ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«ВПРОВАДЖЕННЯ МОНІТОРИНГУ СЕРВЕРА OPENVPN ЧЕРЕЗ ЙОГО ІНТЕГРАЦІЮ З СИСТЕМОЮ PROMETHEUS ТА PYTHON»**

на здобуття освітнього ступеня магістра
за спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Марія БРИКСІНА
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти група КСДМ-61
Марія БРИКСІНА
(ім'я, ПРІЗВИЩЕ)
Керівник: Андрій ЛЕМЕШКО
доктор філософії, (PhD) *(ім'я, ПРІЗВИЩЕ)*
Рецензент:
(ім'я, ПРІЗВИЩЕ)

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти Магістр

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бриксіній Марії Дмитрівні
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Впровадження моніторингу сервера OpenVPN через його інтеграцію з системою Prometheus та Python.

керівник кваліфікаційної роботи Андрій Лемешко к.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна документація OpenVPN, Prometheus, Python.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження систем моніторингу мережі.
2. Дослідження та аналіз технології VPN.
3. Впровадження моніторингу сервера OpenVPN через інтеграцію з системою Prometheus та Python.

5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-23.10.23	
2	Вивчення матеріалів для реалізації моніторингу мережі	23.10-27.10.23	
3	Дослідження систем моніторингу мережі	27.10-30.10.23	
4	Дослідження протоколів VPN	01.11-07.11.23	
6	Реалізація і тестування моніторингу сервера OpenVPN через інтеграцію з системою Prometheus та Python	07.11-27.11.23	
7	Оформлення роботи: вступ, висновки, реферат	27.11-30.11.23	
8	Розробка демонстраційних матеріалів	01.12-06.12.23	

Здобувач вищої освіти _____ Марія БРИКСІНА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Андрій ЛЕМЕШКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 89 стор., 0 табл., 49 рис., хх джерел.

Мета роботи – впровадження моніторингу сервера OpenVPN за допомогою Prometheus та Python для підвищення швидкості реакції на виникаючі інциденти на сервері.

Об'єкт дослідження – сервер OpenVPN.

Предмет дослідження – система моніторингу Prometheus та Python.

Короткий зміст роботи: У роботі досліджено та впроваджено моніторинг сервера OpenVPN за допомогою Prometheus з метою підвищення швидкості реакції на виникаючі інциденти на сервері. Робота включає аналіз теоретичних основ систем моніторингу мережі, протоколів VPN, налаштування та детальний опис практичного впровадження моніторингу мережі дослідження. Результати дослідження підтверджують ефективність технології та її позитивний вплив на вчасне виявлення проблем та можливість прийняття інформованих рішень IT-підприємствами.

КЛЮЧОВІ СЛОВА: OPENVPN, СИСТЕМИ МОНІТОРИНГУ СЕРВЕРА, PROMETHEUS, PYTHON, VPN.

ABSTRACT

Text part of the master's qualification work: 89 pages, 49 pictures, 0 table, xx sources.

The aim of the study is to implement monitoring of the OpenVPN server using Prometheus and Python to enhance the response speed to incidents on the server.

The object of the research is OpenVPN server.

The subject of the research is Prometheus monitoring system and Python.

Summary of the work: The paper explores and implements the monitoring of the OpenVPN server using Prometheus and Python with the goal of improving response speed to incidents on the server. The work includes an analysis of the theoretical foundations of network monitoring systems, VPN protocols, configuration, and a detailed description of the practical implementation of network monitoring in the study. The research results confirm the effectiveness of the technology and its positive impact on the timely detection of problems and the ability for IT enterprises to make informed decisions.

KEYWORDS: OPENVPN, СИСТЕМИ МОНІТОРИНГУ СЕРВЕРА, PROMETHEUS, PYTHON, VPN.

ЗМІСТ

ВСТУП

Протягом останньої чверті століття інформаційні технології суттєво змінили спосіб, яким функціонує сучасне суспільство і бізнес. У новому цифровому середовищі, де забезпечення безпеки та конфіденційності даних є невід'ємною частиною нашого сьогодення, стає важливим розробляти ефективні механізми для захисту мережевого зв'язку. Зростання кількості віддалених працюючих департаментів компаній і широке поширення переносних пристроїв роблять це завдання особливо актуальним.

У сучасному світі віртуальні приватні мережі (VPN) є невід'ємним елементом інфраструктури багатьох організацій. Вони дозволяють користувачам встановлювати зашифроване з'єднання з віддаленою мережею, надаючи безпеку та конфіденційність. Однак, як і будь-яка інша система, вона також потребує моніторингу. Моніторинг є важливим для забезпечення безперебійної роботи та виявлення можливих проблем.

