Actions

Las GitHub Actions utilizan la sintaxis de YAML para definir el flujo de trabajo. Cada flujo de trabajo se almacena como un archivo YAML independiente en el repositorio de código, en un directorio denominado: **.github/workflows.**

Puedes crear un flujo de trabajo de ejemplo en tu repositorio que active automáticamente una serie de comandos cada que se suba código. En este flujo de

trabajo GitHub Actions extrae el código insertado, instala el marco de pruebas de bats y ejecuta un comando básico para generar la versión de los bats: bats -v.

En el repositorio, cree el directorio **.github/workflows/** para almacenar los archivos de flujo de trabajo.

En el directorio .github/workflows/, cree un archivo denominado learn-github-actions.yml y agregue el código siguiente.

```
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

Confirma estos cambios y cargalos a tu repositorio de GitHub.

Tu archivo de flujo de trabajo de GitHub Actions nuevo estará ahora instalado en tu repositorio y se ejecutará automáticamente cada que alguien suba un cambio a éste.

🏃 Quick Learning Github Actions

https://www.youtube.com/watch?v=5GhWrEj35K4
Duración: 25:00

Introducción a GitLab

GitLab es una plataforma de desarrollo de software basada en la web que proporciona un conjunto completo de herramientas para la gestión del ciclo de vida del desarrollo de software. Al igual que GitHub, GitLab utiliza el sistema de control de versiones Git para el seguimiento de cambios en el código fuente.

🏃 Quick Learning GiLab CI/CD

https://www.youtube.com/watch?v=I2qztG7vxPQ
Duración: 1 hora 10 minutos

🏃 Quick Learning Configurar CI/CD con GitLab

https://www.youtube.com/watch?v=2W95rnyU6co
Duración: 8 minutos

● 2.4.4. Azure DevOps

Azure DevOps es un conjunto de Servicios que Microsoft ofrece como continuación de una herramienta que hizo punta en el mercado de Repositorios y manejo de Issues. Si, hablamos de TFS (Team Foundation Services). Este también era la evolución de una herramienta anterior, pero dejémoslo ahí de momento. TFS evolucionó para brindar servicios varios, no solo Repositorio de código fuente, se le llamó en primera instancia, y cuando aún no era un servicio ofrecido por Azure, Microsoft Release Management.

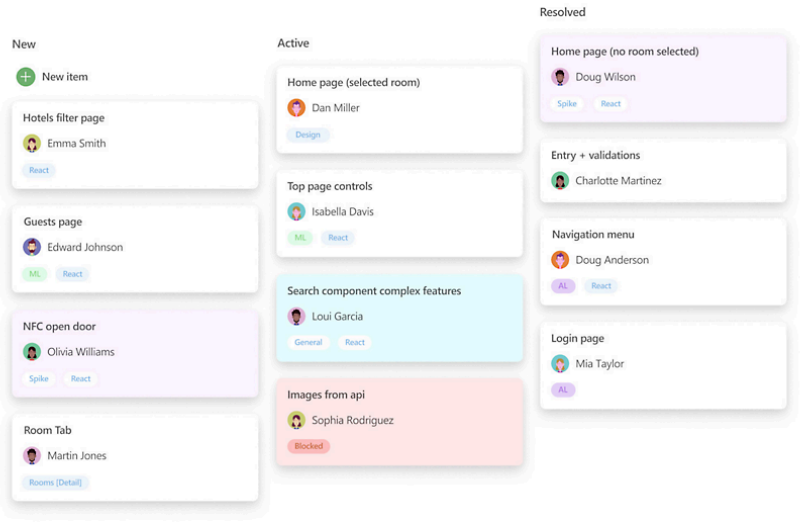
Incluye servicios varios para brindar las diferentes funcionalidades que en el mercado ofrecen productos separados, de manera que tengamos todo junto en un solo lugar. Esto no solo habilita a los profesionales de TI a simplificar el trabajo sino también los costos asociados a estas herramientas.

Por otro lado, esto también ha traído confusión a los usuarios comunes, quienes no comprenden que DevOps es una cultura y no una herramienta.

