# Junior DevOps - Загальні питання

## 1. Що таке DevOps?

DevOps - це методологія або підхід до розробки програмного забезпечення, який поєднує розробку (Development) і експлуатацію (Operations). Згідно з моделлю DevOps, команди розробки та операцій більше не є «відокремленими». Іноді ці дві команди навіть об’єднуються в одну, де інженери працюють протягом усього життєвого циклу програми, від розробки та тестування до розгортання та операцій, і розвивають низку навичок, не обмежуючись однією функцією. Основна ідея DevOps полягає в тому, щоб створити культуру співпраці та комунікації між розробниками програмного забезпечення і операторами систем. Це полегшує розгортання, моніторинг, управління та підтримку ПЗ після релізу.

##### Основні принципи DevOps включають:

*Автоматизацію*: Використання інструментів автоматизації для зменшення ручної роботи та підвищення ефективності процесів розробки та експлуатації.

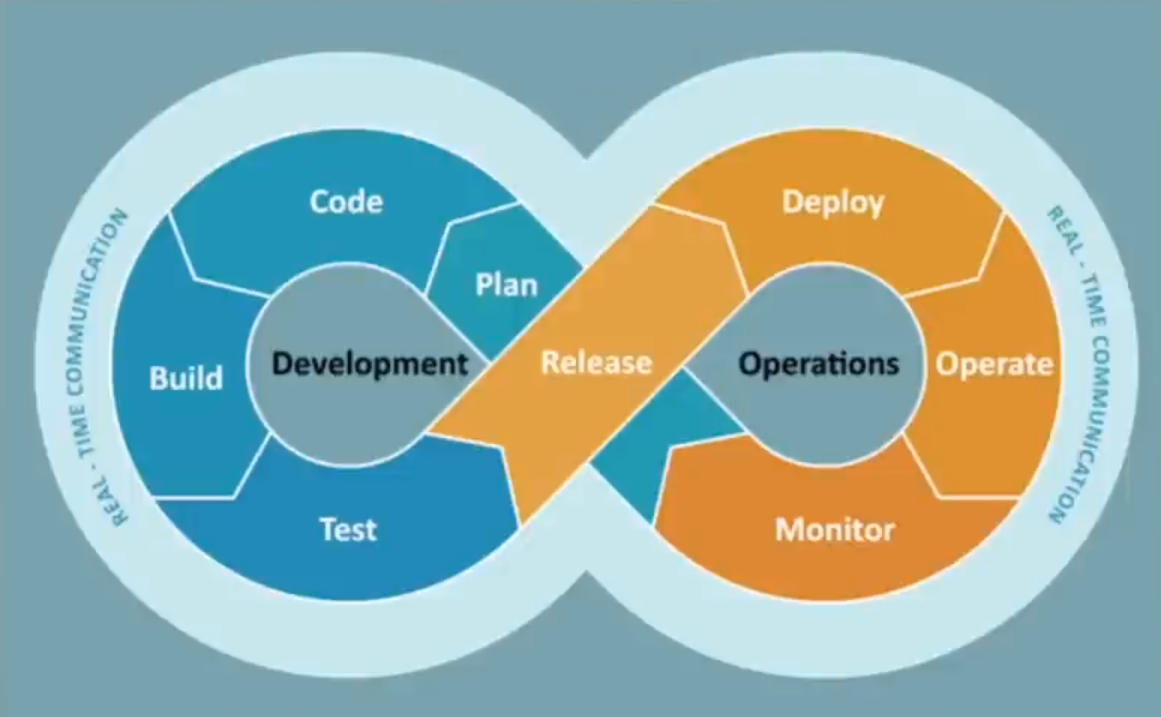
*Контроль версій*: Використання систем контролю версій, таких як Git, для збереження і відстеження змін у програмному коді та інфраструктурі.

*Часті релізи*: Часте релізування програмного забезпечення, щоб швидко впроваджувати нові функції та виправляти помилки.

*Безперервна інтеграція та неперервна доставка*(CI/CD), які автоматизують процес випуску програмного забезпечення від створення до розгортання.

*Інфраструктура як код* (IaC) допомагає керувати середовищем розробки, тестування та виробництва повторюваним і ефективнішим способом.

*Моніторинг та логування*: Забезпечення ефективного моніторингу та реєстрації подій у програмному забезпеченні та інфраструктурі для виявлення проблем та швидкого реагування на них.



**Переваги DevOps:**

*Висока швидкість роботи та доставки*

*Масштаб.* Автоматизація та узгодженість допомагають ефективно та з меншим ризиком керувати складними або мінливими системами. Використовуючи інфраструктуру як код і політику як код, ви можете визначити, а потім відстежувати відповідність у масштабі.

*Покращення співпраці.* В моделі DevOps команди розробки та експлуатації тісно співпрацюють, ділять багато обов’язків і поєднують робочі процеси. Це підвищує ефективність і економить час.

*Надійність і Безпека,* використовуючи автоматизовані політики відповідності, детальні елементи керування та методи керування конфігурацією.

## 2. Ви набираєте адресу будь-якого сайта у браузері, що після цього відбувається.

Коли ви набираєте адресу будь-якого сайту у браузері, відбувається послідовність дій, яка включає:

*Парсінг запиту.*Браузер спочатку аналізує введену вами адресу і отримує інформацію про протокол (наприклад, HTTP або HTTPS) та доменне ім'я сайту (наприклад, google.com).

*Резольвація доменного імені.* Браузер перевіряє DNS-сервери, щоб отримати IP-адресу, пов'язану з доменним іменем. Це дозволяє браузеру знайти сервер, на якому розташований сайт.

Резольвація доменного імені - це процес перетворення доменного імені (наприклад, example.com) на відповідну IP-адресу. Коли ви вводите доменне ім'я в браузері, браузер відправляє запит до DNS-сервера для отримання IP-адреси, пов'язаної з цим доменним іменем. DNS-сервери перевіряють свою базу даних і повертають IP-адресу браузеру. Браузер використовує отриману IP-адресу для встановлення з'єднання з веб-сайтом.

Коли користувач повторно відвідує HTTPS веб-сайт протягом визначеного періоду часу, браузер автоматично встановлює з'єднання HTTPS без перевірки DNS завдяки автоматично налаштованої політики HSTS (HTTPS Strict Transport Sequrity)

*Встановлення з'єднання.* Браузер встановлює з'єднання з сервером за допомогою протоколу TCP/IP. Це включає запит на встановлення з'єднання і обмін синхронізаційними повідомленнями:

а) Відправлення запиту на встановлення з'єднання (TCP handshake):

Браузер відправляє запит на встановлення з'єднання до веб-сайту, використовуючи IP-адресу, отриману під час резольвації доменного імені.

Запит містить вказівку браузера про те, який порт слід використовувати для з'єднання (наприклад, порт 80 для HTTP або порт 443 для HTTPS).

б) Встановлення з'єднання (TCP handshake):

Браузер і веб-сервер обмінюються спеціальними контрольними пакетами

Відбувається передача синхронізаційних (SYN) та підтвердних (ACK) пакетів між браузером і сервером.

в) Відправлення запиту HTTP:

Після успішного встановлення з'єднання TCP/IP, браузер може відправити запит HTTP для отримання вмісту веб-сторінки.

Запит містить метод (GET, POST, і т.д.), заголовки запиту і, за потреби, дані запиту.

г) Отримання відповіді від сервера:

Веб-сервер обробляє отриманий запит і генерує відповідь, яка містить заголовки відповіді та вміст веб-сторінки.

Відповідь передається через встановлене з'єднання TCP/IP до браузера.

д) Обробка відповіді браузером:

Браузер отримує відповідь і обробляє її, включаючи перевірку статусного коду (наприклад, 200 OK), зчитування заголовків відповіді і відображення вмісту сторінки.

е) Закриття з'єднання:

Після завершення передачі вмісту сторінки, браузер може закрити з'єднання TCP/IP, якщо він більше не потрібен.

Якщо браузер повинен завантажити додаткові ресурси (зображення, скрипти, стилі), він може відправити додаткові запити на сервер і повторити процес знову.

*Запит до сервера.* Браузер надсилає запит до сервера, що містить інформацію про запитану сторінку або ресурси (наприклад, HTML-сторінку, зображення, стилі CSS або скрипти JavaScript). Цей запит складається з кількох компонентів:

а) Метод запиту: Це тип запиту, який браузер надсилає до сервера. Найпоширенішими методами є GET і POST. GET використовується для отримання ресурсів з сервера, тоді як POST використовується для відправлення даних на сервер.

б) URL (Uniform Resource Locator): Це адреса ресурсу, який браузер запитує від сервера. URL складається з протоколу (наприклад, http або https), доменного імені (наприклад, example.com) і шляху до ресурсу на сервері (наприклад, /path/to/resource).

в) Заголовки запиту: Заголовки містять додаткову інформацію про запит, таку як типи даних, мову, кешування, cookies і багато іншого. Наприклад, заголовок "Accept-Language" може вказувати, яку мову веб-сторінки браузер віддає перевагу, а заголовок "User-Agent" містить інформацію про браузер і операційну систему користувача.

г) Тіло запиту (якщо присутнє): Деякі запити можуть мати тіло, яке містить додаткові дані, що надсилаються на сервер. Наприклад, при використанні методу POST, дані форми можуть бути включені в тілі запиту.

*Отримання відповіді від сервера.* Сервер обробляє запит і надсилає відповідь до браузера. Відповідь може містити HTML-код, зображення, CSS, JavaScript і інші ресурси.

*Обробка відповіді.* Браузер отримує відповідь і починає обробляти її. Він аналізує HTML-код, виконує JavaScript, завантажує інші ресурси, які можуть бути знайдені в HTML, такі як зображення, стилі CSS і шрифти.

*Рендерінг сторінки.* Браузер перетворює отриманий HTML-код, CSS і JavaScript в рендеровану сторінку, яку ви бачите на екрані. Це включає відображення тексту, зображень, стилів, розміщення елементів і взаємодію з користувачем.

*Завантаження додаткових ресурсів.* Якщо сторінка містить посилання на інші ресурси, такі як зображення, стилі CSS або скрипти JavaScript, браузер завантажує їх, щоб повністю відобразити сторінку.

**Основні методи HTTP запитів:**

GET: Використовується для отримання ресурсу з сервера. Клієнт надсилає запит до сервера і отримує відповідь, яка містить запитаний ресурс. GET-запити повинні бути ідемпотентними, тобто повторення одного й того ж GET-запиту не повинно мати впливу на сервер або його дані.

POST: Використовується для відправлення даних на сервер для обробки. Наприклад, при відправленні форми на веб-сторінці, дані форми надсилаються на сервер за допомогою POST-запиту. POST-запити не є ідемпотентними, тому кожен POST-запит створює новий ресурс або змінює існуючий.

PUT: Використовується для створення або заміни ресурсу на сервері. Клієнт надсилає дані, які повинні бути збережені на сервері за певним URL. Якщо ресурс існує, PUT-запит замінює його, а якщо він не існує, він створює новий ресурс.

DELETE: Використовується для видалення ресурсу на сервері за певним URL. Клієнт надсилає DELETE-запит і просить сервер видалити ресурс. Видалення ресурсу може бути частково або повністю залежно від реалізації сервера.

PATCH: Використовується для часткового оновлення ресурсу на сервері. Клієнт надсилає дані, які містять лише змінені або оновлені поля ресурсу, і сервер застосовує ці зміни до ресурсу.

**SSL (Secure Sockets Layer) та TLS (Transport Layer Security)** є протоколами шифрування, які забезпечують безпеку під час передачі даних через Інтернет. Основна різниця між ними полягає в версіях та покращеннях протоколу. Основні відмінності між SSL та TLS:

SSL був розроблений в 1990-х роках, тоді як TLS є його наступником і був випущений в 1999 році.

TLS є покращеною версією SSL і має більш сильні алгоритми шифрування та безпеки.

SSL використовує номери версій 2.0, 3.0, тоді як TLS використовує версії 1.0, 1.1, 1.2, 1.3.

TLS є більш безпечним і рекомендованим протоколом для захищеної передачі даних через Інтернет.

## 3. Як працює HTTPS?

**HTTP** (Hypertext Transfer Protocol) - це протокол передачі даних в інтернеті. Він працює на основі простого протоколу запит-відповідь та використовує текстові повідомлення (які можуть бути читані людиною і машинами) для комунікації між веб-браузерами (клієнтами) та веб-серверами. Ось як працює HTTP:

*Встановлення з'єднання:* Клієнт (веб-браузер) встановлює з'єднання з веб-сервером за допомогою TCP/IP і посилає запит до сервера, щоб отримати певну веб-сторінку або ресурс.

*Відправлення запиту:* Клієнт формує HTTP-запит, який містить метод (GET, POST, тощо), URL-адресу ресурсу, заголовки та дані запиту. Запит відправляється на сервер.

*Обробка запиту сервером:* Веб-сервер отримує HTTP-запит, перевіряє, який ресурс запрошено, і генерує відповідь на запит.

*Відправлення відповіді:* Веб-сервер формує HTTP-відповідь, яка містить статус-код (наприклад, 200 OK, 404 Not Found), заголовки та, за необхідності, дані відповіді. Відповідь відправляється назад до клієнта.

*Отримання відповіді клієнтом:* Клієнт отримує HTTP-відповідь від сервера. Він аналізує статус-код, заголовки та дані відповіді. Залежно від отриманої відповіді, клієнт може відображати сторінку, виконувати певні дії або інші дії відповідно до запиту.

Особливості HTTP:

*Запити та відповіді*: HTTP дозволяє клієнтським програмам надсилати запити на сервер для отримання різних ресурсів, а також отримувати відповіді від сервера.

*Ідемпотентність*: кожен запит HTTP вважається окремим, і сервер не пам'ятає попередні запити від клієнта.

*Текстовість*: У класичному вигляді HTTP використовує текстові команди для обміну даними між клієнтом та сервером.

*Використання порту 80*: HTTP використовує TCP-порт 80 для комунікації.

Протокол небезпечний: HTTP не забезпечує шифрування даних.

Для передачі конфіденційної інформації рекомендується використовувати **HTTPS**.

**HTTPS** (Hypertext Transfer Protocol Secure) - це протокол *безпечної* передачі даних в інтернеті. Він є розширенням протоколу HTTP і використовує шифрування для захисту конфіденційності та цілісності переданих даних. Ось як працює HTTPS:

*Шифрування даних:* При використанні HTTPS дані, які передаються між веб-браузером (клієнтом) і веб-сервером, шифруються. Це означає, що дані перетинають мережу у зашифрованому вигляді і не можуть бути прочитані зловмисниками, які можуть перехоплювати з'єднання.

*Сертифікати SSL/TLS:* Для забезпечення шифрування використовуються сертифікати SSL/TLS (Secure Sockets Layer/Transport Layer Security). Веб-сервер має свій власний сертифікат, який містить публічний ключ. Клієнт (веб-браузер) перевіряє цей сертифікат, щоб переконатися, що він отримує дані від правильного сервера і що з'єднання є безпечним.

*Установлення з'єднання:* При встановленні з'єднання між клієнтом і сервером відбувається процес, в якому вони обмінюються інформацією про підтримувані шифри і ключі шифрування. Після успішного встановлення з'єднання веб-браузер і сервер можуть безпечно обмінюватися даними.

*Захист від змін:* HTTPS також забезпечує цілісність даних. Це означає, що дані, які передаються між клієнтом і сервером, не можуть бути змінені під час передачі. Якщо дані будуть змінені, то це буде помічено, і з'єднання буде вважатися ненадійним.

## 4. Поясніть концепцію Infrastructure as a Code, для чого це потрібно і які проблеми вирішує?

**Infrastructure as a Code (IaaC)** - це концепція управління та надання ресурсів комп'ютерних центрів через машинно-читабельні файли визначення, а не за допомогою фізичної конфігурації апаратного забезпечення або інтерактивних інструментів конфігурації. За допомогою IaC, інфраструктура може бути автоматизовано створена, налаштована та керована шляхом використання файлів з кодом.

Основна ідея IaC полягає в тому, щоб розглядати інфраструктуру як код, що дозволяє використовувати ті самі принципи розробки програмного забезпечення для управління інфраструктурою. Ось деякі переваги та проблеми, які вирішує IaC:

**Переваги IaaC**:

Автоматизація: IaC дозволяє автоматизувати процеси створення, налаштування та управління інфраструктурою. Це зменшує ризик помилок та забезпечує швидку та надійну розгортку ресурсів.

Повторне використання: Кодова база IaC може бути повторно використана для створення інших інфраструктур.

Infrastructure as Code (IaaC) **вирішує проблеми**, пов'язані з інфраструктурою обчислювальних потужностей:

*Крихкість інфраструктури:* Традиційне управління інфраструктурою може бути крихким, оскільки кожна зміна виконується вручну і вимагає ручного налаштування. Це може призвести до помилок та несправностей. IaaC дозволяє визначати інфраструктуру у вигляді коду, що забезпечує автоматичну та надійну розгортку.

*Несумісність інфраструктури:* У компаніях часто використовуються різні платформи і рішення для інфраструктури, що може призвести до несумісності та складнощів у керуванні. IaC дозволяє визначати інфраструктуру універсальним мовою (кодом), що полегшує розгортку та управління незалежно від платформи.

*Масштабованість:* Ручне масштабування інфраструктури може бути складним та потребувати часу. IaaC дозволяє автоматично масштабувати ресурси на основі потреб, що забезпечує ефективне використання ресурсів та забезпечує гнучкість.

*Документація:* Традиційна інфраструктура часто не має документації або вона застаріла. IaC надає можливість документувати інфраструктуру як код, що дозволяє зберігати актуальну та зрозумілу документацію про інфраструктуру.

*Контроль версій ПЗ та контроль змін:* IaaC дозволяє контролювати версії інфраструктурного коду та здійснювати зміни систематично. Це полегшує відстеження змін, відкочування до попередніх версій або відновлення системи в разі проблем.

# Junior DevOps - Linux

## 5. Опишіть загальну архітектуру операційної системи.

Основні компоненти архітектури операційної системи:

*Ядро операційної системи*: Ядро є центральною частиною операційної системи. Воно виконує базові функції, такі як керування процесами, пам'яттю та пристроями, планування ресурсів та забезпечення безпеки.

*Завантажувач*: Завантажувач відповідає за процес завантаження операційної системи при включенні комп'ютера. Він завантажує ядро операційної системи в пам'ять та запускає його.

*Командний інтерпретатор*: Командний інтерпретатор або оболонка є інтерфейсом між користувачем та операційною системою. Він дозволяє користувачеві взаємодіяти з операційною системою шляхом введення команд.

*Драйвери пристроїв*: Драйвери пристроїв забезпечують взаємодію між операційною системою та апаратними пристроями, такими як принтери, монітори, клавіатури тощо. Вони дозволяють операційній системі керувати та використовувати ці пристрої.

*Графічний інтерфейс*: Графічний інтерфейс надає користувачеві зручний спосіб взаємодії з операційною системою за допомогою графічних об'єктів, таких як вікна, кнопки та меню.

LINUX Debian та RHEL є **монолітними OS**. Монолітне ядро (англ. Monolithic kernel) є класичною і поширеною архітектурою ядра операційних систем. Монолітні ядра надають багатий набір абстракцій та функцій, включаючи керування процесами, пам'яттю, пристроями, планування ресурсів та забезпечення безпеки. Це означає, що всі ці функції вбудовані безпосередньо в ядро операційної системи.

Ось кілька прикладів **не монолітних** операційних систем:

*Мікроядерна архітектура*: У мікроядерних операційних системах основні функції ядра обмежені до основних операцій, таких як керування міжпроцесними комунікаціями та керуванням пам'яттю. Решта функцій, такі як драйвери пристроїв та файлові системи, виконуються у відокремлених процесах-серверах, що спілкуються з ядром через мінімальний набір інтерфейсів.

*Екзоядерна архітектура*: В екзоядерних операційних системах значна частина функцій ядра виконується у відокремлених процесах-бібліотеках, які працюють у привілейованому режимі. Це дозволяє розширювати та модифікувати функціональність ядра без необхідності зміни самого ядра.

*Гіпервізори*(VMware vSphere, MS Hyper-V Server, Citrix hypervisor),або віртуалізаційні платформи дозволяють запускати багато віртуальних машин на одному фізичному сервері. Кожна віртуальна машина має свою віртуальну OS, яка може бути монолітною або іншого типу. Гіпервізори забезпечують ізоляцію та керування ресурсами між віртуальними середовищами.