На фоні стрімкого розвитку хмарних технологій та розширення масштабів інтернет-підключень, важливість ефективного моніторингу та управління серверами VPN набуває ще більшого значення. Впровадження системи моніторингу через інтеграцію із Prometheus та використання мови програмування Python не лише підвищить рівень безпеки, а й забезпечить оперативну відслідковуваність та реагування на потенційні проблеми в реальному часі.

Ця магістерська робота фокусується на впровадженні моніторингу сервера OpenVPN через інтеграцію з системою Prometheus та Python. Основна мета дослідження - вивчення можливостей системи моніторингу Prometheus, оцінка продуктивності та ефективності моніторингу сервера OpenVPN за допомогою цієї технології. Основні завдання роботи включають вивчення можливостей інтеграції системи Prometheus із сервером OpenVPN, розробку прототипу інфраструктури

1 АНАЛІЗ СИСТЕМ МОНІТОРИНГУ МЕРЕЖІ

1.1 Огляд систем моніторингу мережі

Система моніторингу - це комплекс програмних та апаратних засобів, призначених для нагляду, аналізу та контролю за різними аспектами функціонування об'єкта чи системи. У широкому сенсі термін "система моніторингу" може використовуватися для опису різних видів моніторингу, включаючи екологічний, фізичний, соціальний, інформаційний та інші.

У вузькому контексті інформаційних технологій система моніторингу відноситься до засобів, які слідкують за станом та діяльністю інформаційних систем, мереж, серверів, програм тощо. Основна мета такої системи - забезпечити оперативний контроль за всіма важливими параметрами та ресурсами, що впливають на ефективність та доступність інформаційних технологій.

Система моніторингу може включати в себе різноманітні функції, такі як:

Вимірювання ресурсів**:** Моніторинг включає вимірювання використання різних ресурсів, таких як центральний процесор, оперативна пам'ять, дисковий простір, мережевий трафік та інші.

Виявлення проблем та аномалій**:** Система моніторингу спроможна виявляти аномальні стани, помилки чи незвичайну активність в системі, що може свідчити про можливі проблеми або загрози.

Журналювання подій**:** Запис і аналіз журналів подій дозволяє відстежувати дії користувачів та події в системі, що допомагає в розслідуванні проблем та забезпечує безпеку.

Автоматизована реакція: Засоби автоматизованої реакції можуть бути використані для вжиття певних заходів у випадку виявлення певних умов або проблем.

Забезпечення доступності: Моніторинг може включати в себе вимірювання доступності сервісів та компонентів мережі з метою попередження витрат часу та реагування на відмови.

Звітність та аналітика: Система моніторингу може створювати звіти та аналітичні дані, які допомагають адміністраторам та керівництву робити обгрунтовані рішення щодо оптимізації інфраструктури та планування розвитку.

Незалежно від конкретного контексту використання, система моніторингу визначається своєю здатністю надавати оперативну інформацію про стан об'єкта моніторингу та забезпечувати зручні інструменти для аналізу та взаємодії з цією інформацією.

Сучасні організації все більше залежать від стабільної та ефективної мережевої інфраструктури для забезпечення своєї діяльності. У цьому контексті система моніторингу мережі виграє ключову роль у забезпеченні безперебійного функціонування та вчасного виявлення можливих проблем. Вона стає невід'ємною частиною інформаційно-технічного стеку будь-якої компанії, забезпечуючи адміністраторам інструментарій для моніторингу, аналізу та управління мережевою інфраструктурою.

Система моніторингу мережі повинна здатна автоматично відстежувати та аналізувати різноманітні параметри, включаючи пропускну спроможність, витрати ресурсів, затримку пакетів та інші ключові метрики. Це дозволяє оперативно реагувати на зміни в мережі та попереджати можливі перебої у роботі. Додатково, система моніторингу може забезпечувати детальну статистику щодо використання ресурсів, що дозволяє оптимізувати їх розподіл та раціоналізувати інфраструктуру.

Важливою складовою системи моніторингу є інтеграція з іншими інструментами управління мережею. Це може включати автоматизовану реакцію на виявлені проблеми, резервне копіювання конфігурацій, аналіз журналів подій та інші функції, що полегшують адміністрування та забезпечують високий рівень доступності мережі.

Загальний успіх системи моніторингу мережі полягає не лише в її технічних можливостях, але і в здатності відповідати конкретним потребам конкретної організації. При виборі такої системи слід враховувати масштаби та потужність мережі, особливості бізнес-процесів та готовність до масштабування інфраструктури в майбутньому. Тільки індивідуалізований підхід до розгортання та конфігурації дозволить забезпечити максимальну ефективність і надійність мережевого моніторингу в умовах постійних змін і розвитку технологій.

У сучасному бізнес-середовищі, де конкуренція постійно зростає, а технологічні вимоги зростають експоненційно, система моніторингу мережі виступає як ключовий елемент для забезпечення надійності та ефективності операцій. Зокрема, у зв'язку з розповсюдженням хмарних технологій та віртуалізації, важливість моніторингу зростає, оскільки виникають нові виклики для управління розподіленою мережевою інфраструктурою.