Veamos algunos de los servicios que incluye y sus funcionalidades:

Azure Boards 

Mantenga un seguimiento del trabajo con paneles Kanban, registros de trabajo pendiente interactivos y herramientas de planeamiento muy eficaces. Los informes y la rastreabilidad sin igual convierten a Azure Boards en el lugar perfecto para todas sus ideas, grandes o pequeñas.

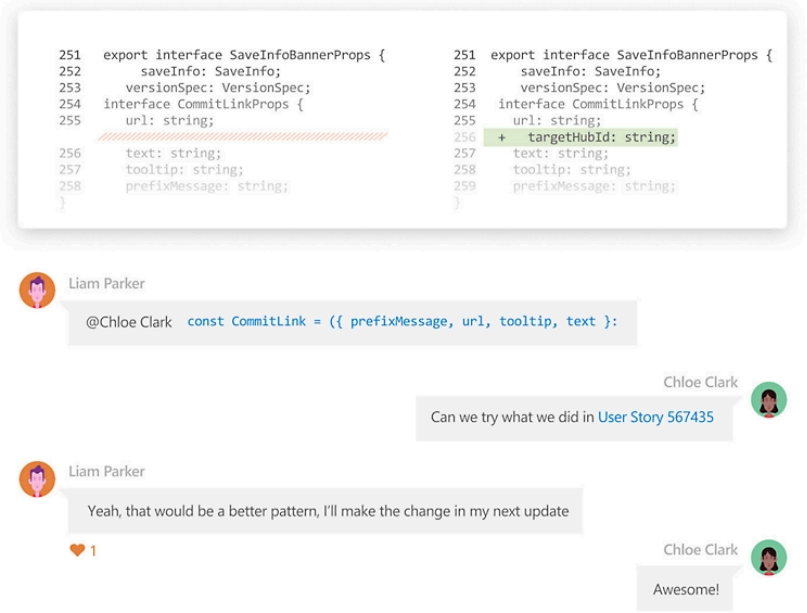


Otras herramientas de Mercado con la misma funcionalidad:

- Jira
- Trello
- Kanbanboard

Azure Repos 

Consiga un excelente hospedaje GIT, flexible y con revisiones de código muy eficaces, así como repositorios gratuitos ilimitados para todas sus ideas, desde un proyecto de una sola persona hasta el repositorio más grande del mundo.

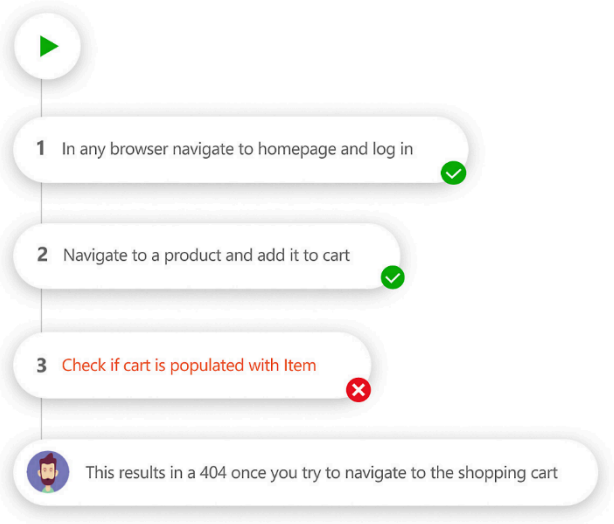


Otras herramientas de Mercado con la misma funcionalidad:

- GitHub (es la misma herramienta dado que fue adquirida por Microsoft)
- GitLab
- Bitbucket

Azure Test Plans

Realice pruebas periódicas y publique versiones con confianza. Mejore la calidad global del código con herramientas de pruebas manuales y exploratorias de DevOps para sus aplicaciones.

