# 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 команду, 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 програмного забезпечення після його 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**

**Надійність**

**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.

**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 протоколу. Клієнт посилає запит до сервера, щоб отримати певну веб-сторінку або ресурс.

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

*Обробка запиту сервером:* Веб-сервер отримує HTTP-запит і обробляє його. Він 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.

**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 Code, для чого це потрібно і які проблеми вирішує?

**Infrastructure as Code (IaC)** - 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 IaC**:

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 (IaC) **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. IaC 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. IaC 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:* IaC 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 вашому запитанні ${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?

29.Для чого потрібні network masks?

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

31.Що таке колізія? Чому виникає?

32.Що таке проксі?

33.Що таке firewalls і навіщо вони потрібні?

34.Що таке NAT і навіщо він потрібен?

35.Які типи IP-адрес ви знаєте?

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

Clouds

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

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

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

Automation

40.Що таке IaaC і навіщо він потрібний?

41.Що таке Terraform?

42.Які інструменти автоматизації ви знаєте?

Information Security

43.У чому різниця між аутентифікацією та авторизацією?

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

45.Як безпечно передати дані своєму колезі?

46.Що таке MFA, TOTP?

Віртуалізація

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

48.Як для при запуску Docker-контейнера «повісити» його з 80-го порту в контейнері на 8081 на хост?

49.Як передати у віртуальну машину USB device?

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

CI/CD

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

52.Опишіть основні етапи CI/CD.

53.Опишіть приклад процесу CI (та/або CD), який починається з моменту, коли розробник запушив зміни/PR до Git?

54.Розкажіть про різновиди тестів, які ми можемо використовувати в CI пайплайні.

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

56.Які види тестів ви знаєте і навіщо вони потрібні?

Development

57.Git. Як вирішити merge conflict? Що таке rebase, cherry-pick?

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

59.Які UI використовували?

60.Яка різниця між GitLab/GitHub/Bitbucket?

61.Яка різниця між Git pull/Git fetch?

62.Що таке Git-flow?

63.Версіонування. Яка різниця між SemVer та CalVer?

64.Тестування. Які існують види? Як писати тести, TDD?

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

Monitoring/Logging

66.Які метрики треба збирати? Різниця між infrastructure та application monitoring.

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

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

69.Розкажіть про підходи до збору application логів.

Практичні завдання

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

72.Розберіть структуру сервісу (на прикладі Docker-compose).

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