Однією з ключових функцій системи моніторингу мережі є здатність виявляти та прогнозувати аномалії в мережевому трафіку. Це стає особливо важливим у контексті кібербезпеки, де виявлення атак або несанкціонованого доступу може врятувати від серйозних наслідків. Моніторингова система повинна бути здатна реагувати на незвичайні зміни та надавати адміністраторам необхідні інструменти для вжиття відповідних заходів у випадку виявлення потенційно небезпечних ситуацій.

Однак ефективність моніторингу мережі не обмежується лише виявленням проблем. Ця система також має забезпечувати детальний аналіз роботи мережі, щоб адміністратори могли приймати обгрунтовані рішення щодо оптимізації ресурсів та підвищення продуктивності. Збір та аналіз данних про використання мережевих ресурсів, патернів трафіку та навантаження на окремі вузли дозволяє попереджати перевантаження та вчасно масштабувати інфраструктуру з метою забезпечення оптимальної продуктивності.

У сучасній бізнес-екосистемі, де мобільність персоналу стає нормою, система моніторингу повинна також забезпечувати віддалений доступ до неї. Це

дозволяє адміністраторам ефективно взаємодіяти з системою навіть здалеку, що дозволяє оперативно реагувати на проблеми та здійснювати конфігураційні зміни без необхідності фізичного присутності.

Крім того, інтеграція системи моніторингу мережі з іншими інструментами управління та автоматизації є важливою складовою. Автоматизовані сценарії реагування на події, такі як автоматичне відновлення послуг після збою або автоматичне масштабування ресурсів, полегшують рутинні завдання адміністрування та забезпечують більш ефективне використання інфраструктури.

1.2 Типи систем моніторингу мережі

Система моніторингу представляє собою необхідний елемент для ефективного управління та підтримки мережевої інфраструктури. Її завданням є періодична перевірка доступності та стану кожного вузла та елемента мережі, а також надання інформації про будь-які виявлені проблеми або недоступність відповідальній особі. Деякі системи моніторингу можуть навіть активно управляти мережею, надаючи можливість реагувати на зміни автоматично.

Типовий сценарій функціонування системи моніторингу включає в себе регулярні перевірки стану всіх елементів мережі. Це може бути здійснено за допомогою пінгування, тестування доступності ресурсів, аналізу трафіку та інших методів. У випадку виявлення будь-яких аномалій або недоступності, система моніторингу негайно повідомляє відповідальну особу чи адміністратора.

Існують різні типи систем моніторингу, які можна класифікувати за різними критеріями, такими як масштаб системи, методи вимірювань, ступінь автоматизації та інші. Один з можливих підходів - це розрізняти системи моніторингу за їхнім типом розміщення: вони можуть використовуватися як додаток на сервері або як індивідуальні пристрої.

Тип "додаток на сервері" передбачає встановлення програмного забезпечення моніторингу на сервері мережі. Це може бути ефективним для

великих підприємств або комплексних мереж, де централізоване управління моніторингом дозволяє забезпечити єдність та координацію заходів.

Іншим підходом є використання індивідуальних пристроїв для моніторингу. Це може бути корисним для менших мереж або для тих випадків, коли необхідно відокремлено відстежувати певні частини інфраструктури.

Топологія системи моніторингу визначає, як саме компоненти цієї системи взаємодіють між собою. Одним з поширених підходів є "централізована топологія", де всі пристрої моніторингу підключені до центрального вузла або сервера. Це спрощує координацію та управління, але може стати буттям однієї точки відмови.

Другий можливий варіант - "розподілена топологія". У цьому випадку пристрої моніторингу розташовані на різних частинах мережі та взаємодіють один з одним для обміну інформацією. Це забезпечує більшу надійність, але може вимагати більше ресурсів для координації.

Також можливий гібридний підхід, який поєднує елементи обох топологій. Наприклад, централізоване керування може бути поєднано з розподіленими датчиками або пристроями для вимірювань на місці.

Важливим аспектом систем моніторингу є їхні можливості з аналізу та використання отриманої інформації. Тут можна розглянути різні методи аналізу, від стандартного відображення стану мережі до використання аналітичних інструментів для передбачення можливих проблем.

Взагалі, вибір типу та топології системи моніторингу повинен враховувати конкретні потреби та характеристики мережі в конкретній організації. Інтеграція ефективної системи моніторингу в мережеву інфраструктуру є ключовим елементом для забезпечення її надійності, безпеки та оптимальної продуктивності.

Виділимо наступні типи систем моніторингу керуючись наявними у них функціями для слідкуванням за станом мережі:

Базові системи моніторингу: Ці системи зазвичай виконують періодичні перевірки стану елементів мережі, а їхні можливості зазвичай обмежені визначенням доступності чи недоступності конкретного елемента.