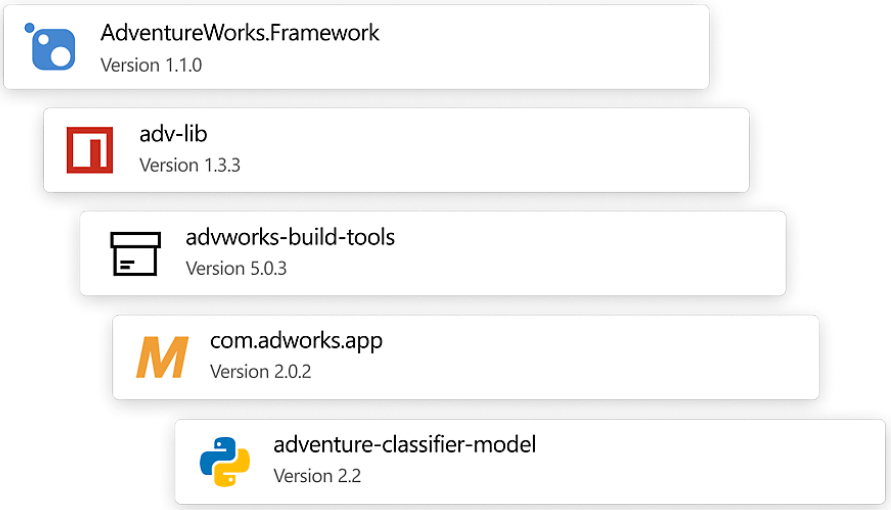


Otras herramientas de Mercado con la misma funcionalidad:

- JIRA
- BrowserStack
- GitLab
- TestRails

Azure Artifacts

Comparta paquetes Maven, npm, NuGet y Python de orígenes públicos y privados con todo su equipo. Integre el uso compartido de paquetes en sus canalizaciones de CI/CD de una forma sencilla y escalable.



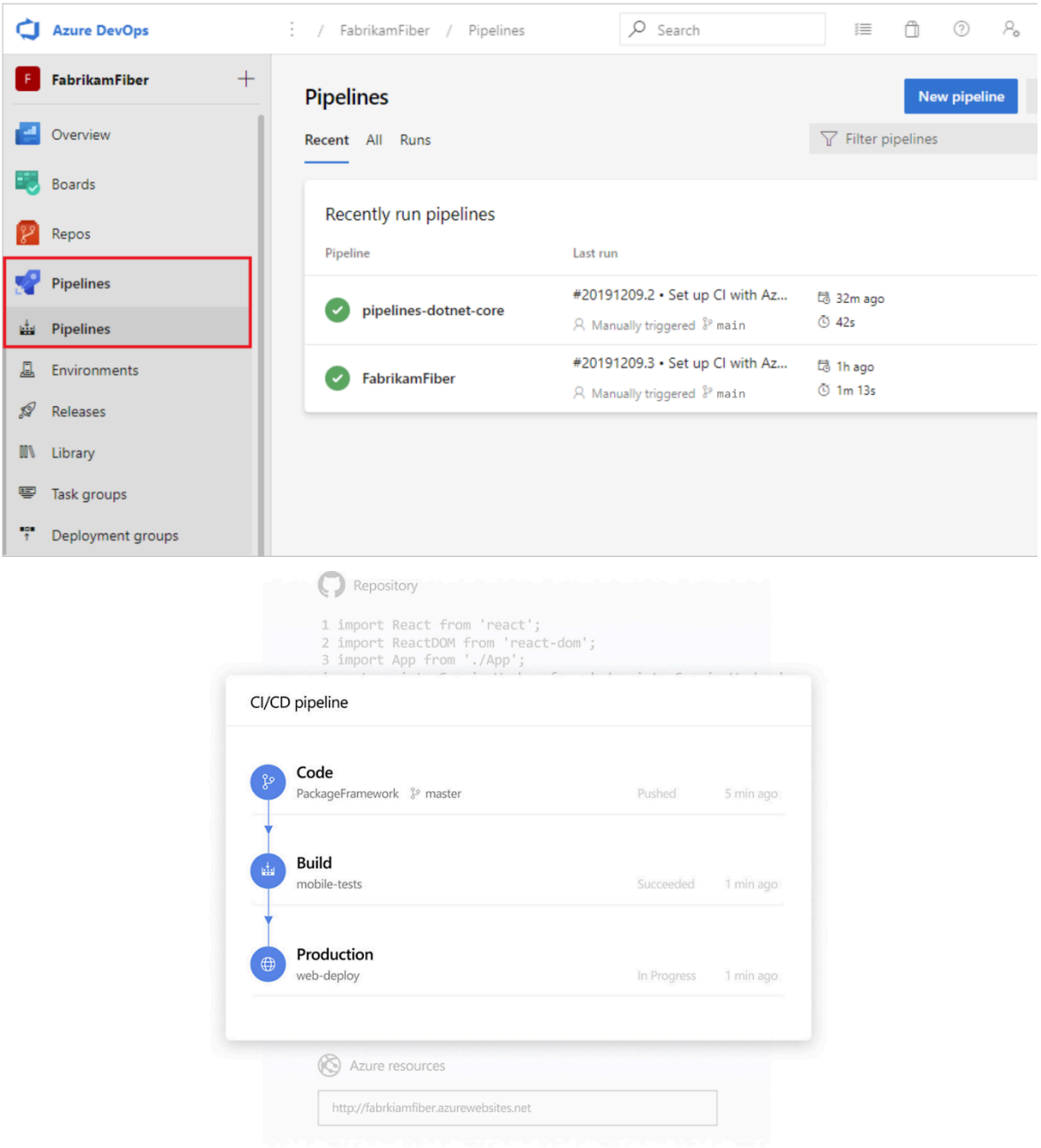
Otras herramientas de Mercado con la misma funcionalidad:

- Artifactory (JFrog)
- Nexus (Sonatype)
- GitLab
- GitHub
- DockerHub

Azure Pipelines

Compile, pruebe e implemente soluciones con cualquier lenguaje, en cualquier nube o en el entorno local. Ejecute archivos en paralelo en Linux, macOS y Windows, e implemente contenedores en hosts individuales o en Kubernetes.

Azure Pipelines compila y prueba automáticamente los proyectos de código. Admite todos los lenguajes principales y los tipos de proyecto y combina la integración continua, la entrega y las pruebas continuas para compilar, probar y entregar el código a cualquier destino.