*Мережеві OS*(Cisco IOS, Juniper Junos, Windows 10 Server, Windows NT, Open Network Linux, DENT, SONiC): Network Operating Systems зазвичай використовують розподілену архітектуру, де функції операційної системи розподілені на кілька вузлів мережі. Кожен вузол виконує певні функції, такі як керування мережевими пристроями, аутентифікація користувачів або зберігання даних.

## 6. Опишіть основне призначення операційної системи.

Операційна система (ОС) керує всіма ресурсами комп'ютерної системи і забезпечує взаємодію між апаратним та програмним забезпеченням. ОС є основною програмою, яка запускається при включенні комп'ютера, і вона виконує наступні основні функції:

*Управління апаратним забезпеченням:* ОС дозволяє взаємодіяти з апаратним забезпеченням комп'ютера, таким як процесор, пам'ять, пристрої введення-виведення тощо. Вона забезпечує доступ до ресурсів комп'ютера і керує їх використанням.

*Управління процесами:* ОС керує виконанням процесів (програм), розподіляючи ресурси комп'ютера між ними. Вона контролює запуск, зупинку, призупинення та призначення пріоритетів процесам.

*Управління пам'яттю:* ОС відповідає за керування доступом до пам'яті комп'ютера. Вона розподіляє доступну пам'ять між процесами і контролює її використання. ОС також відповідає за вирішення проблеми нестачі пам'яті шляхом використання вторинної пам'яті (такої як жорсткий диск) для збереження даних, які не вміщуються в оперативну пам'ять.

*Управління файловою системою:* ОС забезпечує доступ до файлів та каталогів на зберігання даних. Вона дозволяє створювати, видаляти, копіювати та переміщувати файли, а також забезпечує контроль доступу до них.

*Забезпечення інтерфейсу користувача:* ОС надає спосіб взаємодії з комп'ютером через графічний інтерфейс користувача (GUI) або командний рядок. Вона дозволяє користувачам запускати програми, налаштовувати систему та працювати з файлами.

*Забезпечення безпеки:* ОС відповідає за захист системи та даних користувачів від несанкціонованого доступу, вірусів та інших загроз. Вона надає механізми автентифікації, авторизації та шифрування для забезпечення безпеки інформації.

*Підтримка мережі:* ОС може забезпечувати функції для підключення до мережі, надання мережевих служб (наприклад, доступ до Інтернету) і керування мережевими пристроями.

## 7. Навіщо потрібні файлові системи? Які існують?

**Функції файлових системі як важливої частини ОС:**

*Організація даних*: Файлові системи організовують дані на зберігальних пристроях, таких як жорсткі диски, флеш-накопичувачі тощо. Вони дозволяють створювати, зберігати та впорядковувати файли та директорії для зручного доступу до них.

*Контроль доступу*: Файлові системи визначають, які користувачі та програми мають доступ до певних файлів. Це допомагає забезпечити безпеку даних та уникнути несанкціонованого доступу до конфіденційної інформації.

*Відновлення даних*: Файлові системи роблять можливим відновлення даних в разі їх втрати або пошкодження. Вони зберігають метадані, які дозволяють відновити файли та відновити їхню цілісність.

*Оптимізація роботи з файлами*: Файлові системи дозволяють швидкий доступ до файлів та оптимізують їх розміщення на зберігальних пристроях для підвищення продуктивності.

**Популярні файлові системи:**

*FAT (File Allocation Table): Є* однією з найпоширеніших файлових систем, яка використовується на різних носіях, таких як флеш-накопичувачі, SD-карти, тощо. FAT32 дозволяє створювати файли до 4Гб. Має відносно обмежені можливості управління правами доступу та безпекою, управління простором на носії.

Приклади: FAT12, FAT16, FAT32

*NTFS (New Technology File System):* Розроблена корпорацією Microsoft, NTFS є потужною та надійною файловою системою, яка підтримує розширені функції безпеки та керування правами доступу.

Приклади: Windows 10 використовує NTFS за замовчуванням для жорстких дисків.

*exFAT (Extended File Allocation Table): Б*ула розроблена компанією Microsoft для використання на флеш-накопичувачах та інших мобільних пристроях.

Приклади: Використовується на SD-картах, флеш-накопичувачах, зовнішніх жорстких дисках.

*HFS+ (Hierarchical File System Plus):* Розроблена компанією Apple для операційних систем Mac OS. HFS+ має розширені можливості для керування файлами та метаданими.

Приклади: Використовується на жорстких дисках та SSD-накопичувачах в пристроях Mac.

*EXT4 (Fourth Extended Filesystem): Є* однією з найпопулярніших файлових систем у середовищі Linux. Вона підтримує великі обсяги даних та високу надійність.

Приклади: Використовується у багатьох дистрибутивах Linux, таких як Ubuntu, Fedora, тощо.

*ZFS (Zettabyte File System):* є розподіленою файловою системою, розроблена і впроваджена Sun Microsystems. Розрахована на роботу з дуже великими обсягами даних, має високу надійність та цілісність даних (вбудовані механізми захисту даних, включаючи контрольні суми, копіювання з ліквідацією алокування та інш.). ZFS дозволяє створювати миттєві знімки (snapshots) файлової системи, щоб швидко відновлювати дані до попередніх станів а такоє для створення віртуальних копій файлової системи. ZFS має вбудовану підтримку компресії даних та шифрування. ZFS має підтримку кешування та тирінгу для оптимізації доступу до даних на різних рівнях.

## 8. У чому різниця між віртуалізацією і контейнеризацією?

Віртуалізація використовує віртуальні машини з окремими операційними системами, тоді як контейнеризація використовує контейнери з додатками та їх залежностями, які виконуються на спільній операційній системі.

*Віртуалізація* забезпечує можливість запускати декілька віртуальних машин (VM) на одному фізичному сервері.

Кожна віртуальна машина має свою власну операційну систему, яка віртуалізована та виконується незалежно від інших віртуальних машин на тому ж сервері.

Цей підхід вимагає більшого обсягу ресурсів (пам'яті, обчислювальної потужності) через віртуалізацію операційних систем.

*Контейнеризація* дозволяє ізолювати та запускати додатки та їх залежності в контейнерах.

Кожен контейнер використовує спільну операційну систему з господарською машиною, але ізольований від інших контейнерів.

Цей підхід дозволяє більш ефективно використовувати ресурси, оскільки не потрібно віртуалізувати операційні системи для кожного контейнера.

## 9. У чому переваги контейнерів?

*Легковагість*: Контейнери зазвичай потребують менше ресурсів, оскільки вони ділять ядро операційної системи з хост-системою, що дозволяє їм працювати більш ефективно.

*Швидкість розгортання*: Контейнери запускаються швидше, оскільки вони не вимагають повного завантаження операційної системи для кожного екземпляра.

*Портативність*: Контейнери можна легко переміщати між різними середовищами, оскільки вони містять усі необхідні залежності для виконання програм.

*Масштабованість*: Контейнери можна легко масштабувати вгору або вниз, оскільки вони використовуються для розподілення додатків на дрібні частини, які можуть бути керовані окремо.

*Ізоляція ресурсів*: Контейнери надають відокремлення ресурсів від хост-системи, що дозволяє їм працювати незалежно один від одного.

## 10. Яка файлова структура у Linux (UNIX) систем, що розташовані в / etc, / dev, / proc, / sys, / lib, / var (кілька директорій на вибір)?

Файлова структура в UNIX-подібних операційних системах, таких як Linux, має ієрархічну організацію, яка починається з кореневого каталогу ("/"). Основні каталоги (директорії):

/ : Кореневий каталог, він є початковим пунктом файлової системи.

/bin: Містить основні виконувані файли, необхідні для завантаження та функціонування системи.

/boot: Містить файли, необхідні для завантаження операційної системи.

/dev: Файли пристроїв, які представляють пристрої в системі.

/etc: Містить конфігураційні файли для системних процесів та програм.

/home: Каталоги користувачів.

/lib та /lib64: Бібліотеки, необхідні для роботи виконуваних файлів в /bin та /sbin.

/media: Точка монтування для зовнішніх носіїв, таких як USB-накопичувачі.

/mnt: Точка монтування для тимчасового монтажу файлових систем.

/opt: Додаткове програмне забезпечення.

/proc: Віртуальна файлова система, що містить інформацію про процеси та систему.

/root: Домашній каталог адміністратора (root).

/sbin: Виконувані файли для системного адміністрування.

/tmp: Тимчасові файли.

/usr: Додаткові програми та файли, які не є необхідними для старту або відновлення системи.

/var: Змінні дані, такі як файли журналів, тимчасові файли тощо.

Типова структура файлів в директорії /etc для Linux:

/etc/passwd: Файл, що містить інформацію про користувачів системи.

/etc/group: Файл, що містить інформацію про групи користувачів.

/etc/shadow: Файл, де зберігаються зашифровані паролі користувачів.

/etc/hostname: Файл, який містить ім'я хоста комп'ютера.

/etc/hosts: Файл, який містить список IP-адресів та відповідних імен хостів.

/etc/resolv.conf: Файл, що містить інформацію про DNS-сервери.

/etc/network/: Папка з конфігураційними файлами для мережевих параметрів.

/etc/apt/: Папка з конфігураційними файлами для системи керування пакунками APT.

/etc/ssh/: Папка з конфігураційними файлами для SSH-сервера та клієнта.

/etc/sudoers: Файл, що містить конфігурацію доступу користувачів до sudo.

/etc/fstab: Файл, що містить інформацію про файлові системи та точки монтування.

/etc/crontab: Файл, що містить розклад завдань для планувальника завдань cron.

/etc/profile: Файл, який містить налаштування профілю для всіх користувачів.

Типова структура файлів в директорії /var для Linux:

/var/log: Містить файли журналів, які містять інформацію про події, які сталися в системі.

/var/spool: Папка, в якій зберігаються файли тимчасового сховища для процесів, такі як черги друку або поштові скриньки.

/var/cache: Містить тимчасові файли, що генеруються програмами.

/var/tmp: Тимчасові файли, які зберігаються між різними сеансами роботи.

/var/lib: Містить змінні дані, такі як файли баз даних та інші дані, необхідні для роботи програм.

/var/www: Папка, в якій зазвичай зберігаються файли веб-сайтів.

/var/mail: Містить поштові скриньки користувачів.

/var/spool/mail: Містить тимчасові файли поштових скриньок користувачів.

## 11. Що таке Load Average?

*Load Average (середнє навантаження)* - це метрика, яка вимірює кількість завдань, які очікують в черзі або активно виконуються на процесорі в певний час. Зазвичай вимірюється у вигляді трьох значень, які відображають середнє навантаження протягом останніх 1, 5 та 15 хвилин.

Це показник, який використовується для оцінки того, наскільки завантажений системний ресурс (зазвичай процесор) у певний момент часу. Він не вимірює конкретне завдання чи процес, а просто вказує на загальне навантаження системи.

Якщо значення Load Average велике, це може свідчити про те, що система перевантажена, і може бути важко виконувати нові завдання. На відміну від CPU, яке вимірює процент використання процесора, Load Average враховує як активне, так і очікуюче навантаження.

Для однопроцесорної операційної системи Load Average можна розуміти наступним чином:

*Значення = 1.0*: Якщо Load Average дорівнює 1.0, це означає, що система повністю завантажена. Це означає, що в середньому є одне активне або очікуюче завдання, яке вимагає виконання.

*Значення < 1.0*: Значення менше 1.0 вказує на те, що середнє навантаження менше, ніж потужність системи, і система працює з комфортом.

*Значення > 1.0*: Значення більше 1.0 означає, що середнє навантаження перевищує потужність системи. Наприклад, якщо Load Average дорівнює 2.0, це означає, що в середньому система має подвійне навантаження відносно своєї потужності.

*Динаміка змін*: Важливо також враховувати динаміку змін Load Average. Наприклад, якщо значення зростає протягом тривалого часу, це може свідчити про зростання навантаження на систему.

*Load Average вважається високим*, якщо воно перевищує кількість ЦП на сервері. Наприклад, якщо кількість ЦП на нашому сервері становить лише 4, але середнє навантаження, яке ми бачимо, становить 5,4, ми маємо високе середнє навантаження.

## 12. У чому різниця між soft та hard symlink?

У UNIX-подібних операційних системах існують два типи символьних посилань: м'які (soft) та жорсткі (hard). Ось основні відмінності між ними:

*М'яке (soft) символьне посилання:*

Це посилання на інший файл або каталог у файловій системі.

Якщо оригінальний файл або каталог переміщується або видаляється, м'яке посилання втрачає свою дійсність і вказує на неіснуючий об'єкт.

Створене за допомогою команди ln -s.

Відображається як файл зі звичайною іконкою в більшості графічних файлових менеджерів.

*Жорстке (hard) символьне посилання:*

Це посилання на інший файл або каталог у файловій системі.

Якщо оригінальний файл або каталог переміщується або видаляється, жорстке посилання все ще вказує на початковий об'єкт, оскільки воно посилається на **inode** файлу.

Створене за допомогою команди ln.

Не може посилатися на каталоги та файли на різних файлових системах.

Відображається як окремий файл з власним **inode** в файловій системі.

Ось деякі ключові аспекти **inode** файлів:

*Inode зберігає метадані про файл*, такі як дата та час створення, дата та час останньої модифікації, розмір файлу, власник, дозволи доступу та посилання на блоки даних.

*Кожен inode має унікальний ідентифікатор* всередині файлової системи, що дозволяє оперувати файлами та каталогами шляхом посилання на їхні inode.

*Inode містить посилання на блоки даних*, де зберігається вміст файлу. У випадку великих файлів або файлів, що займають більше місця, ніж може вмістити один блок даних, inode містить посилання на додаткові блоки даних.

*Жорсткі символьні посилання в UNIX* використовують inode файлу для створення посилання на оригінальний файл.

## 13. Як працюють file permissions, навіщо директорії права виконання (+x)?

У операційній системі Linux, такій як Debian, робота file permissions базується на концепції "власник-група-інші" (owner-group-others) та "читання-запис-виконання" (read-write-execute). Ось як вони працюють:

*Власник, група та інші*: Кожен файл чи каталог має власника (owner), групу (group) та інших користувачів (others), які можуть бути визначені в системі.

*Читання, запис та виконання*: Для кожного з цих трьох типів користувачів (власник, група, інші) можуть бути встановлені дозволи на читання (read), запис (write) та виконання (execute).

*Належність до груп*: Кожен користувач в Linux може бути членом однієї або декількох груп. Файл може мати групові дозволи, які застосовуються до всіх користувачів, які належать до цієї групи.

*Налаштування дозволів*: Для зміни дозволів на файл або каталог використовуються команди chmod (change mode) та chown (change owner). Ці команди дозволяють встановлювати, змінювати та видаляти дозволи на файли та каталоги.

*Вплив на безпеку та доступ*: Дозволи на файли впливають на безпеку системи, оскільки вони визначають, хто має доступ до конкретних файлів та які операції можуть бути виконані над ними.

У UNIX-подібних операційних системах, включаючи Linux, директорії потребують права виконання (+x) з кількох причин:

*Доступ до вмісту*: Право виконання для директорії дає користувачам можливість звертатися до її вмісту. Без цього права користувачі не зможуть переглядати чи працювати з файлами, розташованими всередині директорії.

*Навігація*: Право виконання дозволяє користувачам навігувати в директоріях, використовуючи команди, такі як cd (change directory). Без цього права користувачі не змогли б перейти всередину директорії.

*Виконання команд*: Для виконання деяких операцій, таких як видалення, перейменування чи створення файлів всередині директорії, користувачеві потрібно мати право виконання на цій директорії.

У Linux структура цифрового коду file permissions представлена трьома числами, які відповідають за права доступу для власника, групи та інших користувачів:

Власник (Owner): Дозволи власника файлу.

Група (Group): Дозволи групи, до якої належить файл.

Інші користувачі (Others): Дозволи для всіх інших користувачів.

Кожне з цих чисел складається з трьох бітів, які представляють дозволи на читання, запис та виконання відповідно. Ось як ця структура представляється:

Читання (read) = 4

Запис (write) = 2

Виконання (execute) = 1

Таким чином, комбінації цих чисел утворюють різні комбінації дозволів. Наприклад:

Читання та запис відкриті для власника, а інші користувачі мають лише читання: 644

Повні дозволи для власника, а інші користувачі мають лише читання: 744

Повні дозволи для власника, групи та інших користувачів: 777

## 14. Що таке zombie process?

*Zombie-процес* - це термін, який використовується в UNIX-подібних операційних системах для позначення процесу, який завершив свою роботу, але ще не був повністю видалений з системи. Ось деякі їхні ключові риси:

*Завершення роботи*: Коли процес завершує свою роботу, він надсилає сигнал PID батьківському процесу, що вказує на його завершення. Батьківський процес повинен обробити цей сигнал та видалити ресурси, пов'язані з завершеним процесом.

*Загублені батьківські процеси*: Якщо батьківський процес не обробив сигнал про завершення дитячого процесу, останній може перетворитися на zombie-процес.

*Метафора*: Термін "zombie" використовується в аналогії з нежитьми, оскільки ці процеси не виконують жодних корисних дій, але все ще займають обмежені ресурси системи.

*Не займають ресурси процесора або пам'яті*: Хоча zombie-процес є присутнім у списку процесів, він не витрачає ресурси процесора або пам'яті, оскільки фактично не виконує жодної дії.

*Видалення з системи*: Щоб видалити zombie-процес із системи, його батьківський процес повинен обробити сигнал про завершення та звільнити ресурси, пов'язані з цим процесом.

Загалом, zombie-процеси є небажаним явищем, і вони можуть виникати в результаті неправильного керування процесами батьківськими процесами. Вони не завдають прямої шкоди системі, проте їх наявність може свідчити про проблеми у керуванні процесами.

## 15. За допомогою чого можна зібрати інформацію про поточний стан процесора, пам’яті, диска, мережі?

Є кілька стандартних утиліт Linux, які можна використовувати для збору інформації про поточний стан процесора, пам'яті, диска та мережі. Ось кілька з них:

Для процесора:

top: Відображає загальну інформацію про систему, включаючи використання процесора, пам'яті та інші системні ресурси.

htop: Покращена версія команди top, яка надає інтерактивне відображення використання ресурсів, включаючи процесор.

Для пам'яті:

free: Показує вільну та використану пам'ять, а також пам'ять, що використовується для буферів та кешування.

vmstat: Надає інформацію про віртуальну пам'ять, включаючи статистику процесора та пам'яті.

Для диска:

df: Відображає інформацію про вільне місце на файлових системах.

iostat: Надає статистику введення/виведення, включаючи використання диска.

Для мережі:

ifconfig: Показує інформацію про мережеві інтерфейси.

netstat: Надає інформацію про мережеві з'єднання, маршрутизацію та інші мережеві параметри.

## 16. Що таке swappiness?

Swappiness - це параметр ядра Linux, який контролює частоту використання файлу підкачки (swap file). Файл підкачки використовується для збереження неактивних сторінок пам'яті на жорсткому диску, коли фізична пам'ять системи стає недостатньою.

*Значення swappiness може бути від 0 до 100*. Чим більше значення swappiness, тим більша ймовірність, що система буде використовувати файл підкачки. Значення 0 означає, що система буде намагатися якомога менше використовувати файл підкачки, а значення 100 означає, що система буде активно використовувати файл підкачки.

*Зміна значення swappiness може впливати на продуктивність системи*. Наприклад, якщо ви маєте достатньо фізичної пам'яті і не хочете, щоб система часто використовувала файл підкачки, ви можете зменшити значення swappiness. З іншого боку, якщо у вас обмежена фізична пам'ять і ви хочете максимально використовувати файл підкачки, ви можете збільшити значення swappiness.

Щоб змінити значення swappiness в Linux, ви можете використовувати команду sysctl або змінити значення в файлі /proc/sys/vm/swappiness. Наприклад, для встановлення значення swappiness на 10 ви можете виконати команду:

$ sudo sysctl vm.swappiness=10

## 17. Як подивитися вільне місце на диску в Linux Debian?

Щоб подивитися вільне місце на диску в операційній системі Linux Debian, ви можете скористатися командою df (скорочення від "disk free"). Вона показує інформацію про вільне місце на всіх доступних файлових системах.