Базові системи моніторингу працюють на принципі використання ICMP-запитів та отримання відповідей від мережевих пристроїв. Основною перевагою цього підходу є його простота та легкість впровадження. Вони ідеально підходять для невеликих локальних мереж, де основний інтерес полягає в визначенні доступності або недоступності певних вузлів чи пристроїв.

Зазначеної простоти відповідає також і обмежений обсяг інформації, яку такі системи можуть надавати. Зазвичай, це обмежується вказанням того, чи є вузол доступним чи недоступним, а також можливою відповіддю на питання про час відповіді. Однак для великих мереж чи тих, що вимагають більш глибокого аналізу, цей тип систем моніторингу може бути недостатнім.

Зараз розглянемо ряд факторів, які варто враховувати при виборі та впровадженні базових систем моніторингу. Перш за все, важливо визначити обсяг мережі та її характеристики. Для невеликих мереж з обмеженим обсягом з'єднань базові системи моніторингу можуть бути ефективними та економічно вигідними рішеннями.

Однак при розгляді великих та складних мереж можуть виникнути обмеження. Системи, що базуються на протоколі ICMP, часто не забезпечують достатню глибину аналізу для виявлення проблем, таких як витрати ресурсів чи аномальні патерни трафіку. Також слід враховувати можливість виникнення "фальшивих тривог" при використанні базових систем моніторингу, оскільки вони можуть не завжди надавати повний контекст для аналізу виявлених невідповідностей.

Розширені системи моніторингу: Цей тип моніторингу відрізняється використанням різноманітних протоколів, таких як SNMP (Simple Network Management Protocol), CDP (Cisco Discovery Protocol), SSH (Secure Shell), що надає їм здатність моніторити різні аспекти пристроїв у мережі.

Основна перевага розширених систем моніторингу полягає в їхній здатності збирати розгорнуту інформацію про пристрої в мережі. Це включає стан запущених служб, використання системних ресурсів, потік даних та інші ключові показники. Використання різних протоколів дозволяє отримувати доступ до різноманітної та деталізованої інформації, що робить цей тип моніторингу особливо корисним для складних мережевих середовищ.

Одним з важливих елементів розширених систем моніторингу є SNMP, що надає можливість зчитувати дані з мережевих пристроїв та виконувати дії на них. Цей протокол є відкритим та широко використовується в інфраструктурах різного розміру. За допомогою SNMP, система моніторингу може отримувати інформацію про здоров'я пристроїв, налаштовувати їх параметри та надсилати повідомлення про події.

До інших важливих протоколів, які використовуються в розширених системах моніторингу, належить CDP. Цей протокол, розроблений компанією Cisco, дозволяє автоматичне виявлення та опис мережевих пристроїв у Cisco-мережах. Використання CDP дозволяє отримати докладну інформацію про конфігурацію та зв'язки пристроїв, що сприяє більш ефективному моніторингу та управлінню мережею. Аналогом даного протоколу являється LLDP (Link Layer Discovery Protocol), який широко розповсюджений і не являється пропрієтарним як CDP.

SSH використовується для безпечного з'єднання з мережевими пристроями та забезпечення можливості виконання команд та отримання інформації. Це особливо важливо в сучасних мережах, де безпека вважається пріоритетом.

У розширених системах моніторингу сервери зазвичай використовують локальних операторів для забезпечення ефективного взаємодії та реагування на виявлені аномалії. Локальні оператори можуть бути відповідальні за аналіз інформації, отриманої від системи моніторингу, та прийняття необхідних заходів для забезпечення стабільності та безпеки мережі.

Системи моніторингу з активним контролем: Цей тип моніторингу вирізняється здатністю системи взаємодіяти і навіть керувати параметрами мережевих пристроїв, реагуючи на виниклі події. Відмінною рисою є можливість реалізації автоматичних сценаріїв, спрямованих на ефективне управління та оптимізацію мережевого середовища.

Основна перевага систем моніторингу з активним контролем полягає у їхній здатності взаємодіяти з мережевими пристроями та впливати на їхню поведінку. Це означає, що адміністратори можуть визначати автоматичні відповіді на певні умови або події в мережі. Це робить такі системи особливо корисними для центрів обробки даних, великих корпоративних мереж, високодоступних кластерів та інших складних інфраструктур.

Наприклад, в разі виявлення перевищення певного рівня трафіку або неправильної роботи пристроїв, система може ініціювати автоматичні заходи для відновлення нормального стану. Це включає в себе можливість переключення трафіку на інші канали, перезавантаження пристроїв або виконання інших команд для вирішення проблем.

Окрім того, системи з активним контролем можуть забезпечувати великий обсяг аналізу та статистики профілювання мережі. Вони можуть визначати витрати ресурсів, ефективність пристроїв та служб, а також прогнозувати можливі проблеми. Це дозволяє адміністраторам приймати обґрунтовані рішення щодо оптимізації та підтримки інфраструктури.