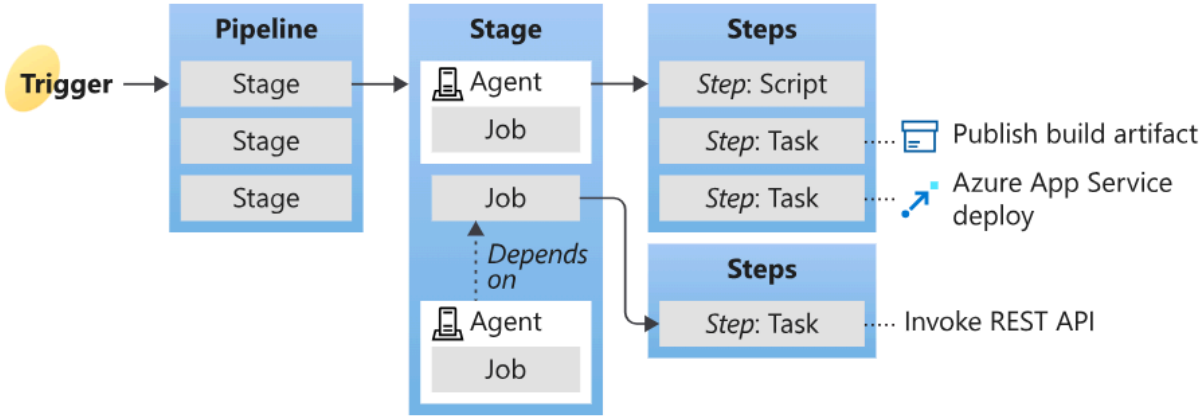


Otras herramientas de Mercado con la misma funcionalidad:

- GitLab
- GitHub
- Bitbucket
- Jenkins
- TeamCity
- CircleCI
- AWS Code Pipelines
- GCP Code Build and Pipelines

Generando Pipelines en Azure DevOps

Un pipeline en Azure DevOps es una representación visual y/o codificada de un proceso de entrega continua (CI) y/o implementación continua (CD). Un pipeline describe las etapas y acciones que deben ejecutarse para construir, probar y desplegar una aplicación o servicio de software. Puedes definir un pipeline en Azure DevOps utilizando el Editor de Diseño visual o mediante un archivo YAML.



Editor de Diseño Visual:

1. **Inicio en Azure DevOps:**
   - Inicia sesión en tu instancia de Azure DevOps y navega al proyecto en el que deseas crear o editar un pipeline.
2. **Ir a Pipelines:**
   - Dirígete a la sección "Pipelines" en el menú de navegación.
3. **Crear Nuevo Pipeline:**
   - Haz clic en el botón "New Pipeline" para comenzar a crear un nuevo pipeline.