$ df -h /opt

Filesystem Size Used Avail Use% Mounted on

/dev/sda4 705G 140G 530G 21% /opt

df відображає обсяг дискового простору, доступного у файловій системі, що містить кожен аргумент імені файлу. Якщо ім’я файлу не вказано, буде показано доступний простір у всіх змонтованих файлових системах. Дисковий простір за замовчуванням відображається в блоках по 1 КБ, якщо не встановлено змінну середовища POSIXLY\_CORRECT, у цьому випадку використовуються блоки розміром 512 байт.

Якщо аргумент є абсолютним ім’ям файлу вузла дискового пристрою, що містить підключену файлову систему, df показує простір, доступний у цій файловій системі, а не у файловій системі, що містить вузол пристрою. В Ubuntu команда df не може показати простір, доступний у немонтованих файлових системах, оскільки для більшості типів систем це вимагає дуже глибоких знань про структуру файлової системи.

## 18. Що таке inode?

**Inode** (інод) - це структура даних в операційних системах UNIX, яка описує об'єкт файлової системи, такий як файл або каталог. Кожен файл або каталог в UNIX має свій власний inode, який містить метадані про цей об'єкт.

Основні атрибути, які зберігаються в inode, включають:

* Розмір файлу або каталогу.
* Права доступу до файлу (читання, запису, виконання) для різних користувачів.
* Власник файлу або каталогу.
* Група, до якої належить файл або каталог.
* Дату та час створення, зміни та останнього доступу до файлу.
* Кількість посилань на файл або каталог.
* Вказівник на блоки даних, які містять фактичний вміст файлу.

**Inode** є важливою частиною файлової системи UNIX, оскільки він дозволяє системі відстежувати та керувати файлами та каталогами. Кожен файл або каталог може мати свій унікальний номер inode, за допомогою якого система може швидко знаходити та обробляти ці об'єкти.

Ось деякі ключові аспекти **inode** файлів:

*Inode зберігає метадані про файл*, такі як дата та час створення, дата та час останньої модифікації, розмір файлу, власник, дозволи доступу та посилання на блоки даних.

*Кожен inode має унікальний ідентифікатор* всередині файлової системи, що дозволяє оперувати файлами та каталогами шляхом посилання на їхні inode.

*Inode містить посилання на блоки даних*, де зберігається вміст файлу. У випадку великих файлів або файлів, що займають більше місця, ніж може вмістити один блок даних, inode містить посилання на додаткові блоки даних.

*Жорсткі символьні посилання в UNIX* використовують inode файлу для створення посилання на оригінальний файл.

## 19. Розкажіть поетапно процес завантаження Linux від моменту ввімкнення живлення комп’ютера.

Процес завантаження Linux від моменту ввімкнення живлення комп'ютера може бути розділений на кілька етапів:

*BIOS (Basic Input/Output System)*: Після ввімкнення живлення комп'ютера, першим етапом є робота BIOS. BIOS виконує початковие самотестування (POST - Power-On Self Test), щоб перевірити апаратне забезпечення комп'ютера. Після успішного завершення POST, BIOS шукає завантажувальну операційну систему.

*Завантажувач системи (Boot Loader)*: Після завершення POST, BIOS передає управління завантажувачу системи (boot loader). Завантажувач системи, такий як GRUB (Grand Unified Bootloader), syslinux або LILO (Linux Loader), знаходиться на початковому секторі жорсткого диска або на іншому завантажувальному пристрої. Завантажувач системи дозволяє вибрати операційну систему для завантаження, включаючи Linux.

*Завантаження ядра*: Після вибору операційної системи, завантажувач системи зчитує ядро операційної системи з відповідного місця зберігання (наприклад, жорсткого диска) і завантажує його в пам'ять комп'ютера. Ядро Linux є основною частиною операційної системи і відповідає за керування ресурсами комп'ютера та виконання програм.

*Ініціалізація системи*: Після завантаження ядра, відбувається ініціалізація системи. Цей етап включає налаштування апаратного забезпечення, завантаження необхідних драйверів та запуск системних служб. Ініціалізація системи підготовлює операційну систему до роботи з користувачем.

*Завантаження середовища робочого столу або командного рядка*: Після ініціалізації системи, Linux завантажує середовище робочого столу (наприклад, GNOME, KDE) або командний рядок (термінал). Це залежить від налаштувань системи та вибору користувача.

*Вхід в систему*: Останній етап - вхід в систему. Користувач вводить свої облікові дані (логін та пароль) для аутентифікації. Після успішного входу в систему, користувач отримує доступ до робочого столу або командного рядка і може почати використовувати Linux.

Це загальний опис процесу завантаження Linux. Конкретні деталі можуть варіюватися залежно від конфігурації системи та використовуваної версії Linux.

## 20. Що станеться під час виконання команд:

1. cat file1 > file2

2. cat file1 >> file2

Is used to copy the contents of file1 to file2 in a Unix-like operating system. If file2 already exists, it will be overwritten with the content of file1.

To append the content of file1 to file2 rather than overwriting it, you can use >> instead of >.

## 21. У чому різниця між Ctrl+C та Ctrl+Z?

У командних оболонках для операційних систем Linux, є різниця між Ctrl+C та Ctrl+Z:

Комбінація клавіш Ctrl+C використовується для переривання (завершення) виконання поточного процесу. Коли ви натискаєте Ctrl+C, сигнал SIGINT (інтерпретація переривання) надсилається до процесу, що виконується. Це призводить до того, що процес завершується і повертає управління командній оболонці. Ctrl+C корисно в ситуаціях, коли потрібно негайно зупинити виконання команди або програми.

Комбінація клавіш Ctrl+Z використовується для призупинення (зупинки) виконання поточного процесу. Коли ви натискаєте Ctrl+Z, сигнал SIGTSTP (призупинення) надсилається до процесу. Це призводить до тимчасової зупинки процесу і повернення управління командній оболонці. Процес можна відновити пізніше за допомогою команди fg (foreground) або bg (background). Ctrl+Z корисно, коли потрібно тимчасово призупинити виконання команди або програми, але залишити її в активному стані для подальшого використання.

Отже, основна різниця між Ctrl+C та Ctrl+Z полягає в тому, що Ctrl+C завершує процес, а Ctrl+Z призупиняє його тимчасово.

## 22. Як перенаправити одночасно stderr та stdin?

Для перенаправлення stderr та stdin одночасно в Bash на Linux можна скористатися символом "<" для stdin та "2>" для stderr. Наприклад:

command 2>&1 < file.txt

У цій команді:

"command" - це команда, яку ви хочете виконати.

"2>&1" - цей фрагмент перенаправляє stderr (2) до того ж потоку, що й stdin (1).

"file.txt" - вказує, що stdin повинен бути перенаправлений з файлу file.txt.

Таким чином, команда виконається з використанням вмісту файлу file.txt як вхідних даних (stdin), а будь-які повідомлення про помилки (stderr) також будуть перенаправлені до цього ж місця, куди йдуть stdin.

## 23. Як вбити процес? Які є типи сигналів?

За допомогою ідентифікатора процесу (PID): Ви можете використати команду "kill" разом з ідентифікатором процесу (PID) для завершення процесу. Наприклад, щоб завершити процес з PID 12345, ви можете ввести команду:

$ kill 12345

За допомогою сигналу: Ви також можете вказати конкретний сигнал, який ви хочете надіслати процесу. Наприклад, для негайного завершення процесу можна використати сигнал SIGKILL, що відповідає номеру сигналу 9:

$ kill -9 12345

За допомогою імені процесу: Ви також можете використати команду "pkill" для завершення процесу за його іменем. Наприклад, якщо треба завершити процес з ім'ям "firefox", можна використати команду:

$ pkill firefox

У середовищі Linux існує кілька типів сигналів, які використовуються для комунікації між процесами та для управління їхньою поведінкою. Ось декілька типів сигналів, які можна зустріти в Linux:

*SIGTERM*: Цей сигнал відправляється для запиту про зупинку процесу. Процес може обробити цей сигнал та завершити роботу чисто, звільнивши ресурси.

*SIGKILL*: Цей сигнал негайно завершує процес. Він не може бути перехоплений або заблокований процесом, тому варто використовувати його обережно, оскільки він може призвести до втрати даних або некоректної роботи системи.

*SIGHUP*: Цей сигнал зазвичай використовується для перезавантаження процесу або програми, якщо він підтримує перезавантаження.

*SIGINT*: Цей сигнал генерується при натисканні комбінації клавіш Ctrl+C у терміналі, що зазвичай використовується для відправлення сигналу процесові зупинити свою роботу.

*SIGSTOP та SIGCONT*: SIGSTOP призупиняє виконання процесу, тоді як SIGCONT відновлює його виконання.

Це лише декілька прикладів типів сигналів в Linux. Існують ще інші типи

## 24. Що робить команда grep?

Команда grep у Bash використовується для пошуку тексту у вмісті файлів або виводу інших команд. Вона широко використовується для фільтрації виводу за допомогою регулярних виразів. Основне призначення команди grep - знаходження рядків, що відповідають певному шаблону у вхідних даних. Ось кілька прикладів використання команди grep:

*Пошук у файлі*: Ви можете використати команду grep для пошуку певного тексту у вмісті файлу. Наприклад, якщо ви хочете знайти всі рядки у файлі "file.txt", які містять слово "pattern", ви можете ввести:

$ grep "pattern" file.txt

*Пошук у виводі команди*: Крім того, можна використати команду grep для фільтрації виводу інших команд. Наприклад, якщо треба знайти всі рядки у виводі команди "ls", які містять слово "file:

$ ls -l | grep "file"

Використання регулярних виразів: Команда grep також підтримує використання регулярних виразів для більш гнучкого пошуку. Наприклад, ви можете використати регулярний вираз для пошуку всіх рядків, які починаються з певної послідовності символів:

$ grep "^pattern" file.txt

Також для роботи з регулярними виразами э команда egrep, яка є ще більш потужним інструментом для пошуку та фільтрації тексту в Linux-системах.

## 25. Що таке скрипт bash?

Скрипт у середовищі Linux, написаний для виконання у оболонці (shell) Bash, є текстовим файлом, який містить послідовність команд, які можуть бути виконані операційною системою. Скрипти Bash використовуються для автоматизації виконання завдань та обробки даних у середовищі командного рядка.

Ось кілька ключових аспектів щодо скриптів Linux Bash:

*Розширення файлу*: Зазвичай скрипти Bash мають розширення ".sh", але це не обов'язково. Якщо файл має відповідні права доступу для виконання, його можна виконати безпосередньо з командного рядка. Google codestyle наполягає на відсутності розширення.

*Синтаксис*: Скрипти Bash пишуться з використанням синтаксису, специфічного для оболонки Bash, включаючи умовні конструкції, цикли, функції та інші вбудовані команди.

*Автоматизація*: Скрипти Bash часто використовуються для автоматизації рутинних завдань, таких як резервне копіювання даних, обробка файлів, налаштування середовища тощо.

*Виконання команд Linux*: У скрипті можна використовувати будь-які команди, які можна виконати в командному рядку, включаючи виклики інших програм та обробку результатів їх роботи.

*Передача аргументів*: Скрипти можуть приймати аргументи від користувача або від інших скриптів, що дозволяє їх параметризацію та використання в різних контекстах.

Створення та використання скриптів Bash є потужним засобом для автоматизації процесів та виконання різноманітних завдань у середовищі Linux.

## 26. Які типи змінних використовують у bash?

Ось деякі типи змінних, які можна використовувати в bash:

*Рядкові змінні:* У bash рядкові змінні використовуються для зберігання текстових даних:

$ name="John"

*Числові змінні:* Числові змінні використовуються для зберігання числових значень:

$ count=10

*Масиви:* В bash також можна створювати масиви, які зберігають послідовності значень:

$ numbers=(1 2 3 4 5)

*Змінні середовища:* Змінні середовища використовуються для зберігання інформації про середовище виконання скрипту:

$ echo $HOME

*Спеціальні змінні:* В bash є також спеціальні змінні, такі як $0 (ім'я виконуваного файлу), $1, $2 (параметри командного рядка) та інші.

## 27.Що виведуть команди:

1. echo ${hostname};

Якщо змінна hostname не була попередньо визначена або не містить значення, то команда виведе порожній рядок.

Фігурні дужки у ${hostname} вказують вивести змінну з ім'ям hostname, а не змінну середовища $HOSTNAME.

2. echo $(hostname);

Виведе мережеве ім’я локального хоста. Наприклад Me-xUBUNTU, тобто назву хоста (ім'я комп'ютера), яке зазвичай зберігається в змінній середовища $HOSTNAME.

# Junior DevOps - Networks

## 28. Що таке модель OSI, TCP/IP?

Модель OSI (Open Systems Interconnection) та модель TCP/IP (Transmission Control Protocol/Internet Protocol) - дві різні концепції, що використовуються для організації та стандартизації комунікаційних систем:

**Модель OSI (Open Systems Interconnection)**

Є теоретичною концепцією, що складається з сімох рівнів, кожен з яких визначає окремі функції мережі. Основні рівні моделі OSI включають:

* Physical Layer (Фізичний рівень): Визначає електричні, оптичні та механічні властивості передавання даних через мережу.
* Data Link Layer (Рівень з'єднання даних): Відповідає за надання надійного з'єднання між двома вузлами мережі.
* Network Layer (Мережевий рівень): Відповідає за маршрутизацію даних через мережу.
* Transport Layer (Транспортний рівень): Забезпечує надійність та контроль цілісності даних під час передавання.
* Session Layer (Рівень сеансів): Визначає та керує сеансами зв'язку між пристроями.
* Presentation Layer (Рівень представлення): Відповідає за перетворення даних у формат, зрозумілий для приймача.
* Application Layer (Рівень застосунків): Забезпечує можливість взаємодії з програмами та мережевими службами.

**Модель TCP/IP**

Це зразок мережевої архітектури, що включає чотири рівні, які відрізняються від моделі OSI:

* Network Interface Layer (Рівень мережевого інтерфейсу): Отримує та відправляє дані через фізичні мережеві пристрої.
* Internet Layer (Інтернет-рівень): Відповідає за маршрутизацію даних через мережу.
* Transport Layer (Транспортний рівень): Забезпечує надійність та контроль цілісності даних під час передавання.
* Application Layer (Рівень застосунків): Забезпечує можливість взаємодії з програмами та мережевими службами.

Основною відмінністю між цими двома моделями є кількість та призначення рівнів, а також спосіб, якими вони організовані. Модель OSI має більше рівнів і більш деталізовану структуру, тоді як модель TCP/IP є більш простою та широко вживаною.

У практичному застосуванні, протоколи TCP/IP використовуються для побудови Інтернету, тоді як модель OSI використовується тільки для опису та аналізу мережевих протоколів.

## 29. Для чого потрібні network masks?

Мережеві маски (network masks) використовуються для ідентифікації частини IP-адреси, яка відділяє мережі та підмережі. Ось докладніше про призначення network masks:

*Розділення IP-адреси на мережу та хоста*: Мережева маска допомагає визначити, яка частина IP-адреси відноситься до мережі, а яка - до конкретного пристрою (хоста) в цій мережі.

*Ідентифікація мережі та підмережі*: Використання мережевої маски дозволяє визначити, до якої мережі або підмережі належить певна IP-адреса.

*Розподіл IP-адрес*: Мережеві маски використовуються для розподілу доступних IP-адрес на мережі та підмережі, забезпечуючи ефективне використання адресного простору.

*Маршрутизація*: Мережеві маски використовуються для визначення шляхів маршрутизації в комп'ютерних мережах, дозволяючи маршрутизаторам правильно направляти пакети даних.

*Підтримка підмереж*: Застосування мережевих масок дозволяє створювати підмережі, що сприяє більш ефективному управлінню IP-адресами та ресурсами мережі.

IPv4 Range - 0.0.0.0 - 255.255.255.255

(00000000.00000000.00000000.00000000 - 11111111.11111111.11111111.11111111)

*Public and Private IP division*

**- Private IP => for local network design**

**Class A** => 10.0.0.0 - 10.255.255.255 (total up to 256^3-2 = 16777214 IPv4)

10.0.0.0 - reserved for Network IP

10.0.0.1 - the first usable IP

...

10.255.255.254 - the last usable IP

10.255.255.255 - reserved for broadcast

**Class B** => 172.16.0.0 - 172.31.255.255a (total up to 256x256-2 = 65534 IPv4)

172.16.0.0 - reserved for Network IP

172.16.0.1 - the first usable IP

...

172.16.0.255

172.16.1.0

172.16.1.1

...

172.16.255.254 - the last usable IP

172.16.255.255 - reserved for broadcast

**Class C** => 192.168.0.0 - 192.168.255.255 (total up to 254 IPv4)

192.168.0.0 - reserved for Network IP

192.168.0.1 - the first usable IP

...

192.168.0.254 - the last usable IP

192.168.0.255 - reserved for broadcast

There are also Class D and Class E but we'll not use them because they are used for research and multicasting. Each class match different requirements.

**- Public IP => open Internet 54.86.23.90**

Nets divided with subnet masks. Subnet mask will decide on the network that an IP address range starts at what IP and ends with what IP.

*CIDR notation - Classes Internet Domain Routing*

255.0.0.0 - a subnet mask representation for OS

11111111.00000000.00000000.00000000 - a binary subnet mask representation

/8 - CIDR subnet mask representation

255.255.0.0 - a subnet mask representation for OS

11111111.11111111.00000000.000000000 - a binary subnet mask representation

/16 - CIDR subnet mask representation

255.255.255.0 - a subnet mask representation for OS

11111111.11111111.11111111.000000000 - a binary subnet mask representation

/24 - CIDR subnet mask representation

So a private network with a range 172.20.0.0/16 can be divided on 256 subnets:

- 172.20.0.0/24

- 172.20.1.0/24

...

- 172.20.255.0/24

'Wildcards' are opposite to subnets masks! (шаблони — протилежність маскам підмереж)

There are a lot of subnet calculators in the Internet, for example:

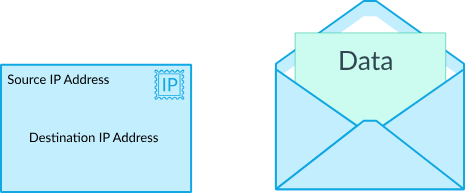
# https://jodies.de/ipcalc

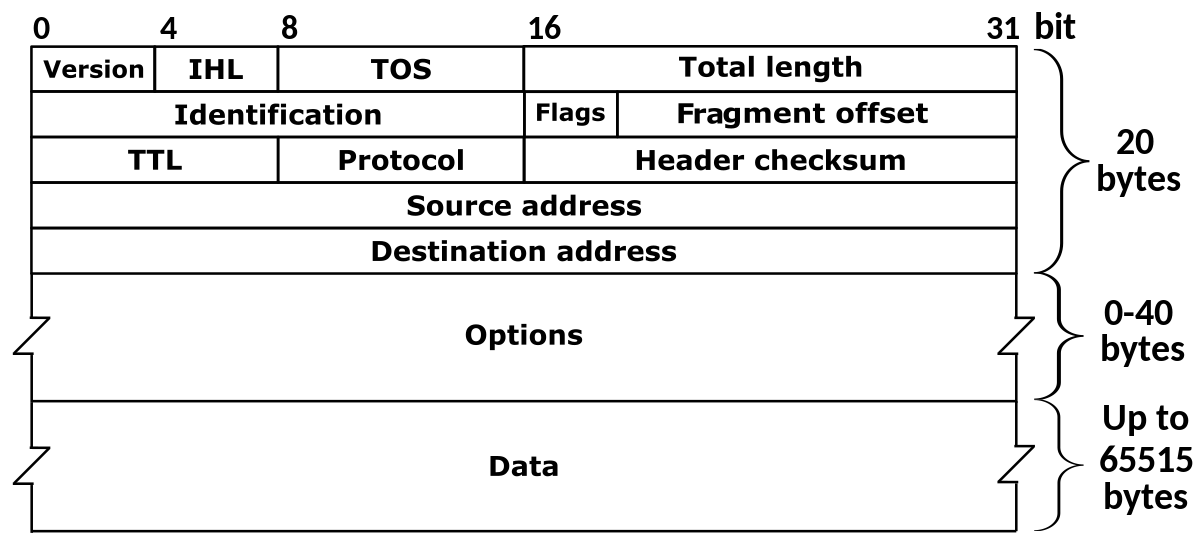
## 30. Структура IP-пакета. З чого складається? Що таке фрагментація IP-пакета та чому вона відбувається?