1.3 Архітектура систем моніторингу мережі

Архітектура систем моніторингу мережі визначає структуру та організацію компонентів, які взаємодіють для ефективного виявлення, аналізу та реагування на події в мережевому середовищі. Цей комплексний підхід включає в себе різні елементи, такі як сенсори, сервери моніторингу, бази даних, інтерфейси для взаємодії з користувачем та інші, які спільно працюють для забезпечення

надійності та ефективності мережі. Давайте розглянемо детальніше основні аспекти архітектури систем моніторингу мережі.

Однією з ключових складових архітектури є сенсори, які відповідають за збір даних з різних частин мережі. Сенсори можуть використовувати різні методи, такі як пінгування, вимірювання трафіку, аналіз логів та інші, для отримання інформації про стан пристроїв, послуг та ресурсів мережі. Ці дані передаються до центрального сервера моніторингу для подальшого аналізу та обробки.

Центральний сервер моніторингу виконує ключову роль у системі, приймаючи дані від сенсорів та забезпечуючи централізований аналіз та управління. Він може використовувати бази даних для зберігання часових даних та вести журнали подій. Цей сервер також відповідає за визначення правил для попереджень та сповіщень, а також за ініціацію дій у випадку виявлення аномалій чи неполадок в мережі.

Ще однією важливою складовою є інтерфейс користувача, який надає можливість взаємодії з системою моніторингу. Це може бути веб-інтерфейс, десктопний додаток чи API для інтеграції з іншими системами. Інтерфейс користувача дозволяє адміністраторам моніторингу відслідковувати стан мережі, отримувати повідомлення про події та приймати рішення щодо оптимізації та вдосконалення мережевого середовища.

Бази даних використовуються для зберігання історичних даних про стан мережі. Це дозволяє аналізувати та порівнювати дані з різних періодів часу, виявляти тенденції та передбачати можливі проблеми. Також бази даних забезпечують швидкий доступ до інформації для центрального сервера та інших компонентів системи.

Для забезпечення допоміжних функцій, таких як збереження конфігураційних параметрів, автентифікація та авторизація, використовуються служби безпеки та конфігурації. Ці елементи важливі для забезпечення безпеки та стабільності системи моніторингу мережі.

Узагальнюючи, архітектура систем моніторингу мережі - це комплексна система, в якій кожен компонент виконує визначену роль для забезпечення ефективної та надійної роботи мережевого середовища. Інтеграція різноманітних елементів дозволяє адміністраторам ефективно виявляти, аналізувати та вирішувати проблеми в мережі, забезпечуючи високий рівень доступності та продуктивності.

Однією з важливих рис архітектури є механізми виявлення та вирішення неполадок в мережі. Це може включати в себе автоматизовані процеси виявлення аномалій, аналіз журналів подій, відслідковування тривог та реалізацію автоматичних сценаріїв для відновлення нормального стану. Такі механізми забезпечують оперативність системи моніторингу в умовах реального часу та реагують на зміни в мережі.

Додатковим аспектом, який впливає на архітектуру, є можливість інтеграції з іншими інфраструктурними компонентами та системами. Інтеграція з іншими моніторинговими рішеннями, системами безпеки, або іншими інструментами управління мережею може покращити комплексність та функціональність системи моніторингу.

Системи моніторингу мають розуміти і враховувати різноманітні типи пристроїв та технологій, які входять в склад мережі. Розширення можливостей моніторингу для включення різноманітних протоколів забезпечує універсальність та адаптивність системи до різних умов мережевого середовища.

Питання масштабованості є також важливим аспектом архітектури систем моніторингу. Забезпечення ефективної роботи системи в умовах зростаючого обсягу даних та розширення мережі вимагає дбайливого планування та реалізації масштабованих механізмів збору, зберігання та аналізу інформації.

Однією з інноваційних характеристик може бути використання штучного інтелекту та машинного навчання. Ці технології можуть покращити аналіз даних, забезпечуючи більш точне виявлення аномалій та вивчення патернів поведінки мережі, що дозволяє автоматизовано реагувати на ризикові сценарії.

Незважаючи на індивідуальні особливості кожної мережі даних, існує загальна ідея, яка визначає архітектуру для надійності, швидкості та ефективності передачі даних. Ця універсальна концепція широко застосовується в мережах передачі даних, а Cisco і її Enterprise Campus є прикладом компанії, яка визначає ці принципи в своїй архітектурі. Основна ідея цієї архітектури полягає в створенні трирівневої ієрархічної мережі з ядром, розподілом і рівнями доступу. Ця концепція сприяє майбутньому росту мережі, полегшує маршрутизацію, а також забезпечує адресацію і автономію окремих компонентів мережі.