4. **Seleccionar Repositorio:**
   - Selecciona el repositorio de código fuente asociado con tu aplicación.
5. **Elegir Plantilla o Comenzar en Blanco:**
   - Puedes comenzar con una plantilla predeterminada que coincida con el tipo de aplicación que estás desarrollando o puedes comenzar en blanco.
6. **Configuración del Pipeline:**
   - Utiliza el Editor de Diseño visual para agregar pasos o tareas al pipeline. Estos pasos pueden incluir la restauración de dependencias, la construcción de la aplicación, la ejecución de pruebas, y el despliegue a diferentes entornos.
7. **Configuración de Variables y Parámetros:**

- Puedes configurar variables y parámetros que se utilizan en el pipeline, lo que facilita la personalización y reutilización de este.

8. **Revisar y Guardar:**
   - Revisa y valida tu pipeline antes de guardarlo. Puedes agregar opciones avanzadas y configuraciones adicionales según tus necesidades.
9. **Ejecutar y Monitorear:**
   - Después de guardar el pipeline, puedes ejecutarlo manualmente o configurar desencadenadores automáticos basados en eventos de código, como confirmaciones o cambios en una rama.

Pipeline YAML

Alternativamente, puedes definir tu pipeline utilizando un archivo YAML.
Como ya comentamos anteriormente en este curso YAML es un standard hoy en día para la mayoría de los sistemas de gestión de CI/CD. No obstante, las palabras clave utilizadas para cada uno de ellos pueden variar. Recomendamos no aprender las palabras de memoria, dado que cambian todo el tiempo mientras las herramientas evolucionan. En lugar de esto, es una buena practica referirse a la documentación de estas para referencia.

```
trigger:
- main

pool:
  vmImage: 'windows-latest'

steps:
- script: echo Hello, world!
  displayName: 'Run a one-line script'
```

Este archivo YAML especifica que el pipeline se activará en cambios en la rama main, utilizará una imagen de máquina virtual de Windows y ejecutará un script simple que imprime “Hello, ¡world!".

Quick Learning Pipeline de CI con Azure DevOps

https://www.youtube.com/watch?v=YY4EAp8yeN0
Duración: 18:00

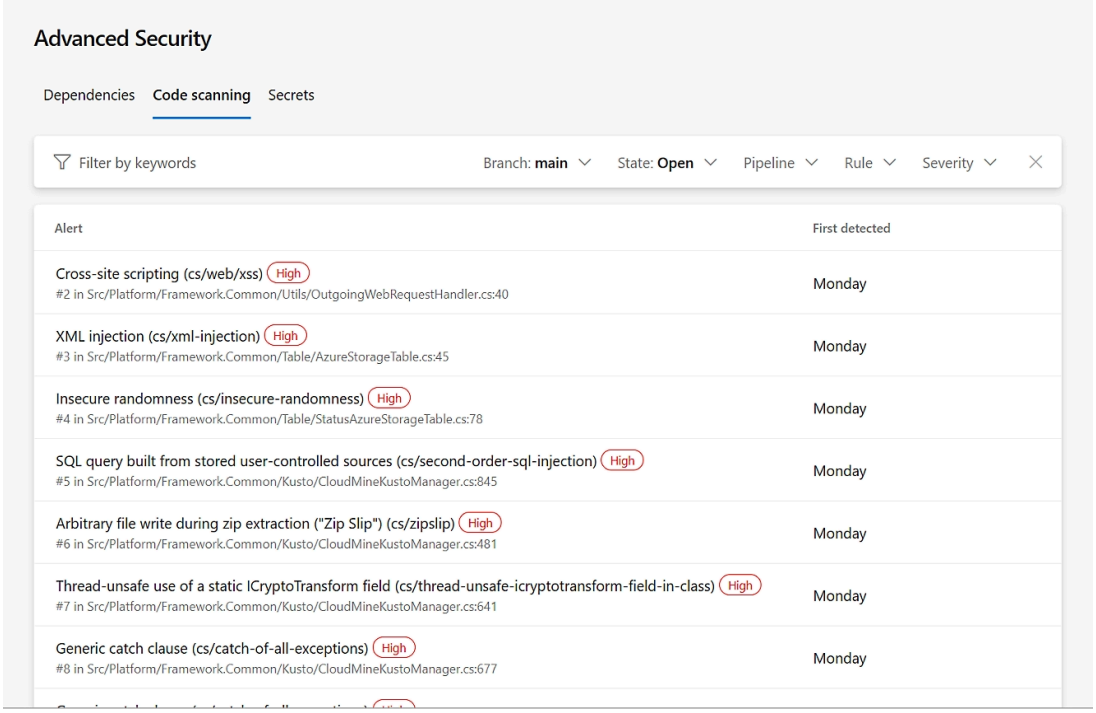
Quick Learning YAML Pipelines

https://www.youtube.com/watch?v=jFmSPpmgeR8
Duración: 7:30

GitHub advanced security para Azure DevOps

GitHub Advanced Security es un servicio de pruebas de seguridad de aplicaciones (AppSec) que se inserta en el flujo de trabajo del desarrollador. Permite a los equipos de desarrollo, seguridad y operaciones (DevSecOps) priorizar la innovación y mejorar la productividad de los desarrolladores sin sacrificar la seguridad.