У більшості комп’ютерних мереж дані розбиваються на маленькі блоки, які називаються пакетами, і пересилаються між пристроями. IP-пакет — це одиниця даних у мережі, яка містить інформацію про адреси джерела та призначення та іншу керуючу інформацію, необхідну для транспортування пакета через мережу.

Процес включає маршрутизатор, який з’єднує локальну мережу з Інтернетом, отримує IP-пакет і зчитує адресу призначення. Потім маршрутизатор визначає наступний пункт призначення IP-пакету на основі свого списку доступних маршрутів, широко відомого як таблиця маршрутизації. Таким чином IP-пакет переміщується від одного маршрутизатора до іншого, поки не досягне місця призначення.

Ви можете розглядати IP-пакети як поштові листи: заголовок — це конверт із усією інформацією про маршрутизацію, яка потрібна поштовому відділенню, а корисне навантаження — це лист, який читає лише одержувач.

Кожен IP-пакет містить як заголовок (20 або 24 байти), так і дані (змінної довжини). Заголовок містить IP-адреси джерела та призначення, а також інші поля, які допомагають маршрутизувати пакет. Дані — це фактичний вміст, наприклад рядок літер або частина веб-сторінки.

**Version**: This field indicates the IP protocol version, such as IPv4 or IPv6, which helps determine how the packet should be routed.

**Total length**: Specifies the packet's total length of the data payload (the message) and the header.

Protocol: Determines the high-level protocol, like TCP, UDP, etc., used in the data payload.

**Time to Live (TTL)**: Used to prevent packets from circulating indefinitely, which could cause network congestion. Every time a packet gets forwarded, the router decreases the TTL by one, and when the TTL becomes zero, the packet gets discarded, helping to avoid congestion.

**Source IP address**: Specifies the IP address of the device that sent the packet. It’s used to reply to the sender.

**Destination IP address**: Identifies the IP address of the intended recipient of the packet. Routers use it to determine where to forward the packet to reach its final destination.

**Переваги використання IP-пакетів**

*Покращена продуктивність мережі*: IP-пакети містять IP-адреси джерела та призначення даних, що надсилаються, що забезпечує ефективну маршрутизацію мережевого трафіку. IP-адреси відправника та одержувача даних дозволяють маршрутизаторам швидко й точно направляти трафік до потрібного пункту призначення.

*Простіше масштабування*: ви можете швидко масштабувати служби, видаляючи або додаючи репліки за допомогою IP-пакетів. Оскільки кожній службі призначається віртуальна IP-адреса, ідентифікувати конкретний екземпляр служби та спілкуватися з ними відносно легко.

*Балансування навантаження*: IP-пакети балансувати навантаження на вузли мережі. Балансування навантаження допомагає розподіляти трафік між кількома копіями служби, покращуючи продуктивність і доступність програми.

*Краща спостережуваність*: оскільки IP-пакети дозволяють налагодити мережевий зв’язок та моніторинг. Крім того, кращий моніторинг і відстеження призведуть до кращої спостережливості, що допоможе швидко виявляти та вирішувати проблеми.

Фрагментація IP-пакета - це процес розбиття великого IP-пакета на менші фрагменти для передачі через мережу, яка не може обробити цілий пакет через обмеження щодо максимального розміру пакета. Фрагментація відбувається з наступних причин:

*Максимальний розмір пакета (MTU)*: Кожна мережа має свій максимальний розмір передаваного пакета, відомий як MTU. Якщо розмір IP-пакета перевищує MTU мережі, через яку він повинен бути переданий, то пакет буде розділено на менші фрагменти.

*Передача через мережі з різними MTU*: При переміщенні через різні мережі IP-пакети можуть зіштовхуватися з різними MTU, що також може призвести до фрагментації.

Коли пакет фрагментується, кожен фрагмент має свій власний заголовок IP, а приймаючий кінцевий пристрій повинен зібрати ці фрагменти знову в оригінальний пакет. Фрагментація може вплинути на продуктивність мережі, тому що вона збільшує навантаження на мережеве обладнання.

## 31. Що таке колізія в мережі TCP/IP? Чому виникає колізія в мережі TCP/IP?

Колізія в мережі TCP/IP відбувається, коли два або більше пристроїв намагаються передати дані через мережу одночасно тому, що два комп'ютери намагаються надіслати пакети даних через один і той же мережевий канал у той самий час.

**Ключові аспекти** колізій в мережі TCP/IP:

*Протокол CSMA/CD*: У мережах, які використовують Ethernet, для вирішення колізій використовується протокол CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Цей протокол дозволяє пристроям виявляти колізії та вирішувати їх шляхом відправки сигналу про виявлену колізію, після чого пристрої повторюють спробу надсилання даних.

*Передача даних*: Коли відбувається колізія, пристрої, які виявили колізію, чекатимуть випадковий проміжок часу перед повторною спробою передачі даних. Це дозволяє уникнути подальших колізій у мережі.

*Вплив на продуктивність*: Колізії можуть вплинути на продуктивність мережі, оскільки вони призводять до затримок у передачі даних. Тому важливо мати ефективні засоби виявлення та управління колізіями в мережі TCP/IP.

Колізія в мережі TCP/IP може виникати з різних **причин**, і ось деякі з них:

*Конфлікт IP-адрес*: Якщо два пристрої в мережі мають однакову IP-адресу, то вони не зможуть коректно спілкуватися через конфлікт адрес.

*Конфлікт MAC-адрес*: Коли два пристрої в мережі мають однаковий фізичний MAC-адрес, це може спричинити колізію.

*Переповнення мережі*: Якщо мережа перевантажена великою кількістю трафіку, це може призвести до колізій між пакетами даних.

*Помилки в кабелі або обладнанні*: Дефективні кабелі або обладнання можуть спричинити передачу некоректних сигналів, що в свою чергу може спричинити колізії в мережі.

*Дублювання IP-адрес*: Якщо в мережі виникає дублювання IP-адрес, це також може спричинити колізії.

Щоб уникнути колізій в мережі, важливо правильно налаштувати IP-адреси, використовувати унікальні MAC-адреси для кожного пристрою, регулярно перевіряти стан обладнання та кабелів, а також моніторити навантаження мережі для запобігання переповненню.

## 32. Що таке проксі в мережі TCP/IP?

Проксі (proxy) в мережі TCP/IP - це сервер-посередник, який використовується для забезпечення проміжного з'єднання між клієнтом і сервером. Ось кілька основних функцій проксі в мережі TCP/IP:

*Фільтрація трафіку*: Проксі може фільтрувати мережевий трафік, що проходить через нього, дозволяючи або блокуючи певні типи запитів або даних. Це може бути корисно для забезпечення безпеки та контролю доступу до веб-ресурсів.

*Підвищення швидкодії*: Проксі може кешувати запити клієнтів і відповіді від серверів, що дозволяє прискорити доступ до ресурсів, особливо у випадках, коли одні й ті самі дані запитуються декілька разів.

*Анонімізація*: Проксі може приховати реальну IP-адресу клієнта від сервера, що дозволяє зберегти приватність та анонімність при перегляді веб-сторінок чи взаємодії з іншими ресурсами в мережі.

*Обхід блокування*: Проксі може допомогти обійти блокування веб-ресурсів, яке може бути встановлене на рівні мережі чи країни.

Проксі може бути налаштований на рівні клієнта або в мережі, і він може мати різні типи, такі як HTTP-проксі, SOCKS-проксі, SSL-проксі та інші, які використовуються для різних цілей у мережі TCP/IP.

## 33. Що таке firewalls в компьютерних мережах і навіщо вони потрібні?

Firewall (брандмауер) в комп'ютерних мережах - це програмне або апаратне забезпечення, яке призначене для контролю трафіку мережі, що проходить через нього, та захисту мережі від несанкціонованого доступу. Основні функції та значення брандмауера :

*Захист від несанкціонованого доступу*: Брандмауер встановлює правила, які контролюють, який мережевий трафік може проходити через нього, та блокує небажані підключення, забезпечуючи захист від хакерських атак, вторгнень та несанкціонованого доступу до мережі.

*Фільтрація трафіку*: Брандмауер може фільтрувати мережевий трафік за різними параметрами, такими як IP-адреси, порти, протоколи тощо. Це дозволяє обмежити доступ до певних ресурсів чи послуг у мережі.

*Захист від вірусів та шкідливих програм*: Багато сучасних брандмауерів також мають можливості виявлення та блокування шкідливих програм та вірусів, які намагаються проникнути в мережу.

*Підвищення конфіденційності та безпеки даних*: Брандмауер може допомогти захистити конфіденційні дані, які зберігаються в мережі, від несанкціонованого доступу.

*Керування мережевим трафіком*: Брандмауер може допомагати в керуванні мережевим трафіком, забезпечуючи при цьому оптимальну швидкість та продуктивність мережі.

## 34. Що таке NAT і навіщо він потрібен в комп’ютерних мережах?

NAT (Network Address Translation) - це технологія, що використовується в комп'ютерних мережах для перетворення IP-адрес клієнтів у внутрішній мережі на інші IP-адреси для зовнішньої мережі, зазвичай для доступу до Інтернету. Ключові аспекти:

*Захист внутрішньої мережі*: NAT дозволяє приховати реальні IP-адреси пристроїв у внутрішній мережі за однією або кількома публічними IP-адресами. Це забезпечує певний рівень безпеки, оскільки зовнішні пристрої бачать лише публічні IP-адреси, а не всі приватні IP-адреси в мережі.

*Економія IP-адрес*: NAT дозволяє використовувати невелику кількість публічних IP-адрес для доступу всіх пристроїв у внутрішній мережі до Інтернету. Це стає особливо важливим у випадку, коли кількість доступних публічних IP-адрес обмежена.

*Дозвіл на спільне використання однієї IP-адреси*: Багато користувачів можуть використовувати одну публічну IP-адресу для доступу до Інтернету через NAT. Це особливо корисно в домашніх мережах та офісних оточеннях.

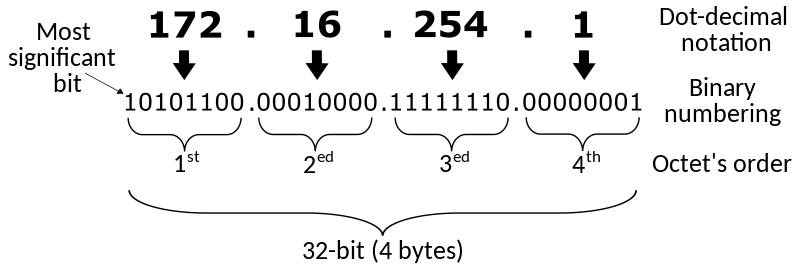
*Підтримка внутрішніх служб*: NAT дозволяє встановлювати внутрішні сервіси та сервери, які можуть бути доступні зовнішнім користувачам через мережу, використовуючи механізми перенаправлення портів (port forwarding).

Отже, NAT є важливою технологією для забезпечення безпеки, ефективного використання IP-адрес та підтримки внутрішніх та зовнішніх комунікацій в комп'ютерних мережах.

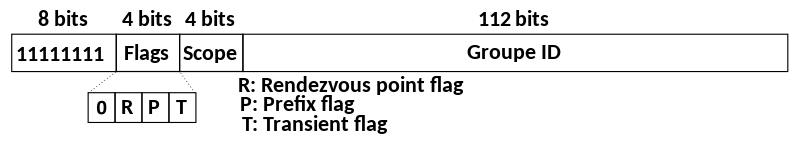
## 35. Які типи IP-адрес бувають?

*IPv4*: Це найпоширеніший тип IP-адрес. Він складається з 4 чисел, кожне з яких може бути від 0 до 255, розділених крапками. Наприклад, 192.168.1.1.

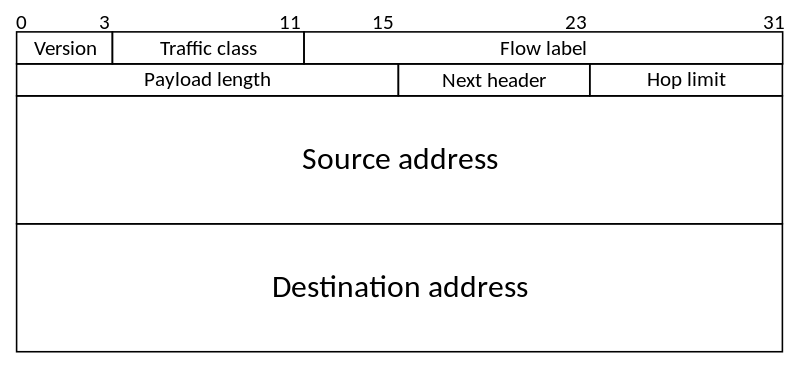
*IPv6*: Цей тип IP-адрес використовується для розширення кількості доступних IP-адрес, оскільки запаси IPv4 починають вичерпуватися. Він складається з груп квадратних дужок, чисел та літер: 3ffe:1900:4545:3:200:f8ff:fe21:67cf. IPv6 використовує 128-бітні адреси

Figure 1: IPv4 decomposition

IPv6 - протокол Інтернет-рівня для міжмережевої роботи з комутацією пакетів і забезпечує наскрізну передачу дейтаграм через кілька IP-мереж, чітко дотримуючись принципів розробки, розроблених у IPv4. Реалізує нові функції. Це спрощує конфігурацію адреси, перенумерацію мережі та сповіщення маршрутизатора під час зміни постачальника підключення до мережі. Спрощує обробку пакетів у маршрутизаторах, покладаючи відповідальність за фрагментацію пакетів на кінцеві точки. Розмір підмережі IPv6 стандартизовано шляхом фіксації розміру частини ідентифікатора хоста в адресі до 64 бітів.

Figure 2: IPv6 Multicast header

*Статичні IP-адреси*: Цей тип IP-адрес призначений для постійного призначення для конкретного пристрою. Він залишається незмінним з часу налаштування.

Figure 3: IPv6 Packet header

*Динамічні IP-адреси*: Ці IP-адреси призначаються пристроям автоматично при підключенні до мережі. Вони можуть змінюватися під час кожного підключення до мережі.

*Публічні та приватні IP-адреси*: Публічні IP-адреси доступні в Інтернеті, тоді як приватні IP-адреси використовуються в межах приватних мереж, наприклад, вдома або в офісі.

## 36. За яким портом і протоколом працюють Ping і Traceroute?

У операційній системі Linux команда **Ping** використовує протокол ICMP (Internet Control Message Protocol) і відправляє запити до вказаної IP-адреси для перевірки доступності цієї адреси в мережі. Ping не працює за допомогою портів, оскільки використовує протокол ICMP, який відмінний від традиційних портів TCP або UDP.

Traceroute відрізняється тим, що він перевіряє, як дані переміщуються через Інтернет. Подібним чином ви можете використовувати час DNS TTL для трасування, але воно не слідує шляху даних між маршрутизаторами.

**Traceroute** також використовує протокол ICMP для відстеження маршруту до вказаної IP-адреси. Traceroute використовує UDP і відправляє пакети до кожного вузла в мережі, збільшуючи значення TTL (Time to Live) у кожному пакеті. Це дозволяє визначити шлях, яким йде пакет до вказаної IP-адреси. У цьому випадку Traceroute використовується з портом 33434.

On Unix-like operating systems, traceroute sends, by default, a sequence of User Datagram Protocol (UDP) packets, with destination port numbers ranging from 33434 to 33534; the implementations of traceroute shipped with Linux, FreeBSD, NetBSD, OpenBSD, DragonFly BSD, and macOS include an option to use ICMP Echo Request packets (-I), or any arbitrary protocol (-P) such as UDP, TCP using TCP SYN packets, or ICMP.

Протоколи TCP (Transmission Control Protocol) та UDP (User Datagram Protocol) є двома основними протоколами транспортного рівня в мережевій моделі OSI. Ось деякі відмінності між ними:

*TCP (Transmission Control Protocol):*

Забезпечує надійний зв'язок між відправником і отримувачем шляхом встановлення з'єднання, передачі даних та підтвердження отримання.

Гарантує доставку даних у правильному порядку та в разі втрати чи пошкодження даних відправляє повторні запити.

Використовується для передачі даних, які вимагають надійності та порядку, наприклад, веб-сторінки, електронна пошта, завантаження файлів.

*UDP (User Datagram Protocol):*

Набагато менш надійний, оскільки не встановлює з'єднання та не гарантує доставку даних.

Швидший у порівнянні з TCP, оскільки не має накладних витрат на управління з'єднанням та підтвердженнями.

Використовується для передачі даних, де швидкість важливіша за надійність, наприклад, стрімінг відео, голосові дзвінки в месенджерах.

Вибір між TCP та UDP залежить від конкретних потреб програми чи сервісу.

# Junior DevOps - Clouds

## 37. У чому різниця між IaaS, PaaS та SaaS?

IaaS, PaaS та SaaS - це різні моделі хмарних послуг, які надаються компаніями для забезпечення інфраструктури, платформи та програмного забезпечення. Ось основні відмінності між ними:

*IaaS (Infrastructure as a Service):*

Надається віртуальна інфраструктура, така як обчислювальні потужності, мережі, зберігання даних. Користувач самостійно керує операційною системою, додатками та даними.

Приклади: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform.

*PaaS (Platform as a Service):*

Надається платформа для розробки, тестування та розгортання додатків. Користувачі можуть будувати та розгортати власні додатки без необхідності управління інфраструктурою.

Приклади: Microsoft Azure, Google App Engine, Heroku.

*SaaS (Software as a Service):*

Надається готове програмне забезпечення через Інтернет. Користувачі мають доступ до програмного забезпечення як послуги без необхідності установки або підтримки.

Приклади: Google Workspace, Microsoft 365, Salesforce.

## 38. Що таке VPC і з яких компонентів має складатись?

*VPC (Virtual Private Cloud)* - це віртуальна приватна мережа в хмарних обчисленнях, яка надає ізольоване середовище для розгортання ресурсів та додатків. Дозволяє користувачам створювати свою власну віртуальну мережу. Ось основні компоненти VPC:

*Subnet (Підмережа):*

Це частина IP-адресного простору в межах VPC, яка визначається маскою підмережі.

Subnet ділиться на публічні та приватні для різних типів ресурсів та застосунків.

*Route Tables (Таблиці маршрутизації)*:

Визначають напрямок трафіку в межах VPC.

Кожна підмережа пов'язана з конкретною таблицею маршрутизації.

*Internet Gateway (Шлюз до Інтернету):*

Забезпечує доступ до Інтернету для ресурсів, розгорнутих у публічних підмережах VPC.

*NAT Gateway (Мережевий адресний переклад):*

Дозволяє приватним ресурсам у внутрішніх підмережах VPC здійснювати доступ до Інтернету через публічні підмережі.

*Security Groups (Групи безпеки):*

Визначають правила для вхідного та вихідного трафіку для ресурсів у VPC.

Контролюють доступ до різних ресурсів в межах VPC.

*Network Access Control Lists (ACL):*

Це списки правил, які фільтрують трафік на рівні підмереж та контролюють доступ до них.

## 39. Що таке cloud-init? Що таке init/systemd/upstart configs?

**cloud-init** - це програмне забезпечення з відкритим кодом, яке використовується для налаштування віртуальних машин та інших облікових записів у хмарних обчисленнях. Ось деякі ключові аспекти cloud-init:

*Автоматизована настройка*: cloud-init дозволяє автоматизовано налаштовувати нові віртуальні машини при їх запуску. Це може включати налаштування мережі, створення користувачів, встановлення SSH ключів та інші операції.

*Підтримка різних хмарних платформ*: cloud-init є стандартом для налаштування віртуальних машин у багатьох хмарних платформах, таких як Amazon Web Services, Google Cloud Platform, Microsoft Azure та інші.

*Конфігураційні файли*: cloud-init використовує конфігураційні файли у форматі YAML для налаштування при створенні віртуальної машини.

*Розширені можливості*: cloud-init підтримує виконання скриптів, завантаження пакунків, встановлення програмного забезпечення та інші завдання налаштування.