В ракурсі систем моніторингу, стратегічне розташування цих систем в контрольованій мережі є ключовою задачею. Оптимальним варіантом, на думку багатьох експертів, є розташування систем моніторингу на основному шарі, яке забезпечує не тільки максимальну доступність, але й забезпечує системі моніторингу доступ до всіх рівнів елементів мережі.

При використанні систем моніторингу необхідно враховувати масштаб мережі та вибирати відповідну архітектуру. Зараз широко використовуються дві основні системи моніторингу - централізована та об'єднана. Обрана архітектура повинна враховувати потреби конкретної мережі та гарантувати ефективність та надійність моніторингу.

Процес вибору архітектури системи моніторингу також повинен враховувати розвиток технологій, таких як штучний інтелект та машинне навчання, які можуть впливати на аналіз та інтерпретацію даних в реальному часі.

Одним із ключових викликів є постійне удосконалення архітектур для врахування нових технологічних можливостей та вимог мереж. Наприклад, використання технологій віртуалізації та забезпечення безпеки даних за допомогою механізмів шифрування стають ключовими аспектами для сучасних систем моніторингу мережі.

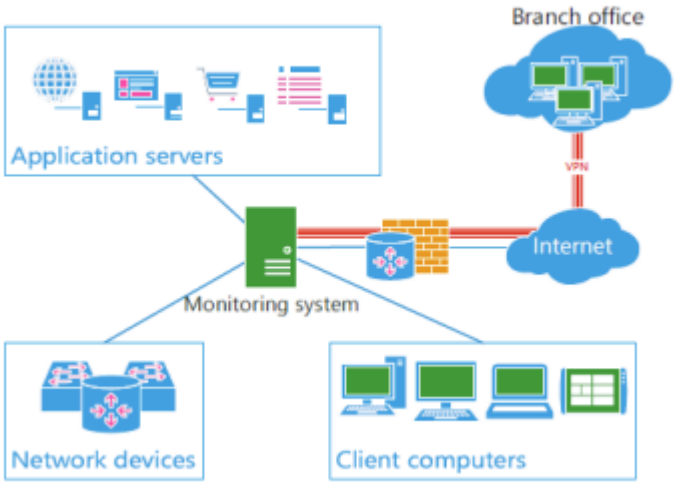


Рисунок 1.1 – Приклад архітектури з централізованою системою моніторингу

Федеративна система моніторингу мережі є інноваційним підходом до організації процесу нагляду та контролю за станом різних складових мережевого середовища. Цей підхід відрізняється від централізованої архітектури тим, що використовує розподілені засоби моніторингу, які функціонують незалежно, але можуть обмінюватися інформацією для забезпечення цілісності та повноти даних про мережу. У цьому контексті федеративний моніторинг стає важливим аспектом управління та оптимізації мережевих ресурсів.

Однією з ключових особливостей федеративної системи моніторингу є розподілена природа її компонентів. Замість централізованого сервера, в якому концентрується вся інформація, федеративна система базується на розподілених агентах або сенсорах, розташованих на різних вузлах мережі. Кожен агент відповідає за збір та аналіз даних в межах свого регіону або сегмента мережі, забезпечуючи локальний огляд стану ресурсів та обладнання.

Важливою характеристикою федеративної системи є здатність до взаємодії та обміну інформацією між різними агентами. Цей обмін може відбуватися за допомогою спеціальних протоколів чи стандартів, що дозволяють агентам об'єднувати свої дані та взаємодіяти для узгодженого реагування на події в мережі.

Важливим елементом федеративного моніторингу є архітектурна гнучкість. Оскільки кожен агент функціонує незалежно, система може легко адаптуватися до змін в мережевій топології та ресурсах. Це робить федеративний підхід ефективним для розширення та оптимізації мережі, зокрема при включенні нових вузлів чи розширенні функціональності.

У федеративній системі моніторингу велика увага приділяється локальному контролю та управлінню. Кожен агент має можливість самостійно реагувати на події в своєму регіоні, вживаючи заходів для усунення проблем та оптимізації ресурсів. Це забезпечує швидке реагування на проблеми та дозволяє максимально використовувати локальні ресурси.

Федеративна система моніторингу може також сприяти забезпеченню безпеки мережі. Оскільки агенти функціонують незалежно, система може виявляти та реагувати на загрози в реальному часі, мінімізуючи час реакції на події безпеки. Механізми шифрування та автентифікації можуть бути використані для забезпечення конфіденційності та цілісності обмінюваних між агентами даних.

Важливим аспектом федеративного моніторингу є його ефективність у виявленні та усуненні аномалій. Багато агентів, що працюють паралельно, можуть виявляти аномальні патерни або атаки в різних частинах мережі, що сприяє більш точному визначенню джерела проблеми та швидшому реагуванню на неї.