Lo que permite hacer este servicio es hacer seguimiento y pruebas de seguridad para las aplicaciones desplegadas. Esto es, a partir de vulnerabilidades conocidas analizar si un software que usted está preparando posee dicha vulnerabilidad. Algo así como funciona ZAP OWASP pero con esteroides. Es un mix de SAST (SonarQube) con DAST (Zap/Nexus). Analiza no sólo código fuente, sino también dependencias en caso de que paquetes externos poseen vulnerabilidades.



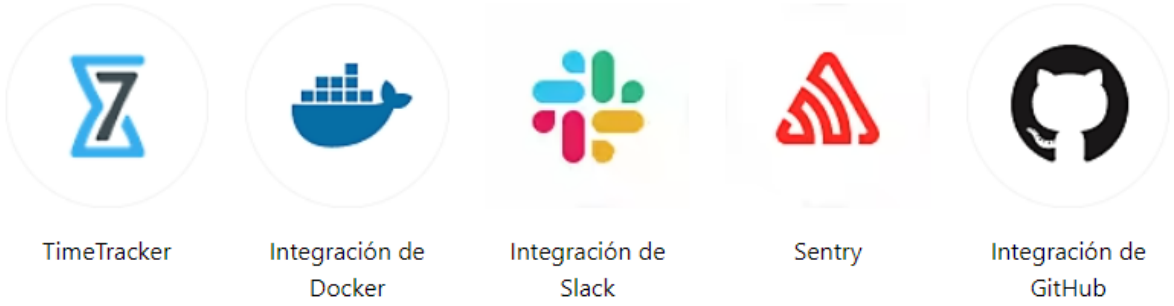
¿Como que Github con Azure DevOps?

Si, puntualmente Microsoft es dueño de GitHub. Es por esto por lo que, aunque las dos herramientas siguen existiendo de manera separada, en un futuro cercano pasaran a ser una sola. Es probable que Azure DevOps deje de existir como tal y se presente como un servicio de GitHub directamente.

Incluso si bien la funcionalidad de este servicio del que estamos hablando no es esa, hay planes para incorporar GitHub Boards muy pronto.

Visual Studio MarketPlace

Acceda a más de 1000 extensiones o cree las suyas propias.



● 2.4.5. AWS Code Pipeline

AWS CodePipeline es un servicio de integración y entrega continuas para realizar actualizaciones de aplicaciones e infraestructura rápidas y fiables. Puede utilizar CodePipeline para modelar y automatizar totalmente los procesos de lanzamiento de software.

Modelado de flujo de trabajo

Una canalización (Pipeline) define el flujo de trabajo del proceso de lanzamiento y describe el avance de una nueva modificación del código por el proceso de lanzamiento. Una canalización incluye una serie de fases (p. ej., compilación, prueba e implementación) que actúan como divisiones lógicas en el flujo de trabajo.

Cada fase se compone de una secuencia de acciones, que son tareas como la compilación de código o la implementación en entornos de prueba. AWS CodePipeline le proporciona una interfaz de usuario gráfica para crear, configurar y administrar la canalización y las diversas fases y acciones, lo que le permite visualizar y modelar con facilidad el flujo de trabajo del proceso de lanzamiento.