**init/systemd/upstart configs** відносяться до конфігураційних файлів в операційній системі Linux, які відповідають за налаштування та управління процесами та службами при завантаженні системи. Ось короткий огляд кожної з цих технологій:

*init*: Це старійший та базовий системний менеджер, який використовувався для ініціалізації системи та управління процесами в більш ранніх версіях Linux. Конфігураційні файли для *init* зазвичай знаходяться в /etc/init.d/ та /etc/rc\*.d/. Ці файли визначають, які служби потрібно запускати або припиняти при завантаженні системи.

*systemd*: Це сучасний менеджер системних процесів, який замінив *init* у більшості сучасних дистрибутивів Linux. Конфігураційні файли для systemd зазвичай знаходяться в /etc/systemd/system/. systemd пропонує розширені можливості управління службами, такі як паралельний запуск, контроль процесів, обмеження ресурсів та інші функції.

*upstart*: Це альтернативний менеджер процесів, який також використовувався в деяких дистрибутивах Linux, таких як Ubuntu. Конфігураційні файли для *upstart* зазвичай знаходяться в /etc/init/. *upstart* також дозволяє визначати, які служби та процеси повинні бути запущені при завантаженні системи.

Усі ці конфігураційні файли дозволяють адміністраторам систем змінювати параметри запуску різних служб та процесів при завантаженні системи, що спрощує управління системою та її компонентами.

# Junior DevOps - Automation

## 40. Що таке IaaC і навіщо він потрібний?

**Infrastructure as Code (IaC)** - це підхід до управління ІТ-інфраструктурою, в якому інфраструктура, така як сервери, мережеві ресурси, бази даних тощо, управляється за допомогою коду та автоматизації.

Також дивись 4. Поясніть концепцію Infrastructure as a Code, для чого це потрібно і які проблеми вирішує? аркуш 5.

**Навіщо потрібне IaC**:

Автоматизація: IaC дозволяє автоматизувати процеси розгортання та управління інфраструктурою, зменшуючи ризик помилок та підвищуючи ефективність.

*Контроль версій*: Інфраструктура визначається в коді, що дозволяє використовувати системи контролю версій, такі як Git, для відстеження змін та відновлення попередніх станів.

*Стабільність та надійність*: Завдяки IaC можна гарантувати, що інфраструктура буде стабільною та надійною завдяки повторюваним та контрольованим процесам розгортання.

*Масштабованість*: Завдяки коду можна швидко та легко масштабувати інфраструктуру, відповідаючи на змінні потреби бізнесу.

*Документація*: Код інфраструктури служить документацією, яка пояснює, як саме працює система та її компоненти.

IaC дозволяє розглядати інфраструктуру як код, що спрощує та полегшує управління й відповідності вимогам, та дозволяє швидко реагувати на зміни в бізнес-потребах.

## 41. Що таке Terraform?

**Terraform** - це інструмент для управління інфраструктурою як кодом (IaC), створений компанією HashiCorp переважно для роботи з хмарними середовищами. Він дозволяє описати вашу інфраструктуру у вигляді коду за допомогою конфігураційних файлів, таких як HCL (HashiCorp Configuration Language), та потім автоматично створити, змінити та видаляти цю інфраструктуру.

**Основні особливості Terraform:**

*Декларативний підхід*: Terraform використовує декларативний підхід до описування інфраструктури, що дозволяє зосередитися на бажаному стані системи, а не на послідовності кроків для досягнення цього стану.

*Підтримка різних провайдерів*: Terraform підтримує багато провайдерів хмарних послуг та інших інфраструктурних ресурсів, включаючи AWS, Azure, Google Cloud, VMware, Kubernetes та багато інших.

*Граф залежностей ресурсів*: Terraform автоматично визначає та керує залежностями між різними ресурсами, що дозволяє ефективно та безпечно управляти складними конфігураціями.

*Інфраструктура як код*: Terraform дозволяє створювати, змінювати та видаляти інфраструктуру за допомогою коду, що спрощує автоматизацію та керування інфраструктурою.

Terraform є потужним інструментом для автоматизації управління інфраструктурою та дозволяє забезпечувати стабільність, надійність та ефективність в управлінні різноманітними інфраструктурними середовищами.

## 42. Які інструменти автоматизації ви знаєте?

*Ansible*: Це відкрите програмне забезпечення для автоматизації управління комп'ютерними системами, включаючи конфігурацію, розгортання програмного забезпечення та оркестрацію різноманітних завдань. Основна мета Ansible - забезпечити простий та ефективний спосіб автоматизації IT-середовищ за допомогою зручного та легкого в користуванні інструменту.

*Chef*: Інструмент конфігурації та автоматизації, який дозволяє управляти станом інфраструктури за допомогою "рецептів" та "ресурсів".

*Puppet*: Інша платформа для конфігурації та управління системами, яка використовує декларативний підхід та мову виразів для опису стану систем.

*Terraform*: Інструмент для управління інфраструктурою як кодом (IaC), створений компанією HashiCorp переважно для роботи з хмарними середовищами.

*CloudFormation*: Сервіс від Amazon Web Services (AWS), який дозволяє описувати та розгортати інфраструктуру AWS у вигляді коду.

*Pulumi*: Інструмент, який дозволяє описувати інфраструктуру як код та розгортати її на різних хмарних платформах, використовуючи мови програмування, такі як JavaScript, Python, Go тощо.

*Bash*: (Bourne Again SHell) - це командний інтерпретатор та мова програмування оболонки в середовищі Unix та Unix-подібних операційних систем. Основна мета Bash - надати користувачам зручний інтерфейс для взаємодії з операційною системою шляхом виконання команд та скриптів.

*Python*: це високорівнева мова програмування, яка часто використовується для автоматизації різноманітних завдань у сфері розробки програмного забезпечення та управління IT-інфраструктурою. Python надає потужні засоби для автоматизації завдань завдяки своїм зручним синтаксисом та великому вибору бібліотек та фреймворків.

Ці інструменти дозволяють автоматизувати конфігурацію та управління інфраструктурою, забезпечуючи ефективність, стабільність та надійність у роботі з різноманітними середовищами.

# Junior DevOps - Information Security

## 43. У чому різниця між аутентифікацією та авторизацією?

Аутентифікація та авторизація - це два різних процеси, які забезпечують безпеку доступу до систем та ресурсів. Ось їх визначення та відмінності:

*Аутентифікація:*

Це процес перевірки ідентичності користувача, який намагається увійти до системи.

Аутентифікація визначає, чи є користувач тим, за кого він собою видається.

Зазвичай це включає в себе введення імені користувача та пароля, а також може включати біометричні дані, карти доступу тощо.

*Авторизація:*

Це процес визначення прав доступу користувача після того, як його аутентифікація була успішною.

Авторизація визначає, до яких ресурсів чи функцій користувач має доступ після вдалого входу в систему.

Наприклад, після входу в систему користувач може мати доступ до певних файлів, може виконувати певні операції або ж мати обмежений доступ.

## 46. Що таке MFA, TOTP?

MFA означає багатофакторну аутентифікацію (Multi-Factor Authentication)

TOTP - це скорочення від "Time-Based One-Time Password"

*Багатофакторна аутентифікація (MFA)*: Це метод аутентифікації, який вимагає від користувача надати додаткові фактори для підтвердження своєї ідентичності. Замість того, щоб використовувати лише пароль, MFA вимагає використання ще одного або кількох факторів, таких як фізичний об'єкт (наприклад, ключ-карта), біометричні дані (наприклад, відбиток пальця) або одноразовий пароль.

*Time-Based One-Time Password (TOTP)*: Це форма MFA, яка використовує випадково згенерований код як додатковий токен аутентифікації. Код TOTP діє лише протягом обмеженого періоду часу і змінюється автоматично через певний інтервал. Користувач отримує цей код на свій мобільний пристрій або інший пристрій, який підтримує TOTP, і вводить його разом зі своїм паролем для підтвердження своєї ідентичності.

**TOTP є одним з методів реалізації MFA** і забезпечує додатковий рівень безпеки, оскільки навіть якщо зловмисник дізнається пароль, він все одно не зможе отримати доступ без додаткового TOTP-коду, який змінюється з часом.

## 44. Сертифікати. Як працює HTTPS? Що таке certificate ciphers?

Сертифікати SSL (Secure Sockets Layer) та TLS (Transport Layer Security) - це технології, які забезпечують шифрування комунікації та підтвердження ідентичності в Інтернеті. Основна різниця між ними полягає в тому, що SSL є старішою версією протоколу, тоді як TLS є його сучасним наступником.

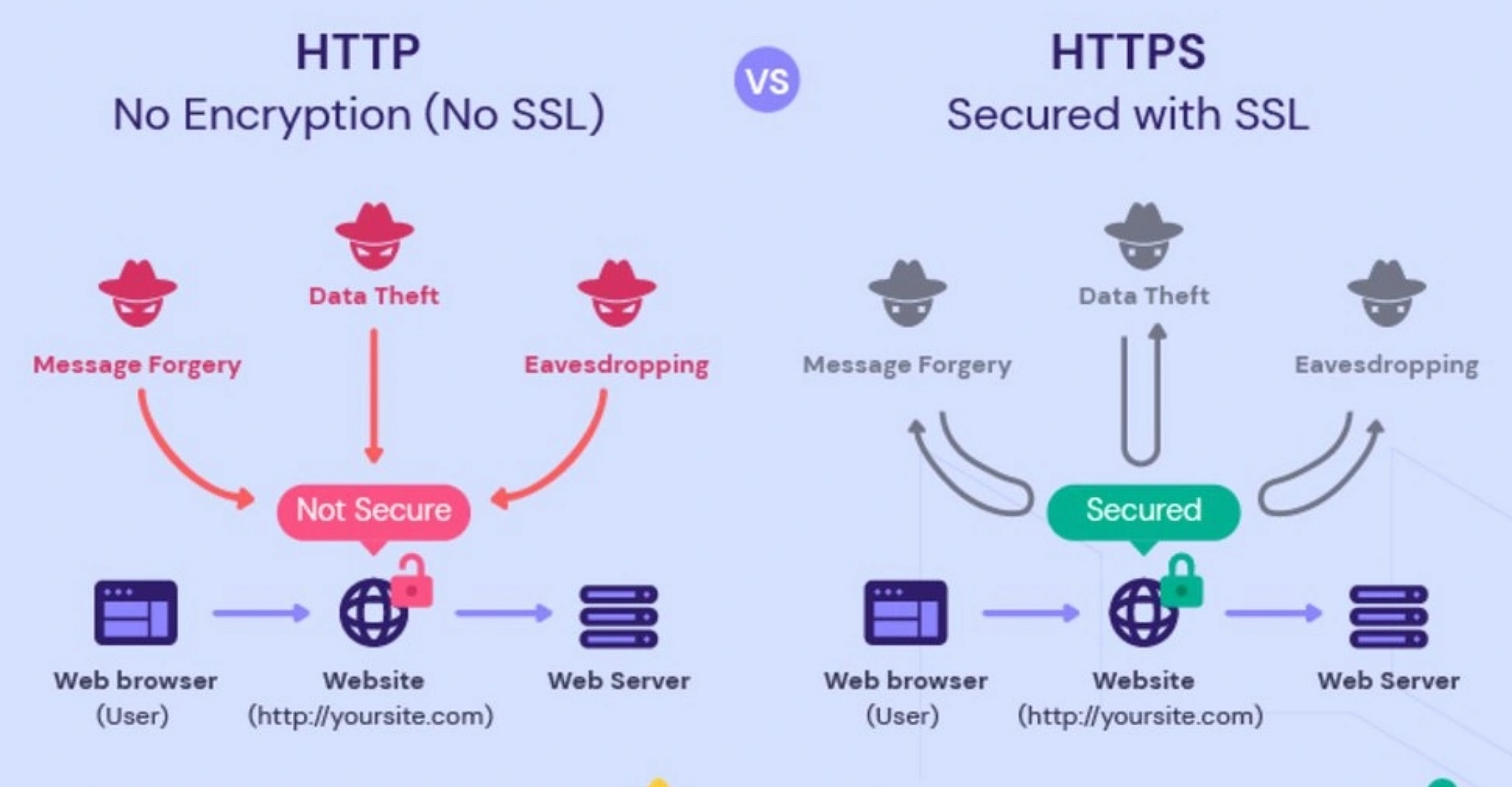
Докладніше дивись 3. Як працює HTTPS? аркуш 4

**certificate ciphers** - Шифри сертифікатів TLS (Transport Layer Security) - це набір алгоритмів, які допомагають забезпечити безпеку мережевого з'єднання. Вони використовуються для шифрування та розшифрування даних, які передаються між клієнтом і сервером під час взаємодії через протокол TLS.

*Основні відомості про шифри сертифікатів TLS:*

* Шифри сертифікатів TLS використовуються для забезпечення конфіденційності, цілісності та аутентифікації даних під час їх передачі по мережі.
* Вони включають в себе різні алгоритми шифрування, хешування та обміну ключами, які використовуються для захисту даних.
* Шифри сертифікатів TLS визначаються в рамках так званих "cipher suites" (наборів шифрів), які включають в себе конкретні алгоритми та параметри шифрування.
* Клієнт і сервер взаємодіють під час процесу "handshake" (взаємодії) для вибору найбільш підходящого шифру сертифіката TLS для забезпечення безпеки з'єднання.
* Шифри сертифікатів TLS можуть бути оновлювані та розширювані для забезпечення більшої безпеки та сумісності з новими технологіями.

*Хто робить валідацію сертифікатів SSL/TLS*? Certification Authority (СА), що продає сертифікати SSL/TLS власникам веб-сторінок, веб-хостинг компаніям та підприємствам. СА перевіряє дані про домен і власника перед видачею сертифіката SSL/TLS.



## 45. Як безпечно передати дані своєму колезі?

Щоб безпечно передати дані своєму колезі через мережу, варто врахувати наступні рекомендації:

*Використовуйте шифрування*: Передавайте дані за допомогою протоколів шифрування, таких як SSL (Secure Sockets Layer) або TLS (Transport Layer Security). Це допоможе захистити дані від несанкціонованого доступу під час їх передачі.

*Використовуйте безпечні мережі*: Передавайте дані через захищені мережі, такі як власна Wi-Fi мережа з паролем або використання віртуальної приватної мережі (VPN). Це допоможе уникнути перехоплення даних з боку зловмисників.

*Перевіряйте достовірність отримувача*: Перед тим, як передати дані, переконайтеся, що отримувач є довіреною особою або організацією. Використовуйте безпечні канали комунікації, такі як зашифровані електронні пошти або месенджери з криптографічними протоколами.

*Застосовуйте паролі та інші методи аутентифікації*: Використовуйте сильні паролі та інші методи аутентифікації, такі як двофакторна аутентифікація, для захисту доступу до ваших даних. Це допоможе уникнути несанкціонованого доступу до переданих даних.

*Оновлюйте програмне забезпечення*: Переконайтеся, що ваше програмне забезпечення, включаючи операційну систему та застосунки, оновлене до останньої версії. Це допоможе уникнути використання вразливостей, які можуть бути використані для несанкціонованого доступу до ваших даних.

*Використовуйте файрвол та антивірусне програмне забезпечення*: Встановіть та оновлюйте файрвол та антивірусне програмне забезпечення на своєму пристрої. Це допоможе виявити та запобігти вторгненням та шкідливому програмному забезпеченню, які можуть загрожувати безпеці ваших даних.

*Будьте обережні з відкритими мережами*: Уникайте передачі чутливих даних через відкриті мережі, такі як громадські Wi-Fi точки доступу. Ці мережі можуть бути небезпечними, оскільки зловмисники можуть перехоплювати передані дані.

# Junior DevOps - Віртуалізація

## 47. У чому різниця між віртуалізацією та контейнеризацією? У чому плюси та мінуси?

Дивись докладно:

8. У чому різниця між віртуалізацією і контейнеризацією?

9. У чому переваги контейнерів?

## 48. Як для при запуску Docker-контейнера підключити його з 80 порту в контейнері на 8081 на хості?

$ docker run -p 8081:80 <назва\_образу>

У цій команді:

* -p 8081:80 вказує на мапування порту 80 в контейнері на порт 8081 на хості.
* <назва\_образу> - це ім'я образу контейнера, який ви хочете запустити.

## 49. Як передати у віртуальну машину USB device?

Щоб передати USB-пристрій у віртуальну машину, ви можете скористатися програмними засобами гіпервізора, такими як VirtualBox або VMware. Ось кроки для передачі USB-пристрою у віртуальну машину за допомогою VirtualBox:

* Відкрийте VirtualBox та виберіть віртуальну машину, до якої ви хочете передати USB-пристрій.
* У вікні віртуальної машини виберіть "Devices" (Пристрої) у головному меню VirtualBox.
* У підменю "USB" ви побачите список доступних USB-пристроїв. Оберіть потрібний USB-пристрій, який ви хочете передати у віртуальну машину.
* Після вибору USB-пристрою, він буде автоматично підключений до віртуальної машини і стане доступним у ній.

Для VMware, процедура може трохи відрізнятися, але загальний принцип залишається схожим - вибрати USB-пристрій для передачі у віртуальну машину через вікно налаштувань гіпервізора.

**Гіпервізор (англ. hypervisor)** - це програмне забезпечення або апаратний засіб, який дозволяє створювати та управляти віртуальними машинами. Головна функція гіпервізора полягає в тому, щоб дозволити кільком операційним системам працювати на одному фізичному комп'ютері, використовуючи його ресурси ефективно та ізоляцію однієї операційної системи від іншої. Існують два типи гіпервізорів:

*Тип 1 (напрямий, bare-metal)*: Цей тип гіпервізора встановлюється безпосередньо на фізичний сервер і управляє віртуальними машинами безпосередньо. Приклади: VMware ESXi, Microsoft Hyper-V, Xen.

*Тип 2 (господарський, host)*: Цей тип гіпервізора встановлюється як додаток до операційної системи. Він дозволяє користувачеві запускати віртуальні машини на робочій станції або сервері. Приклади: VirtualBox, VMware Workstation, Parallels Desktop.

## 50. Docker-контейнер споживає багато SWAP. Що робити?

Docker-контейнер споживає багато пам'яті SWAP коли є проблеми з використанням ресурсів або налаштуваннями контейнера. Ось як можна вирішити ці проблеми:

* *Перевірте ресурси контейнера*: Переконайтеся, що контейнер не споживає занадто багато ресурсів. Встановіть обмеження на використання пам'яті та CPU для контейнера, використовуючи параметри Docker, такі як --memory та --cpus.
* *Оптимізуйте роботу програм у контейнері*: Перевірте програми, які виконуються в контейнері, та впевніться, що вони оптимізовані для ефективного використання ресурсів. Можливо, деякі процеси або служби витрачають занадто багато пам'яті.
* *Перевірте налаштування пам'яті Docker*: Переконайтеся, що налаштування пам'яті Docker відповідають реальним потребам контейнера. Деякі контейнери можуть вимагати зміни параметрів, таких як --memory-swap, для кращого управління пам'яттю.
* *Моніторинг та аналіз ресурсів*: Використовуйте інструменти моніторингу, такі як cAdvisor або Docker Stats, щоб відстежувати використання ресурсів контейнерами та виявити, який процес або служба споживає багато пам'яті.
* *Розгляньте використання інших інструментів*: Деякі випадки потребують інших інструментів віртуалізації або контейнеризації, таких як Kubernetes, які надають більше можливостей для управління ресурсами.

# Junior DevOps - CI/CD

## 51. Що таке Continuous Integration та Continuous Deployment? У чому різниця між Continuous Deployment і Continuous Delivery?