Федеративна система моніторингу також дозволяє оптимізувати використання ресурсів мережі. Кожен агент може визначати ефективне використання доступних ресурсів в своєму регіоні, що дозволяє розподіляти завдання та використовувати ресурси більш ефективно.

У федеративній системі моніторингу великий акцент робиться на інтеграції з іншими системами та сервісами. Це дозволяє спрощувати впровадження нових функціональностей та забезпечувати максимальну сумісність з існуючими інфраструктурами.

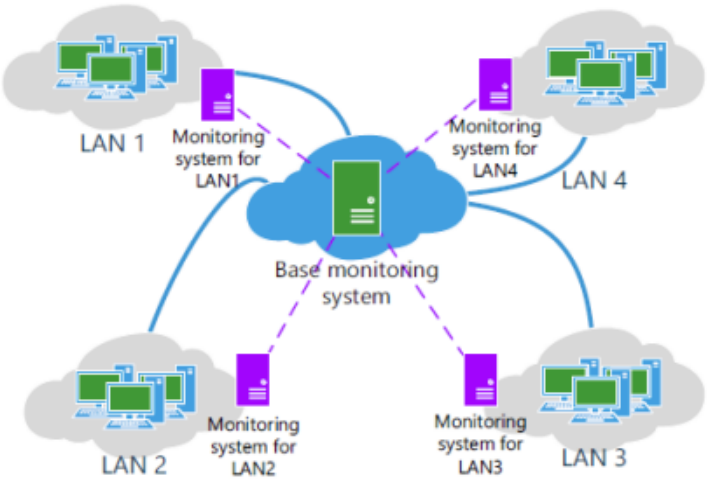


Рисунок 1.2 – Приклад архітектури з федеративною системою моніторингу

1.4 Популярні системи моніторингу

Системи моніторингу в сучасному IT-середовищі відіграють ключову роль у забезпеченні стабільності, ефективності та безпеки роботи комп'ютерних систем та мереж. За допомогою цих систем можна виявляти, відстежувати та аналізувати різні параметри, пов'язані з роботою обладнання, програмного забезпечення та мережевою активністю. Серед найбільш визнаних та використовуваних систем моніторингу виділяються такі платформи, як Zabbix, Nagios, Prometheus та інші.

Zabbix**:** Одна з найпопулярніших систем моніторингу, яка надає широкий спектр можливостей для відстеження різних аспектів мережевої архітектури. Заснована на принципах відкритості та гнучкості, Zabbix вирізняється широким спектром можливостей, які дозволяють вам ефективно контролювати різні параметри, забезпечуючи надійний моніторинг фізичних та віртуальних серверів.

Однією з ключових переваг Zabbix є його здатність моніторити різноманітні аспекти IT-інфраструктури, включаючи використання CPU, обсяги пам'яті, стан мережі, а також роботу різних служб та додатків. Це досягається завдяки можливості встановлення агентів або використанню агентлес підходу, що робить

Zabbix універсальним інструментом для моніторингу різних конфігурацій серверів.

Однією з основних переваг Zabbix є його відкритість. Вільний доступ до вихідних кодів дозволяє користувачам самостійно адаптувати та змінювати систему під свої потреби. Гнучкість Zabbix проявляється в можливості розширення функціоналу завдяки великій кількості модулів та розширень. Це робить Zabbix ідеальним вибором для тих, хто шукає налаштовану на замовлення систему моніторингу.

Ще однією важливою характеристикою Zabbix є його автоматизовані можливості. Система дозволяє налаштовувати автоматичний збір даних, визначати порогові значення для спрацювання алертів та виконувати певні дії у випадку виявлення проблем. Це спрощує процес моніторингу та дозволяє оперативно реагувати на потенційні проблеми.

Засоби алертингу в Zabbix включають в себе різноманітні методи повідомлення, такі як електронна пошта, SMS, та інші, що дозволяє операторам швидко і ефективно реагувати на події. Крім того, система забезпечує можливість створення розширених звітів та графіків для подальшого аналізу динаміки та ефективності інфраструктури.

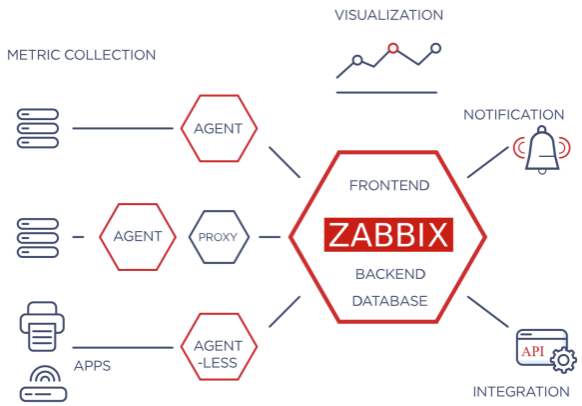


Рисунок 1.3 – Приклад архітектури Zabbix

Однак, слід зазначити, що для не досвідчених користувачів може виникнути певна складність налаштування системи, особливо у великих масштабах.