Ejecución paralela

Puede utilizar CodePipeline para modelar las acciones de compilación, prueba e implementación a los fines de que se ejecuten en paralelo y se incremente la velocidad del flujo de trabajo.

Integraciones

AWS CodePipeline permite extraer código fuente para la canalización directamente desde AWS CodeCommit, GitHub, Amazon ECR o Amazon S3. Puede ejecutar versiones

y pruebas unitarias en AWS CodeBuild. CodePipeline puede implementar sus cambios con AWS CodeDeploy, AWS Elastic Beanstalk, Amazon Elastic Container Service (AmazonEC2) o AWS Fargate.

También puede configurar acciones de AWS CloudFormation que le permiten aprovisionar, actualizar o eliminar recursos de AWS como parte del proceso de lanzamiento. Esto también le permite entregar aplicaciones sin servidor de manera constante creadas con AWS Lambda, Amazon API Gateway y Amazon DynamoDB con el modelo de aplicaciones sin servidor de AWS (AWS SAM).

Puede además activar funciones personalizadas definidas por código en cualquier lugar de la canalización gracias a la integración de CodePipeline con AWS Lambda. Por ejemplo, puede activar una función de Lambda que compruebe si su aplicación web se ha implementado correctamente.

CodePipeline puede configurar una canalización que enlace estos servicios con herramientas de desarrollo de terceros y sistemas personalizados.

Plantillas declarativas

AWS CodePipeline le permite definir la estructura de la canalización mediante un documento JSON declarativo que especifica el flujo de trabajo del lanzamiento, así como las fases y acciones. Estos documentos le permiten actualizar canalizaciones existentes y proporcionar plantillas iniciales para crear nuevas canalizaciones.

Control de acceso

AWS CodePipeline utiliza AWS IAM para administrar quién puede realizar cambios en el flujo de trabajo del lanzamiento y quién puede controlarlo. Puede otorgar a los usuarios acceso a través de usuarios de IAM, roles de IAM y directorios integrados en SAML.

Quick Learning AWS Code Pipeline

https://www.youtube.com/watch?v=ZKTrPObkQm4
Duración: 8:00

Si lo desea puede ejecutar esta práctica que le dará los conocimientos necesarios para trabajar con AWS Pipelines.

Tenga en cuenta que todos los servicios de CI CD funcionan de manera similar, esto quiere decir que, si en su trabajo le solicitan generar un pipeline para TravisCI o para BitBucket no tendrá mayores inconvenientes, debido a que la lógica es la misma, aunque difie

Resumen

En este manual vimos Automatización, Herramientas e Infraestructura para profesionales de TI, explorando sistemas operativos, virtualización y scripting. La sección sobre redes destaca conceptos clave, desde el Modelo OSI hasta troubleshooting, junto con la automatización en networking y la arquitectura de red en la nube. Se profundizo en fundamentos de scripting, incluyendo Powershell, BASH y Python. La importancia del Control de Cambios en CI/CD se destaca en la sección final, donde se exploran herramientas esenciales como Jenkins, SonarQube, GitHub/GitLab, Azure DevOps y AWS Code Pipeline.

Referencias

Libros

1. "The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win" de Gene Kim, Kevin Behr y George Spafford.
2. "Automate This: How Algorithms Came to Rule Our World" de Christopher Steiner.
3. "Networking All-in-One for Dummies" de Doug Lowe.
4. "PowerShell in Action" de Bruce Payette.
5. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation" de Jez Humble y David Farley.

Papers

1. "DevOps at Scale: A Hands-On Guide to Scaling DevOps for the Enterprise" - Disponible en https://cloud.google.com/solutions/devops/devops-at-scale.
2. "A Comprehensive Guide to Continuous Integration, Continuous Deployment, and Continuous Delivery" - Disponible en https://www.infoq.com/articles/CI-CD-CD/.
3. "Understanding the Basics of Network Troubleshooting" - Disponible en https://www.cio.com/article/286690/networking-understanding-the-basics-of-network-troubleshooting.html.
4. "Jenkins: The Definitive Guide" de John Ferguson Smart - Disponible en https://www.jenkins.io/doc/book/.

¡Muchas Gracias!

Nos vemos en el siguiente módulo 💪🏽