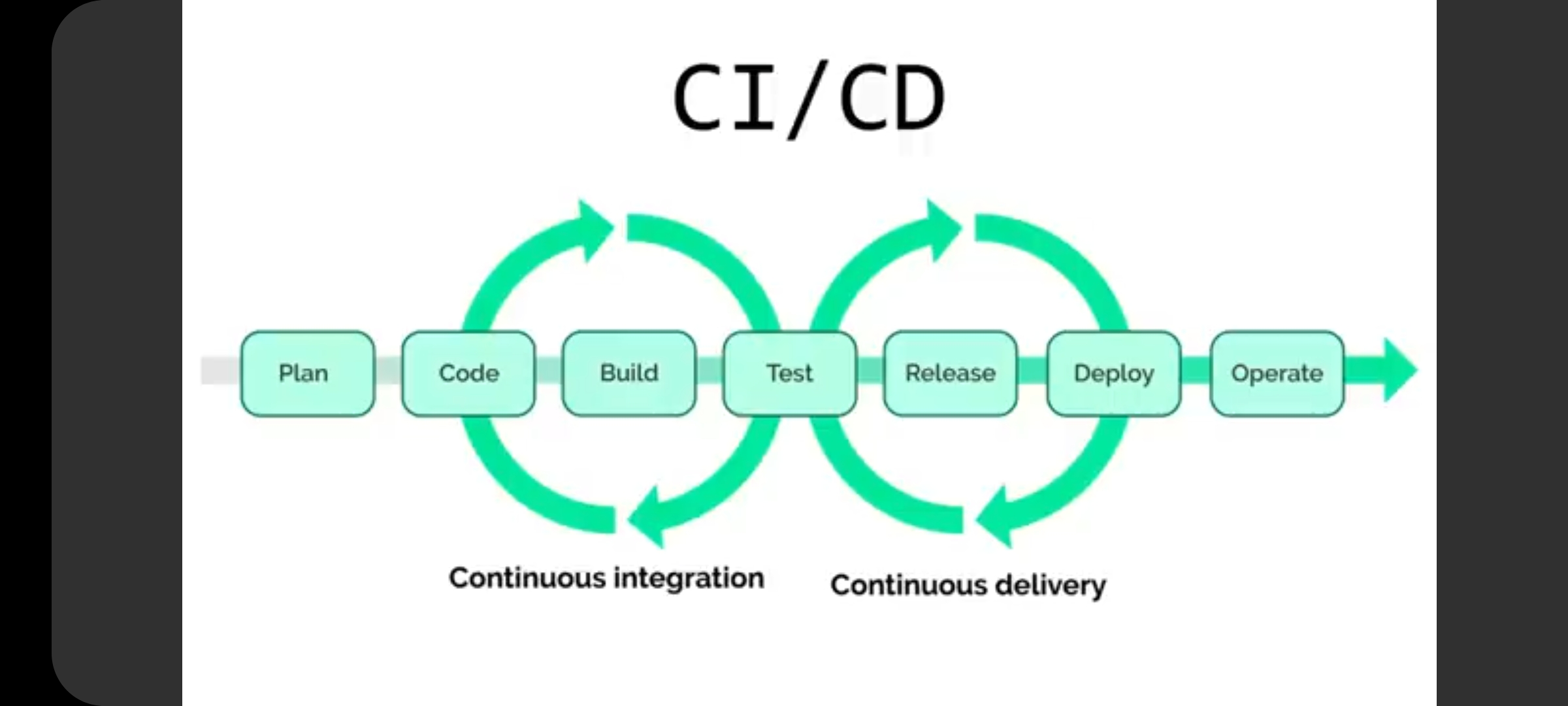
*Continuous Integration (CI)* є практикою розробки програмного забезпечення, яка полягає в автоматизованому об'єднанні (інтеграції) змін у коді команди розробників на регулярній основі. Основна мета CI - це зменшення конфліктів та проблем інтеграції при частому об'єднанні коду в спільний репозиторій та автоматичного запуску тестів для виявлення помилок якомога раніше у процесі розробки. Це допомагає забезпечити швидкий зворотній зв'язок про якість коду та швидке виявлення проблем.

*Continuous Deployment (CD)* - це практика автоматизованого випуску програмного забезпечення до середовищ production після проходження всіх тестів і вимог для випуску. Це означає, що кожне успішне злиття коду в основну гілку гіт-репозиторію автоматично спричиняє розгортання цього коду на виробничні сервери без необхідності втручання розробників або операторів. Це дозволяє швидко впроваджувати новий функціонал та виправлення помилок у середовищі production, зменшуючи час між написанням коду та його виконанням в продукції.

*Continuous* ***Deployment***означає автоматизований процес випуску програмного забезпечення, при якому кожне успішне злиття коду в основну гілку гіт-репозиторію призводить до автоматичного **розгортання цього коду на виробничні сервери без втручання розробників або операторів**.

Швидкий цикл впровадження змін: Це означає, що новий код автоматично впроваджується в продукційне середовище після проходження всіх тестів та вимог для випуску.

Автоматизоване розгортання: Ця практика вимагає високого рівня автоматизації в процесі розгортання, оскільки весь процес випуску відбувається автоматично після злиття коду.

*Continuous* ***Delivery***також передбачає автоматизований процес випуску програмного забезпечення, але **відмінюється тим, що випуск виробничого коду відбувається вручну.** Це означає, що після успішного проходження всіх тестів та вимог для випуску, код готовий до розгортання виробничне середовище, але **розгортання виробничного коду відбувається тільки після ручного підтвердження**.

Керований реліз: Continuous Delivery передбачає, що розгортання нового коду prodiction середовище потребує людського втручання для підтвердження.

## 52. Опишіть основні етапи Continuous Integration та Continuous Deployment (CI/CD).

Ось основні етапи Continuous Integration (CI) та Continuous Deployment (CD):

*Continuous Integration (CI):*

* Злиття коду: Кожен розробник зливає свій код в спільний репозиторій, де він автоматично об'єднується з іншими змінами в коді.
* Автоматичні тести: Після злиття коду запускаються автоматичні тести, які перевіряють, чи не порушено функціональність програми після змін.
* Зворотний зв'язок: Якщо тестування не пройшло успішно, розробники отримують повідомлення про проблеми, що дозволяє їм швидко виправити помилки.

*Continuous Deployment (CD):*

* Автоматизоване тестування: Після успішного CI вступає в силу CD. Автоматично запускаються розширені тести, включаючи тести на вимоги для випуску виробничого коду.
* Автоматичне розгортання: Якщо всі тести пройдені успішно, код автоматично розгортається на виробничні сервери без необхідності втручання розробників.
* Моніторинг та зворотний зв'язок: Після розгортання коду виробничне середовище моніториться для виявлення проблем та збору даних про продукційну роботу програми.

## 53. Опишіть приклад процесу Continuous Integration (CI), який починається з моменту, коли розробник запушив зміни / PR (pull request) до Git?

*Створення гілки та внесення змін*: Розробник створює нову гілку або вносить зміни до існуючої гілки в репозиторії Git, які потрібно перевірити та об'єднати з основною гілкою.

*Створення pull request (PR)*: Після внесення змін розробник створює pull request, щоб запропонувати зміни для об'єднання з основною гілкою. У PR зазвичай надається опис змін, їхній контекст та мета.

*Запуск CI-пайплайну*: Після створення PR система Continuous Integration автоматично запускає CI-пайплайн, який включає в себе наступні кроки:

* Клонування репозиторію: CI-пайплайн клонує код з відповідної гілки або PR.
* Встановлення залежностей: Система встановлює всі необхідні залежності та пакети, необхідні для компіляції та запуску тестів.
* Виконання тестів: Запускаються автоматичні тести, такі як модульні, інтеграційні, функціональні тести та інші, які перевіряють функціональність нового коду та виявляють можливі проблеми.
* Аналіз якості коду: У деяких випадках може бути виконаний аналіз якості коду за допомогою спеціалізованих інструментів.
* Звіти та повідомлення: Після завершення пайплайну розробник отримує звіти про виконання тестів та інші результати.

*Зворотний зв'язок:* Якщо тести пройдені успішно, розробник може об'єднати свої зміни з основною гілкою. У випадку неуспішних тестів розробник отримує повідомлення про проблеми, що дозволяє йому виправити помилки в коді та знову запустити CI-пайплайн.

Цей процес дозволяє автоматизувати тестування, знижує час на виявлення та виправлення помилок, а також забезпечує високу якість коду, що в результаті сприяє покращенню продуктивності розробників та якості програмного забезпечення.

## 53a. Наведіть приклад процесу Continuous Deployment (CD) , який починається з моменту, коли розробник запушив зміни / pull request (PR) до Git?

*Створення та відправлення змін:* Розробник створює нову функціональність або вносить зміни до існуючого коду в репозиторії, а потім відправляє ці зміни через pull request.

*Автоматичне тестування та перевірка якості коду:* Після відправлення змін запускається процес автоматичного тестування та перевірки якості коду, включаючи модульні тести, інтеграційні тести, функціональні тести та інші види тестів. Також проводиться аналіз якості коду за допомогою спеціалізованих інструментів.

*Створення збірки (build):* Після успішного проходження тестів та перевірки якості коду система автоматично створює збірку програми, яка може бути використана для деплою в тестове середовище.

*Автоматичний деплой в тестове середовище:* Після створення збірки відбувається автоматичний деплой програми в тестове середовище, де можна провести ручне або автоматичне тестування нових функцій та змін.

*Автоматичне тестування в тестовому середовищі:* Після деплою в тестове середовище можуть бути запущені автоматичні тести, що допомагають перевірити, як програма працює в реальних умовах.

*Ручне тестування нових функцій та змін:* Окрім автоматичних тестів, можуть бути проведені ручні тести нових функцій та змін для впевненості в їхній працездатності.

*Автоматичний деплой в staging/production середовище: З*міни можуть бути автоматично задеплоєні в staging або production середовище.

*Моніторинг та зворотний зв'язок:* Після деплою в реальне середовище відбувається моніторинг роботи програми, а також збір та аналіз метрик її продуктивності. Це дозволяє вчасно виявляти та виправляти можливі проблеми.

## 54. Розкажіть про різновиди тестів програмного забезпечення, які ми можемо використовувати в Continuous Integration пайплайні.

Модульні тести (Unit tests): Перевіряють окремі модулі програми (найменші частини коду) на коректність роботи ізольовано від інших частин програми.

Інтеграційні тести (Integration tests): Перевіряють, як різні модулі програми взаємодіють між собою.

Функціональні тести (Functional tests): Перевіряють, чи відповідає функціональність програми вимогам специфікації. Вони перевіряють, як програма працює в реальних умовах, спробуючи відтворити різні сценарії використання.

Збірка та розгортання (Build and deployment tests): Перевіряють, чи вдалося створити збірку програми та чи вдалося її розгорнути в тестовому середовищі. Це може включати тести інтеграції з базою даних, веб-сервером тощо.

Тести прийняття: Перевіряють, чи відповідає програма вимогам замовника. Вони можуть бути виконані вручну або автоматизовано.

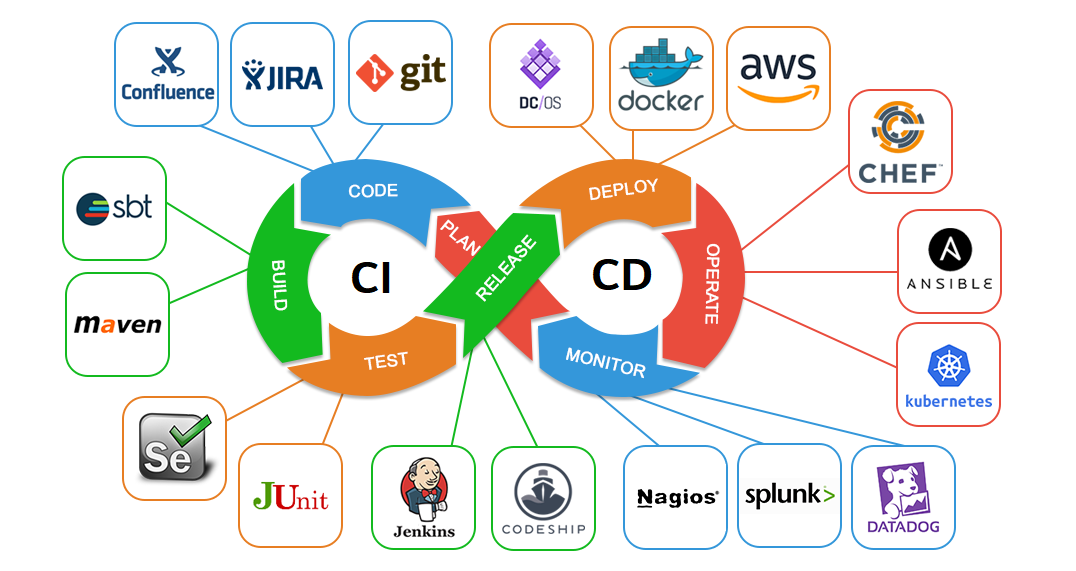
Тести витривалості: Перевіряють, як програма працює під час тривалого навантаження або в умовах обмежених ресурсів.

Тести взаємодії з іншими системами (API tests): Якщо програма взаємодіє з іншими системами чи API, важливо також мати тести, що перевіряють правильність цієї взаємодії.

Тести безпеки (Security tests): Перевіряють, чи програма відповідає вимогам щодо безпеки, чи не має вразливостей та чи не порушується захист даних користувачів.

## 55. Які інструменти CI ви використовували? Чи є досвід роботи з Jenkinsfile?

# 2023-10-05 17:30 \* 136 - Steps for Continuous Integration Pipeline

**CI pipeline setup**

1. Jenkins setup

2. Nexus Setup

3. Sonartube Setup

4. AWS Sequrity Group Setup

5. Installation all the necessary pluging in Jenkins:

+ Nexus plugin;

+ SonarQube plugin;

+ Git plugin;

+ Maven plugin;

+ Zentimestamp etc.

6. integration:

+ Nexus with Jenkins

(it is straightforward, we just need to save the credentials);

+ SonarQube with Jenkins;

(in a few steps).

7. Writing the Pipeline Script

8. Setting automatic notifications

----------------------------------------------------------------------------

def COLOR\_MAP = [

'SUCCESS': 'good',

'FAILURE': 'danger',

]

pipeline { // A simple 'pipeline-as-code' jenkins job with MAVEN Checkstyle Analysis

agent any // and SonarQube integration

environment{

JAVA\_HOME="/usr/lib/jvm/java-11-openjdk-amd64/"

}

stages {

stage('Fetch code') {

steps { //paac = 'pipeline as a code'

git branch: 'paac', url: 'https://github.com/devopshydclub/vprofile-project'

}

}

stage('Build'){

steps {

sh 'mvn install'

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('Checkstyle Analysis ') {

steps {

sh 'mvn checkstyle:checkstyle'

}

}

stage('Sonar Stage') { // SonarQube scans the code

environment { // and uploads all the reports on SonarQube server

scannerHome = tool 'sonar481'

}

steps {

withSonarQubeEnv('sonar') { // SonarQube server declaration

sh '''${scannerHome}/bin/sonar-scanner \

-Dsonar.projectKey=vprofile \

-Dsonar.projectName=vprofile-repo \

-Dsonar.projectVersion=1.0 \

-Dsonar.sources=src/ \

-Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \

-Dsonar.junit.reportsPath=target/surefire-reports/ \

-Dsonar.jacoco.reportsPath=target/jacoco.exec \

-Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml

'''

}

}

}

stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'HOURS') {

// Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails

// true = set pipeline to UNSTABLE, false = don't

waitForQualityGate abortPipeline: true

}

}

}

stage("UploadArtifact") {

steps{ //http:github.com/jenkinsci/nexus-artifact-uploader-plugin

nexusArtifactUploader(

nexusVersion: 'nexus3',

protocol: 'http',

nexusUrl: '192.168.56.12:8081', //private IP

groupId: 'QA',

version: "${env.BUILD\_ID}-${env.BUILD\_TIMESTAMP}",

repository: 'vprofile-nexus-repo',

credentialsId: 'nexuslogin',

artifacts: [

[artifactId: 'vproapp',

classifier: '',

file: 'target/vprofile-v2.war',

type: 'war']

]

)

}

}

}

post {

always {

echo 'Slack Notifications.'

slackSend channel: '#jenkinscicd',

color: COLOR\_MAP[currentBuild.currentResult],

message: "\*${currentBuild.currentResult}:\* Job ${env.JOB\_NAME} build ${env.BUILD\_NUMBER} \n More info at: ${env.BUILD\_URL}"

}

}

}

## 56. Які види тестів програмного забезпечення існують і навіщо вони потрібні?

*Модульні тести (unit tests)*: Ці тести перевіряють окремі модулі, функції або методи програми, щоб впевнитися, що вони працюють правильно. Вони допомагають виявляти помилки на ранніх етапах розробки та забезпечують високу якість коду.

*Різновидом модульних тестів є регресійні тести.*

Скажімо, у нашому сценарії є помилка, і ми намагаємося її виправити. Хорошим підходом до цього було б перше право на помилку тестування, викликавши поведінку з помилками, а потім виправити помилку, щоб тест пройшов.

*Інтеграційні тести (integrational tests)*: Ці тести перевіряють взаємодію між різними модулями або компонентами програми. Вони допомагають впевнитися, що окремі частини програми працюють разом правильно.

*Системні тести*: Ці тести перевіряють поведінку системи як цілісності. Вони охоплюють тестування функціональності, продуктивності та стійкості програми в цілому.

*Тести прийняття*: Ці тести виконуються для перевірки того, чи відповідає програма вимогам замовника. Вони перевіряють, що програмне забезпечення відповідає очікуванням користувачів та замовників.

*Тести відмови або "зламу"*: Ці тести спрямовані на перевірку того, як програмне забезпечення реагує на неправильні вхідні дані або непередбачувані умови. Вони допомагають забезпечити стійкість програми та виявляти можливі проблеми у випадку виникнення непередбачених ситуацій.

*Тест на дим (smoke test)*, який іноді називають тестом верифікації збірки: там, де є дим, є вогонь. Підключіть даний пристрій і подивіться, чи не почне з нього виходити дим.

Димовий тест відповідає на основні запитання, наприклад, чи працює програма?

Ці тести зазвичай виконуються перед більш детальним тестуванням.

Для сценарію автоматизації димовий тест полягав би в тому, щоб запустити його вручну з деякими базовими введеннями та перевірити, чи сценарій завершується успішно.

*Тести навантаження (load tests)*. Ці тести перевіряють, чи система добре поводиться під час значного навантаження. Щоб фактично виконати ці тести, знадобиться генерувати трафік до нашої програми, імітуючи типове використання служби.

Ці тести можуть бути надзвичайно корисними під час розгортання нових версій наших програм, щоб переконатися, що продуктивність не погіршується.

*White-box testing*, яке іноді називають clear-box або transparent testing. Покладається на знання розробників тесту про програмне забезпечення, яке тестується, для створення тестових випадків. За допомогою White-box testing розробник тесту знає, як працює код, і може написати тестові випадки, які використовують розуміння роботи.

*Black-box tests* пишуться з усвідомленням того, що має робити програма, її вимогами чи специфікаціями, але не з усвідомленням того, як вона це робить.

Наприклад, простий Black-box test може полягати в тому, щоб перевірити, що коли ви вводите www.google.de у своєму браузері, повертається сторінка пошуку Google для Німеччини.

# Junior DevOps - Development

## 57. Як вирішити merge conflict для Git? (Розв'язання конфлікту злиття)

*Огляд конфлікту:* Почніть з огляду конфлікту. Git повідомить вас про конфлікт при спробі злиття гілок. Ви побачите позначення конфліктних файлів у ваших файлових менеджерах або в командному рядку.

*Редагування конфліктних файлів:* Відкрийте конфліктні файли у вашому улюбленому текстовому редакторі. Git позначить конфліктні ділянки у файлі, які розділені між

"<<<<<<<", "=======" та ">>>>>>>".

*Видалення зайвих міток:* Видаліть зайві мітки та залиште тільки потрібні зміни. Наприклад, виберіть, яку версію змінити, або об'єднайте зміни так, щоб вони відображали бажаний результат.

*Додавання та збереження змін:* Після вирішення конфлікту збережіть зміни у файлах, що були відредаговані.

*Виконання git add:* Після вирішення конфлікту відредагованих файлів, виконайте команду "git add" для відстеження змін.

## 57а. Що таке rebase для Git?

Це операція, яка дозволяє змінити історію комітів шляхом перебудови гілок або злиття комітів. Основна мета rebase - це змінити історію комітів так, щоб вона виглядала як єдина лінійна послідовність, без зайвих злиттів.

*Основні переваги rebase:*

* Після rebase історія комітів буде виглядати більш послідовною та чистою, оскільки зайві злиття будуть усунені.
* Після rebase злиття гілок буде легше, оскільки в історії не буде зайвих комітів.

Проте, варто пам'ятати, що rebase може призвести до конфліктів злиття, якщо відбулися зміни в тих самих частинах коду, що редагувалися при перебудові історії комітів.

|  |  |
| --- | --- |
| Було | Стало |
| A---B---C topic  /  D---E---F---G master | A'--B'--C' topic  /  D---E---F---G master |

Ось коли ‘git rebase’ може бути корисним:

*Перед злиттям гілок:* Ви можете виконати rebase вашої робочої гілки на головну гілку (зазвичай "master" або "main") перед злиттям, щоб зберегти чистоту історії комітів та уникнути зайвих злиттів.

*Після отримання змін з головної гілки:* Якщо ви працюєте у гілці функціональності і головна гілка отримала зміни, ви можете виконати rebase, щоб зберегти вашу гілку актуальною.

*Виправлення історії комітів*: Якщо ви хочете змінити історію вашої робочої гілки, наприклад, переписати коміти, об'єднати коміти або видалити зайві коміти, ви можете використати rebase.

*При роботі з віддаленими гілками:* Під час спільної роботи з віддаленими гілками, ви можете використовувати rebase для оновлення своєї гілки до останнього стану віддаленої гілки перед відправленням змін.

Ось деякі випадки, коли не рекомендується використовувати "git rebase":

*Гілку використовують інші користувачі*: Якщо ваша робоча гілка була відправлена на віддалений репозиторій і інші користувачі використовують цю гілку, варто утриматися від виконання rebase, оскільки це може призвести до проблем зі злиттям та ускладнить спільну роботу.

*Після публікації змін*: Якщо ви вже відправили зміни на віддалений репозиторій, не рекомендується виконувати rebase, оскільки це може призвести до непорозумінь, конфліктів та непослідовності історії комітів для інших користувачів.

*Після злиття гілок*: Якщо ви вже злили гілку з іншою гілкою (наприклад, з гілкою "master"), не рекомендується виконувати rebase, оскільки це може призвести до складнощів з історією комітів та втрати дійсної історії розвитку проекту.

*При великій кількості конфлікті*в: Якщо під час rebase виникає велика кількість конфліктів злиття, це може стати непродуктивним та призвести до втрати часу. У такому випадку може бути краще використати інші методи злиття гілок, такі як "git merge".

## 57б. Що таке cherry-pick для Git?

Це процес вибору конкретного коміту з однієї гілки та застосування його до поточної робочої гілки. Це дозволяє вам вибирати окремі коміти з інших гілок та застосовувати їх до вашої поточної робочої гілки без необхідності злиття всієї гілки.

Ось приклад використання cherry-pick:

* Спочатку потрібно визначити хеш (ідентифікатор) коміту, який ви хочете вибрати для cherry-pick. Ви можете знайти цей хеш, подивившись історію комітів за допомогою команди "git log".
* Потім ви виконуєте команду "git cherry-pick ", де - це хеш вибраного коміту. Ця команда створить новий коміт у вашій робочій гілці, який містить зміни з вибраного коміту.

Cherry-pick може бути корисним, коли вам потрібно вибрати лише окремі зміни з іншої гілки та застосувати їх до поточної гілки без зайвих змін.

Є ситуації, коли застосування cherry-pick **не є найкращим вибором**:

Багато комітів з однієї гілки: Якщо вам потрібно вибрати багато комітів з однієї гілки та застосувати їх до іншої, використання cherry-pick може стати непродуктивним та часомістким. У цьому випадку може бути краще розглянути злиття гілок або інші методи злиття змін.

Конфлікти злиття: Якщо ви вибираєте коміти, які змінюють ті ж самі рядки коду, це може призвести до конфліктів злиття під час cherry-pick. У таких випадках може знадобитися ручна корекція конфліктів, що ускладнює процес.

Публічні гілки: Якщо ви працюєте з публічними гілками, такими як "master", застосування cherry-pick може призвести до непослідовності історії комітів та ускладнити спільну роботу над проектом.

Застосування до старих комітів: Якщо ви спробуєте використати cherry-pick для застосування старих комітів до робочої гілки, це може призвести до проблем з контекстом та суперечностей з історією комітів.

## 58. У чому різниця між git merge та git rebase?

Різниця полягає в тому, як вони об'єднують зміни з однієї гілки в іншу:

Git Merge:

* Під час злиття гілок за допомогою git merge Git створює новий коміт, який об'єднує зміни з об'єднуваної гілки в поточну гілку. Це призводить до створення додаткового коміту з повідомленням "Merge" у історії комітів.
* Злиття за допомогою git merge зберігає структуру гілок та історію комітів кожної гілки без змін.

Git Rebase:

* Під час перебазування гілок за допомогою git rebase Git бере зміни з поточної гілки та застосовує їх до кінця обраної гілки. Це призводить до створення нових комітів на основі змін з об'єднуваної гілки.
* Після перебазування історія комітів виглядає так, ніби зміни з поточної гілки були зроблені безпосередньо на кінці обраної гілки, що робить історію комітів більш чистою та без "Merge" комітів.

Отже, git merge створює додатковий коміт з об'єднанням, тоді як git rebase застосовує зміни безпосередньо до кінця обраної гілки, створюючи нові коміти.

## 59. Які UI використовували?

*Графічний інтерфейс користувача (GUI):*

GUI використовує графічні елементи, такі як кнопки, меню та вікна, для взаємодії з користувачем. Найпоширеніші GUI включають Microsoft Windows, macOS, Linux з різними середовищами робочого столу, а також мобільні операційні системи, такі як iOS та Android.

*Командний рядок (CLI):*

CLI дозволяє користувачам взаємодіяти з системою шляхом введення текстових команд. Прикладами є командний рядок Windows Command Prompt, Unix-подібні термінали (наприклад, Terminal в macOS та Linux), а також інтерфейси для роботи з Git, npm, та іншими інструментами.

*Веб-інтерфейс (Web UI):*

Веб-інтерфейси дозволяють користувачам взаємодіяти з веб-сайтами та веб-додатками через браузер. Це можуть бути стандартні веб-сайти, віддалені панелі управління, хмарні сервіси та багато іншого.

*Мобільний інтерфейс (Mobile UI):*

Мобільні інтерфейси призначені для взаємодії з користувачем на мобільних пристроях, таких як смартфони та планшети. Це включає мобільні додатки та операційні системи, такі як iOS та Android.

*Голосові інтерфейси (Voice UI):*

Голосові інтерфейси дозволяють користувачам взаємодіяти з системою за допомогою голосових команд. Наприклад: Siri, Google Assistant та Amazon Alexa.

## 60. Яка різниця між GitLab, GitHub та Bitbucket?

Різниця між GitLab, GitHub та Bitbucket полягає в їхніх можливостях та функціоналі:

**GitHub:**

* GitHub є однією з найпопулярніших платформ для хостингу проектів з використанням системи контролю версій Git. Він широко використовується відкритими джерелами та комерційними проектами.
* Орієнтований на спільноту, у GitHub є можливість співпраці, відстеження проблем, ведення Wiki, а також інтеграція з різними іншими інструментами розробки.
* GitHub надає можливості для безкоштовного хостингу відкритих проектів, а також пропонує платні плани для приватних репозиторіїв.

**GitLab:**

* GitLab є рішенням для керування життєвим циклом програмного забезпечення, яке включає в себе не лише систему контролю версій Git, але й інші інструменти, такі як інтеграція CI/CD, відстеження проблем, відправлення змін та багато іншого.
* Окрім хостингу коду, GitLab надає можливості для управління вимогами, планування, розгортання та моніторингу програмного забезпечення.
* GitLab пропонує як хмарні, так і самостійні рішення для розгортання на власних серверах.

**Bitbucket:**

* Bitbucket належить до компанії Atlassian та надає хмарне та самостійне рішення для керування Git-репозиторіями.
* Bitbucket також має вбудовані інструменти для спільної роботи, відстеження проблем, інтеграції з іншими продуктами Atlassian (наприклад, Jira, Confluence).
* Bitbucket також пропонує можливості для безкоштовного хостингу відкритих проектів та платні плани для приватних репозиторіїв.

Хоча всі три платформи надають можливості для хостингу Git-репозиторіїв, вони відрізняються за своїми функціями, спрямованістю та умовами використання.

## 61. Яка різниця між Git pull/Git fetch?

Різниця у їхній функціональності та впливі на локальний репозиторій:

**git fetch:**

* Команда git fetch витягує всі дані з віддаленого репозиторію, але не об'єднує їх з вашою робочою копією. Це означає, що ви завантажуєте всі нові гілки, коміти та зміни, але вони не будуть автоматично об'єднані з вашим поточним робочим станом.
* Після виконання git fetch ви можете переглянути завантажені зміни та вирішити, як їх об'єднати з вашою робочою гілкою.

**git pull:**

* Команда git pull виконує дві операції: вона витягує дані з віддаленого репозиторію за допомогою git fetch, а потім автоматично об'єднує ці зміни з вашою поточною гілкою за допомогою git merge або git rebase, в залежності від налаштувань.
* git pull використовується для одержання оновлень з віддаленого репозиторію та автоматичного об'єднання з вашим поточним робочим станом без додаткових кроків.

Отже, основна різниця між git fetch та git pull в тому, що git fetch лише завантажує дані з віддаленого репозиторію, тоді як git pull виконує завантаження та автоматичне об'єднання цих змін з вашим поточним робочим станом.

## 62. Що таке Git-flow?

Git-flow - це *методологія* роботи з гілками у Git, яка була розроблена інженером Вінсентом Дріссеном і стала популярною у спільноті розробників. Git-flow використовує певну структуру гілок для керування різними аспектами розробки програмного забезпечення.

Основні ідеї методології Git-flow включають наступне:

*Головна гілка (master)*: Головна гілка використовується для публікації стабільних версій програмного забезпечення. Всі зміни, які потрапляють до цієї гілки, повинні бути готові до випуску.

*Гілка розробки (develop):* Гілка розробки є основною гілкою, з якої створюються інші гілки для розробки нового функціоналу. Вона використовується для інтеграції різних функцій та виправлень помилок перед тим, як вони потрапляють до головної гілки.

*Функціональні гілки (feature branches):* Кожна нова функція або велика зміна розробляється у своїй власній гілці, яка виходить з гілки розробки. Після завершення розробки функції вона об'єднується з гілкою розробки.

*Гілки виправлення помилок (bugfix branches):* Якщо виникають помилки, їх виправлення відбувається у відповідних гілках виправлення помилок, які також базуються на гілці розробки.

*Гілки релізу (release branches):* Перед випуском нової версії вона підготовлюється у відповідній гілці релізу, де можуть вноситися остаточні виправлення та підготовка до випуску.

*Гілки підтримки (hotfix branches):* У випадку виникнення критичних помилок у випущеній версії, для їх виправлення використовується гілка підтримки, яка базується на головній гілці.

Git-flow надає структурований підхід до роботи з гілками у Git, що допомагає управляти розвитком програмного забезпечення, використовуючи окремі гілки для різних типів роботи та забезпечуючи стабільність та організацію у процесі розробки.

## 63. Версіонування. Яка різниця між SemVer та CalVer?

Різниця в підході до нумерації версій програмного забезпечення:

**SemVer (семантичне версіювання):**

* Формат: SemVer використовує три числа у форматі "X.Y.Z", де X - це номер головної версії, Y - номер мінорної версії, а Z - номер патча. Наприклад, "1.2.3".
* Суть: У SemVer зміни у версіях програмного забезпечення розглядаються з точки зору сумісності з попередніми версіями. Зміна головної версії вказує на неповну сумісність з попередніми версіями, зміна мінорної версії вказує на додання нового функціоналу зі збереженням сумісності, а зміна патча вказує на виправлення помилок без зміни сумісності.

**CalVer (календарне версіювання):**

* Формат: CalVer використовує дату для нумерації версій, наприклад "2023.05.01" або "2023.1".
* Суть: У CalVer версії пов'язані з конкретною датою, що дозволяє легко визначати порядок та вік кожної версії. Зазвичай в CalVer також можуть бути додаткові розділи для включення інших метаданих, таких як передвипускові версії або виправлення помилок.

Отже, основна відмінність полягає в тому, що SemVer використовує числовий підхід для вказівки змін та сумісності версій, тоді як CalVer використовує дату для нумерації версій, що дозволяє зрозуміти порядок та вік кожної версії.

## 64. Які існують види тестування?

Докладно види тестів дивись аркуш 46

56. Які види тестів програмного забезпечення існують і навіщо вони потрібні?

Види тестування:

Модульне тестування (Unit Testing): Перевіряє окремі модулі або компоненти програми. Метою модульного тестування є виявлення помилок у конкретних функціях або методах програми.

Інтеграційне тестування (Integration Testing): Спрямоване на перевірку взаємодії між різними модулями чи компонентами програми. Він допомагає виявити помилки, що виникають під час інтеграції окремих частин програми.

Системне тестування (System Testing): Виконується на весь продукт як цілісну систему з метою підтвердження відповідності програми вимогам та очікуванням користувачів.

Приймальне тестування (Acceptance Testing): Виконується з метою перевірки відповідності програми вимогам замовника або користувача. Його проводять зазвичай замовники або представники користувачів.

Навантажувальне тестування (Load Testing): Визначає, як програмне забезпечення працює під навантаженням, перевіряючи його продуктивність та стійкість за різних умов навантаження.

Тестування безпеки (Security Testing): Спрямоване на виявлення потенційних уразливостей та захист програми від зловмисних атак.

Тестування користувацького інтерфейсу (UI Testing): Перевіряє взаємодії користувача з програмою для перевірки коректності та зручності користувацького інтерфейсу.

## 64а. Як писати тести програмного забезпечення?

При написанні тестів програмного забезпечення важливо дотримуватися кількох кроків, які допоможуть забезпечити якість тестів і ефективність процесу тестуванн

*Розуміння вимог:* Перш за все, важливо ретельно розібратися в вимогах до програмного забезпечення. Це допоможе зрозуміти очікуваний функціонал і результати роботи програми.

*Планування тестів:* Створення плану тестування, включаючи визначення обсягу тестування, виділення пріоритетів для різних частин програми, створення тестових сценаріїв та вибір тестових кейсів.

*Створення тестових кейсів:* Написання конкретних тестових кейсів для кожного функціонального або нефункціонального вимоги. Тестові кейси повинні бути якнайбільш повними та охоплювати різні можливі сценарії використання програми.

*Реалізація тестів:* Виконання написаних тестових кейсів на програмному забезпеченні з метою перевірки його працездатності та відповідності вимогам.

*Автоматизація тестів:* Там, де це можливо, варто автоматизувати тестування для забезпечення його ефективності та швидкості.

*Аналіз результатів:* Оцінка результатів тестування, виявлення помилок та несоответствий вимогам, уточнення тестових кейсів у відповідності до виявлених проблем.

Загалом, написання тестів програмного забезпечення вимагає систематичного та уважного підходу, а також використання спеціалізованих інструментів тестування для забезпечення якості тестів.

## 64б. Що таке Testing Drive Development (TDD)?

Test-Driven Development (TDD), або "розробка через тестування", є методологією розробки програмного забезпечення, яка базується на написанні тестів перед написанням фактичного коду програми. Основні принципи TDD :

Написання тесту: Розробник спочатку пише тест, який описує очікувану поведінку функції чи модуля програми. Це допомагає чітко сформулювати очікуваний результат від коду.

Запуск тесту на невдалий результат: Початковий тест повинен провалитися, оскільки код, який його виконує, ще не написаний чи не пройшов перевірку.

Написання коду: Після написання тесту розробник починає писати код, який дозволить пройти тестування та вирішити проблему, що описана в тесті.

Запуск тесту на успішний результат: Після написання коду, розробник запускає тест знову. Якщо тест пройдений успішно, це означає, що написаний код вирішує поставлену проблему.

Рефакторинг коду: Після успішного проходження тесту розробник може внести необхідні зміни до коду, щоб зробити його більш читабельним, ефективним та оптимізованим.

Цей цикл повторюється для кожної нової функції або модуля програми. TDD сприяє покращенню якості програмного забезпечення, оскільки він допомагає виявляти та виправляти помилки на ранніх етапах розробки, а також забезпечує наявність підтвердження коректної роботи коду у вигляді тестів.

## 65. У чому різниця між компільованими та інтерпретаційними мовами програмування?

Різниця програмування полягає у способі, яким вони обробляються та виконуються комп'ютером:

*Компільовані мови програмування:*

В компільованих мовах програмування програмний код перетворюється повністю в машинний код (або код байткоду) для виконання в середовищі якоїсь операційної системи під час процесу компіляції.

Результатом компіляції є виконуваний файл, який можна запустити без необхідності компіляції на кожному комп'ютері, де він повинен виконуватися.

Приклади компільованих мов програмування: C, C++, Go, Rust.

*Інтерпретаційні мови програмування:*

В інтерпретаційних мовах програмування програмний код виконується рядок за рядком інтерпретатором або віртуальною машиною.

Виконуваний код генерується безпосередньо в процесі виконання програми, без створення окремого виконуваного файлу.

Приклади інтерпретаційних мов програмування: Python, Ruby, JavaScript, PHP.

Програми-сценарії, написані інтерпретаційними мовами, можуть запускатися без суттєвих змін на кількох різних операційних системах.

# Junior DevOps - Monitoring/Loggin

## 66. Які метрики треба найбільше цікавлять DevOps?

Ось деякі з найбільш цікавих метрик для команд DevOps:

*Час розгортання (Deployment Frequency*): Ця метрика вимірює, як часто новий код або зміни вносяться до продуктивного середовища. Висока частота розгортання може свідчити про швидкі та ефективні процеси розробки та розгортання.

*Час відновлення (Mean Time to Recover, MTTR):* MTTR визначає час, необхідний для відновлення робочого стану системи після виникнення помилки чи збою. Низький MTTR свідчить про швидку відновлюваність системи після неполадок.

*Час розробки (Lead Time):* Ця метрика вимірює час, який потрібно для розробки нового функціоналу від постановки завдання до впровадження його в продуктивне середовище. Мінімізація часу розробки може свідчити про ефективність процесів розробки.

*Ступінь автоматизації (Automation Rate):* Ця метрика визначає, який відсоток процесів розробки та розгортання автоматизований. Високий рівень автоматизації може сприяти підвищенню ефективності та зниженню ризику помилок.

*Стабільність системи (System Stability):* Ця метрика вимірює кількість помилок, збоїв чи відмов в продуктивному середовищі. Вона допомагає визначити стабільність системи та якість впровадженого коду.

Ці метрики можуть допомогти командам DevOps відстежувати та оцінювати ефективність їхніх процесів розробки та розгортання, а також виявляти можливості для поліпшення.

## 66а. Метрики навантаження операційної системи

*Використання процесора (CPU Usage):* Вказує на відсоток часу, протягом якого процесор використовується для виконання обчислень. Високе використання процесора може свідчити про перевантаження системи.

*Використання пам'яті (Memory Usage):* Вказує на обсяг фізичної пам'яті, який використовується програмами та операційною системою. Занадто велике використання пам'яті може вплинути на продуктивність системи.

*Використання дискового простору (Disk Usage):* Вказує на обсяг зайнятого дискового простору. Перевищення максимального обсягу дискового простору може призвести до проблем з зберіганням даних та виконанням програм.

*Мережевий трафік (Network Traffic):* Вказує на обсяг даних, які передаються через мережу в системі. Великий обсяг мережевого трафіку може вказувати на значний обсяг комунікації між системою та іншими пристроями чи сервісами.

*Час відповіді (Response Time):* Вказує на час, який система витрачає на обробку запитів користувачів. Збільшення часу відповіді може свідчити про перевантаження системи чи неефективність програмного забезпечення.

The **free** command displays the total amount of free and used memory, as well as the buffers and caches used by the kernel and swap memory in the system.

$ free

The **watch** command can be useful to monitor memory usage with updates at regular intervals.

$ watch -n 5 free

The **vmstat** command provides detailed statistics about memory usage.

$ vmstat -s

Also you can use the **gnome-system-monitor** application: Press Alt + F2, type gnome-system-monitor

## 66б. Яка різниця між infrastructure monitoring та application monitoring.

*Infrastructure Monitoring:*

Спрямований на відстеження та аналіз різноманітних компонентів фізичної та віртуальної інфраструктури, таких як сервери, мережеве обладнання, сховища даних тощо.

Цей тип моніторингу зазвичай фокусується на параметрах, таких як використання CPU, обсяг доступної пам'яті, стан мережі, навантаження на сервери та інші аспекти, пов'язані з функціонуванням самої інфраструктури.

Мета - забезпечити високу доступність, ефективне використання ресурсів та вчасне виявлення проблем, що виникають на рівні інфраструктури.

*Application Monitoring:*

Більше спрямований на відстеження та аналіз функціонування окремих додатків або сервісів, які працюють на інфраструктурі.

Основний акцент робиться на моніторингу виконання додатків, якість обслуговування клієнтів, швидкість відповіді системи, обсяги запитів тощо.

Мета - забезпечити високу продуктивність, надійність та ефективність роботи конкретних додатків або сервісів.

Отже, infrastructure monitoring спрямований на стан самої інфраструктури, тоді як application monitoring - на функціонування окремих додатків або сервісів, що працюють на цій інфраструктурі.

## 67. Яка різниця між pull та push model у системах моніторингу?

*Pull-модель:*

В pull-моделі, також відомій як polling, агенти на кінцевих пристроях або серверах ініціюють процес збору даних.

Це означає, що центральна система моніторингу (наприклад, моніторинговий сервер) відправляє запити на дані до кожного пристрою чи сервера, і ті відповідають, надсилаючи необхідну інформацію.

Pull-модель часто використовується в традиційних системах моніторингу, де агенти регулярно перевіряють стан систем та повідомляють процесу моніторингу.

*Push-модель:*

У push-моделі, системи або агенти на кінцевих пристроях надсилають дані безпосередньо до центральної системи моніторингу.

Центральна система не ініціює отримання даних, вона лише очікує на їхнє надходження від кінцевих пристроїв.

Push-модель часто використовується в сучасних системах моніторингу, особливо в хмарних сервісах та мікросервісних архітектурах, де пристрої та сервіси можуть активно надсилати дані про свій стан.

Отже, в pull-моделі дані витягуються з кінцевих пристроїв центральною системою моніторингу, тоді як у push-моделі дані надсилаються кінцевими пристроями до центральної системи.

## 68. Яка різниця між Black box та White box monitoring?

*Black Box Monitoring:*

Розглядає систему як чорний ящик, тобто зовнішній спостерігач не має доступу до внутрішньої структури чи деталей системи.

Оцінка проводиться на основі зовнішніх вимірів, таких як доступність, продуктивність, відповідь на запити, швидкість завантаження тощо, без доступу до внутрішніх механізмів роботи системи.

Цей підхід часто використовується для моніторингу зовнішніх служб, веб-сайтів, API та інших компонентів, коли важливо знати, як система функціонує з точки зору зовнішніх користувачів.

*White Box Monitoring:*

Це коли спостерігач має повний доступ до внутрішньої структури та деталей системи, включаючи код, конфігурації, бази даних тощо.

Оцінка проводиться на основі внутрішніх процесів, показників продуктивності, використання ресурсів та інших метрик, які можуть бути доступні тільки зсередини системи.

Цей підхід корисний для моніторингу внутрішніх параметрів систем, таких як використання пам'яті, швидкість виконання запитів до бази даних, стан серверів тощо.

Отже, основна відмінність полягає у тому, що в black box monitoring оцінка базується на зовнішніх вимірах, в той час як в white box monitoring використовуються внутрішні дані та метрики системи.

## 69. Розкажіть про підходи до збору application logs.

*Direct Log File Access:*

Цей підхід полягає в безпосередньому доступі до лог-файлів додатків, які зазвичай знаходяться на файловій системі сервера. Лог-файли можуть бути зчитані напряму за допомогою програм або агентів, які моніторять та збирають дані для подальшого аналізу.

*Logging Libraries:*

Багато мов програмування мають вбудовані бібліотеки для збору логів. Розробники можуть використовувати ці бібліотеки для записування різних подій та стану додатків у спеціалізовані лог-файли.

Цей підхід дозволяє додаткам створювати власні лог-записи у форматі, який вони визначають, що може спростити аналіз додатків.

*Centralized Logging Systems:*

Передбачає використання спеціалізованих систем централізованого збору логів, таких як Elasticsearch, Splunk, або Fluentd. Додатки надсилають свої логи до цих централізованих систем для збору, збереження та аналізу.

Це дозволяє агрегувати логи з різних додатків та серверів, виконувати пошук за ключовими словами, аналізувати та візуалізувати дані для виявлення проблем.

*Containerized Logging:*

У віртуалізованих середовищах, таких як Docker, Kubernetes, лог-файли можуть бути збирані з контейнерів та сервісів, що виконуються у цих середовищах.

Спеціалізовані рішення, такі як Fluentd, Logstash або Docker Logging Drivers, можуть бути використані для збору логів з контейнерів та їх подальшої обробки.

Ці підходи можуть використовуватися окремо або в поєднанні один з одним для забезпечення повного та ефективного збору логів додатків.

## 70. Що таке Prometheus, і як воно працює з метриками

Prometheus - це система моніторингу та тривоги, яка відкрита та відмінно підходить для збору метрик і виявлення проблем у реальному часі. Ось як воно працює з метриками:

*Збір метрик:*

Exposition: Додатки, що моніторяться, можуть "виставляти" свої метрики за допомогою HTTP ендпоінтів. Це може бути зроблено за допомогою бібліотек для різних мов програмування, які дозволяють експонувати метрики у відповідному форматі.

Scraping: Prometheus звертається до цих HTTP ендпоінтів періодично (зазвичай кожні 15-30 секунд) та збирає метрики, які вони виставляють.

*Збереження метрик:*

Часові ряди: Prometheus зберігає метрики у внутрішній базі даних у вигляді часових рядів. Це дозволяє аналізувати метрики протягом певного періоду часу, виявляти тенденції та аномалії.

Централізоване зберігання: Всі зібрані метрики зберігаються централізовано у Prometheus, що дозволяє легко виконувати запити та аналізувати дані.

*Запити та візуалізація:*

PromQL: Prometheus надає мову запитів PromQL для виразного та потужного аналізу метрик. Це дозволяє користувачам складати складні запити для виявлення тенденцій, виконання агрегацій та фільтрації даних.

Візуалізація: Метрики можуть бути візуалізовані за допомогою інтеграції Prometheus з іншими інструментами візуалізації, такими як Grafana, які дозволяють створювати графіки та панелі для моніторингу систем.

*Тривоги:*

Alertmanager: Prometheus може налаштовувати тривоги на основі визначених правил, які спрацьовують при виявленні аномалій або перевищенні порогових значень метрик.

У світі систем моніторингу існує кілька **аналогів** **Prometheus**, які також надають рішення для збору, зберігання та візуалізації метрик, а також спрацювання тривог. Ось декілька з них:

Grafana: Це популярний інструмент візуалізації, який часто використовується разом з Prometheus для створення красивих та інформативних графіків, панелей та іншіх візуальних елементів для моніторингу метрик.

InfluxDB: Це розподілена система збереження часових рядів, яка може використовуватися для збереження метрик, зібраних за допомогою Prometheus. В поєднанні з іншими інструментами, такими як Telegraf та Grafana, InfluxDB може надати повноцінний стек моніторингу.

Zabbix: Це рішення для моніторингу мережі та серверів, яке також має можливості для збору метрик, спрацювання тривог та візуалізації даних. Зазвичай використовується для моніторингу великих і складних інфраструктур.

Prometheus Operator: Цей інструмент дозволяє автоматизувати розгортання та керування вузлами Prometheus та іншими компонентами в Kubernetes.

Nagios: Це одне з найпоширеніших рішень для моніторингу інфраструктури та додатків. Воно надає можливості для збору метрик, спрацювання тривог та моніторингу різних аспектів системи.

Ці інструменти можуть використовуватися окремо або в поєднанні для створення потужних та гнучких систем моніторингу для різних сценаріїв та потреб.

# Junior DevOps - Практичні завдання

## 71а. Напишіть просту програму на Python. Програма має отримувати повідомлення з сервісу черг і друкувати його в stdout. Сервіс черг — на ваш розсуд.

**RabbitMQ** - це популярний сервіс черги повідомлень, який використовується для обміну повідомленнями між різними компонентами програмного забезпечення. Основна мета використання RabbitMQ полягає в забезпеченні надійної та ефективної комунікації між різними додатками або мікросервісами.

**Основні причини використання RabbitMQ сервісу:**

Асинхронна комунікація: RabbitMQ дозволяє відправляти повідомлення асинхронно, що означає, що відправник і отримувач можуть працювати незалежно один від одного. Це дозволяє покращити продуктивність та масштабованість системи.

Розподілена обробка: RabbitMQ дозволяє розподіляти завдання між багатьма обробниками. Він може бути використаний для розподіленої обробки великих обсягів даних або для розподіленої обробки завдань у мікросервісній архітектурі.

Надійна доставка повідомлень: RabbitMQ забезпечує надійну доставку повідомлень, що означає, що вони не втрачаються або не дублюються. Це особливо важливо в системах, де критично важливо, щоб повідомлення були доставлені без помилок.

Гнучкість: RabbitMQ підтримує різні протоколи комунікації, такі як AMQP (Advanced Message Queuing Protocol), MQTT (Message Queuing Telemetry Transport) та інші. Це дозволяє використовувати RabbitMQ в різних сценаріях, включаючи Інтернет речей (IoT), мобільні додатки та багато іншого.

Масштабованість: RabbitMQ може бути легко масштабований для виконання великої кількості повідомлень. Він підтримує горизонтальне масштабування, що дозволяє розподіляти навантаження між кількома серверами.

**Напишіть просту програму на Python. Програма має отримувати повідомлення з сервісу черг RabbitMQ і друкувати його в stdout.**

-------------------------------------------------------------------------------------------------------

import pika

# Підключення до RabbitMQ сервера

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))

channel = connection.channel()

# Створення черги з ім'ям 'hello'

channel.queue\_declare(queue='hello')

# Функція, яка викликається при отриманні повідомлення

def callback(ch, method, properties, body):

print(" [x] Received %r" % body)

# Підписуємося на чергу 'hello' та вказуємо функцію-обробник

channel.basic\_consume(queue='hello', on\_message\_callback=callback, auto\_ack=True)

print(' [\*] Waiting for messages. To exit press CTRL+C')

channel.start\_consuming()

-------------------------------------------------------------------------------------------------------

Для виконання цієї програми знадобиться встановлення бібліотеки pika:

$ pip install pika.

Зверніть увагу, що в прикладі використовується локальний RabbitMQ сервер, тому ви можете змінити параметри підключення за необхідності.

RabbitMQ та CloudAMQP є пов'язаними, але використовуються в різних випадках^

**RabbitMQ**:

* це програмне забезпечення для черги повідомлень, також відоме як брокер повідомлень або менеджер черг. Він дозволяє визначати черги, в які можна поміщати повідомлення для подальшої обробки.
* є відкритим джерелом та підтримує багато мов програмування та протоколів комунікації.
* dи можете встановити та налаштувати свій власний сервер RabbitMQ на своєму обладнанні або в хмарному середовищі.

**CloudAMQP**:

* це хмарна послуга, яка автоматизує налаштування, експлуатацію та масштабування серверів RabbitMQ та LavinMQ.
* dи можете використовувати CloudAMQP для швидкого розгортання та керування вашими чергами повідомлень без необхідності власного налаштування та управління серверами.
* CloudAMQP підтримує різні хмарні платформи, такі як Amazon Web Services (AWS), Google Cloud Platform (GCP) та інші.

Отже, RabbitMQ - це самостійне програмне забезпечення для черги повідомлень, тоді як CloudAMQP - це хмарна послуга, яка автоматизує розгортання та керування серверами RabbitMQ.

## 71б. Напишіть просту програму на Bash. Програма має отримувати повідомлення з сервісу черг і друкувати його в stdout. Сервіс черг — на ваш розсуд.

Ось невеликий приклад програми на мові Bash, яка використовує утиліту curl для отримання повідомлень з вказаного сервісу черги повідомлень CloudAMQP та друкує їх в stdout.

-------------------------------------------------------------------------------------------------------

#!/bin/bash

# URL до сервісу черги

queue\_url="YOUR\_QUEUE\_URL\_HERE"

# Отримання повідомлення з сервісу черги за допомогою HTTP GET запиту

message=$(curl -s $queue\_url)

# Друк отриманого повідомлення в stdout

echo $message

-------------------------------------------------------------------------------------------------------

У цій програмі використовується утіліта curl для виконання HTTP GET запиту до вказаного URL сервісу черги. Отримане повідомлення зберігається у змінній message, яка потім виводиться в stdout за допомогою команди echo.

Зверніть увагу, що в даному прикладі ви повинні замінити YOUR\_QUEUE\_URL\_HERE на фактичний URL вашого сервісу черги повідомлень.

**Сервіс черги повідомлень CloudAMQP** найчастіше використовують для реалізації асинхронного обміну повідомленнями між різними компонентами або мікросервісами в розподіленій системі для забезпечення:

* *Розсилки повідомлень*: CloudAMQP дозволяє відправляти повідомлення в чергу, які потім можуть бути оброблені асинхронно. Це особливо корисно, коли потрібно розсилати повідомлення багатьом отримувачам або коли час обробки повідомлення може зайняти тривалий час.
* *Мікросервісної архітектури*: CloudAMQP дозволяє реалізувати мікросервісну архітектуру, де кожен сервіс може взаємодіяти з іншими сервісами через чергу повідомлень. Це дозволяє забезпечити локальну обробку повідомлень кожним сервісом і зменшити залежність між компонентами системи.
* *Розподіленої обробки завдань*: CloudAMQP дозволяє розподіляти завдання між багатьма обробниками, що дозволяє покращити продуктивність та масштабованість системи. Кожен обробник може брати повідомлення з черги та обробляти їх паралельно.
* *Збереження повідомлень*: CloudAMQP забезпечує надійне збереження повідомлень у випадку, якщо отримувач не готовий їх обробити в даний момент. Це дозволяє уникнути втрати повідомлень та забезпечити надійну доставку.
* *Масштабованості*: CloudAMQP дозволяє легко масштабувати систему, додавши більше обробників або черг. Це дозволяє забезпечити високу продуктивність та швидку обробку повідомлень, навіть при збільшенні навантаження.

## 71в. Напишіть простий yaml для Ansible. Програма має отримувати повідомлення з сервісу черг RabbitMQ і друкувати його в stdout.

-------------------------------------------------------------------------------------------------------

---

- hosts: localhost

tasks:

- name: Install required RabbitMQ client library

pip:

name: pika

state: present

- name: Receive message from RabbitMQ and print to stdout

vars:

queue\_host: "YOUR\_RABBITMQ\_HOST"

queue\_user: "YOUR\_RABBITMQ\_USERNAME"

queue\_pass: "YOUR\_RABBITMQ\_PASSWORD"

queue\_name: "YOUR\_QUEUE\_NAME"

script: |

import pika

def callback(ch, method, properties, body):

print("Received message: %r" % body)

connection = pika.BlockingConnection(pika.ConnectionParameters(queue\_host, 5672, '/', pika.PlainCredentials(queue\_user, queue\_pass)))

channel = connection.channel()

channel.queue\_declare(queue=queue\_name)

channel.basic\_consume(queue=queue\_name, on\_message\_callback=callback, auto\_ack=True)

print('Waiting for messages. To exit press CTRL+C')

channel.start\_consuming()

-------------------------------------------------------------------------------------------------------

У цьому файлі конфігурації ми визначаємо один хост (localhost) та дві задачі:

* Встановлення необхідної бібліотеки RabbitMQ для Python за допомогою модуля pip.
* Отримання повідомлень з RabbitMQ та виведення їх в stdout за допомогою скрипту на мові Python. В цій задачі ми використовуємо бібліотеку pika для з'єднання з RabbitMQ, отримання повідомлень з черги та їх виведення в stdout.

Зверніть увагу, що в цьому YAML файлі вам необхідно замінити YOUR\_RABBITMQ\_HOST, YOUR\_RABBITMQ\_USERNAME, YOUR\_RABBITMQ\_PASSWORD та YOUR\_QUEUE\_NAME на фактичні значення вашого сервісу черги RabbitMQ.

Цей YAML файл можна використовувати для виконання задачі отримання повідомлень з сервісу черги RabbitMQ та їх виведення в stdout за допомогою Ansible.

## 72. Розберіть структуру сервісу Docker-compose.

Docker-compose - це інструмент, який дозволяє визначати та запускати багатоконтейнерні додатки в середовищі Docker. За допомогою Docker-compose ви можете описати конфігурацію вашого додатку у файлі YAML, який потім можна використовувати для створення та керування контейнерами.

Основною одиницею структури Docker-compose є сервіси, це окремі контейнери, які виконують певні функції вашого додатку. Кожен сервіс описується як окремий блок у файлі YAML і має свої властивості та параметри.

Основні елементи структури сервісу Docker-compose:

* version: Вказує версію Docker-compose, яку ви використовуєте.
* services: Це ключове слово, за яким слідує список сервісів вашого додатку. Кожен сервіс описується як окремий блок з іменем сервісу та його конфігурацією.
* image: Вказує образ Docker, який буде використовуватись для створення контейнера сервісу. Ви можете використовувати готові образи з Docker Hub або вказати свій власний образ.
* ports: Вказує порти, які будуть відкриті на контейнері та доступні зовнішнім системам. Наприклад, "80:80" означає, що порт 80 контейнера буде доступний на порту 80 хостової системи.
* volumes: Вказує шляхи до файлів або каталогів, які будуть монтуватись у контейнер. Це дозволяє зберігати дані між запусками контейнера та забезпечує персистентність.
* environment: Вказує змінні середовища, які будуть доступні у контейнері. Наприклад, ви можете встановити змінну "MYSQL\_ROOT\_PASSWORD" для контейнера бази даних MySQL.
* networks: Вказує мережі, до яких буде підключений контейнер. Це дозволяє контейнерам взаємодіяти між собою.

-------------------------------------------------------------------------------------------------------

**version**: "3.8"

**services**:

mongodb:

**image**: mongo

**volumes**:

- mongodbdata:/data/db

env\_file:

- ./env/mongo.env

container\_name: azmongodb

backend:

# build: ./backend # permitted syntax for exactly 'Dockerfile' name

build:

context: ./backend

dockerfile: Dockerfile

**volumes**:

- ./backend/src:/app/src

**ports**:

- 8000:8000

env\_file:

- ./env/backend.env

depends\_on:

- mongodb

container\_name: azbackend

frontend:

build:

context: ./frontend

dockerfile: Dockerfile

**volumes**:

- ./frontend/src:/app/src

**ports**:

- 3000:3000

stdin\_open: true

tty: true

depends\_on:

- backend

**volumes**:

mongodbdata:

-------------------------------------------------------------------------------------------------------

## 73. Практична сесія роботи з Git (Git command line: fetch, push, pull, rebase, checkout, submodules).

Git submodules - це функціональність, що дозволяє включати один репозиторій Git в інший репозиторій Git як підмодуль. Підмодуль - це посилання на конкретний коміт у зовнішньому репозиторії, яке зберігається в основному репозиторії.

Основна ідея використання git submodules полягає в тому, щоб мати можливість включати залежності з інших репозиторіїв у ваш проект. Це особливо корисно, коли ви хочете використовувати сторонні бібліотеки або модулі, які розробляються окремо.

Основні поняття та операції, пов'язані з git submodules:

* *Додавання підмодуля*: Ви можете додати підмодуль до вашого репозиторію за допомогою команди git submodule add. Ця команда додає посилання на зовнішній репозиторій та його коміт у вашому репозиторії.
* *Оновлення підмодуля*: Якщо зовнішній репозиторій, на який ви посилаєтесь як підмодуль, оновлюється, ви можете оновити його у вашому репозиторії за допомогою команди git submodule update. Ця команда оновлює підмодуль до останнього коміту у зовнішньому репозиторії.
* *Клонування репозиторію з підмодулями*: При клонуванні репозиторію, який містить підмодулі, вам потрібно виконати додаткові кроки, щоб отримати вміст підмодулів. Ви можете виконати команду git submodule init для ініціалізації підмодулів та git submodule update для отримання вмісту підмодулів.
* *Робота з підмодулями*: Підмодулі ведуть себе як окремі репозиторії Git. Ви можете перейти до папки підмодуля, виконати команди Git та здійснювати зміни в підмодулі як у звичайному репозиторії.
* *Оновлення підмодуля до нової версії*: Якщо зовнішній репозиторій, на який ви посилаєтесь як підмодуль, випускає нову версію, в