Необхідність глибокого розуміння IT-інфраструктури може викликати труднощі в початковому впровадженні. Тим не менш, з належним розумінням та досвідом, Zabbix стає потужним інструментом для забезпечення стабільності та ефективності IT-систем.

Nagios: Є однією з найстаріших та найбільш відомих систем моніторингу, яка визначається своєю довгою історією, активною спільнотою користувачів та неперервним розвитком. Заснована на принципах відкритості та гнучкості, ця система дозволяє користувачам ефективно контролювати та управляти станом різних компонентів їхніх інфраструктур.

Однією з ключових переваг Nagios є його філософія, спрямована на забезпечення гнучких можливостей моніторингу. Це досягається завдяки використанню різних плагінів, які дозволяють взаємодіяти з різними типами обладнання та програмного забезпечення. Широкий спектр плагінів робить Nagios універсальним для різних середовищ, де можуть використовуватися різноманітні протоколи та стандарти.

Система має розширюваність завдяки своїй модульній структурі. Це дозволяє користувачам додавати нові функціональності та плагіни для відстеження нових типів ресурсів чи взаємодії з іншими системами. Така гнучкість робить Nagios підходящим в різних сценаріях використання.

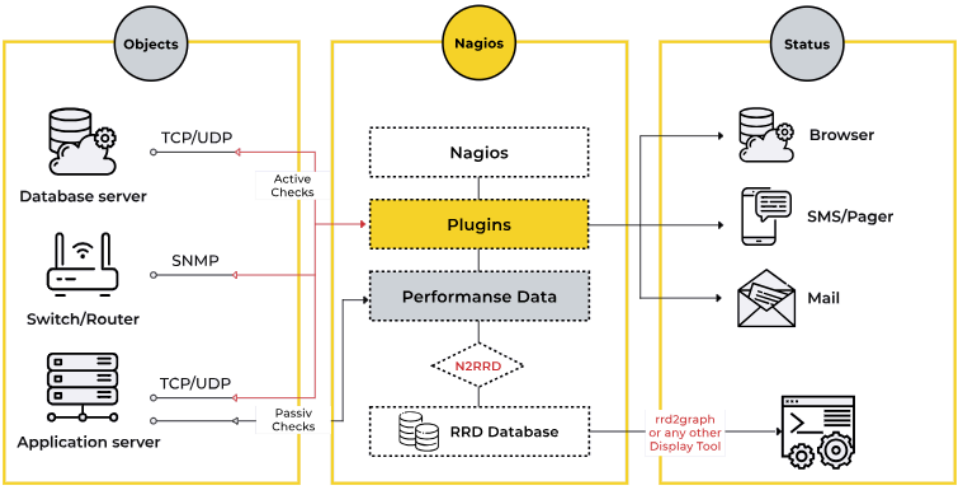


Рисунок 1.4 – Приклад архітектури Nagios

Однією з важливих можливостей Nagios є налаштовуваність різних рівнів алертингу. Користувачі можуть визначати, як система повинна реагувати на різні події та стани, відправляти повідомлення або автоматично виконувати певні дії. Це дозволяє ефективно управляти реагуванням на проблеми та мінімізувати вплив відмов на роботу.

Незважаючи на свої переваги, Nagios може виявитися менш гнучким у порівнянні з іншими сучасними системами моніторингу. Деякі конкуренти можуть надавати більше інтеграцій, аналітичних можливостей та зручних інтерфейсів. Також, в деяких випадках, Nagios може бракувати деяких сучасних функцій, що притаманні новішим рішенням у цій галузі.

ELK Stack: представляє собою потужний інструментарій для збору, зберігання, обробки та візуалізації логів і данних. Цей стек використовується для моніторингу та аналізу різноманітних систем, додатків та сервісів, надаючи користувачам зручний та потужний інтерфейс для роботи з великими обсягами даних. Розглянемо більше деталей про кожен компонент ELK Stack.

Elasticsearch виступає в ролі основного зберігача даних та пошукового двигуна. Він забезпечує потужний механізм індексації, що дозволяє швидко знаходити та аналізувати дані. Elasticsearch використовує сучасний алгоритм пошуку, який дозволяє виконувати складні запити за короткий час.

Logstash відповідає за збір, обробку та передачу лог-даних до Elasticsearch. Він може обробляти дані з різних джерел, таких як журнали подій, файлові системи, бази даних тощо. Logstash надає гнучкість у налаштуванні обробки даних, дозволяючи фільтрувати, трансформувати та розподіляти дані перед їх індексацією в Elasticsearch.

Kibana — це візуалізаційний інтерфейс, який надає користувачам зручний спосіб відображення та аналізу даних, збережених в Elasticsearch. За допомогою Kibana можна створювати різноманітні графіки, діаграми, таблиці та інші візуальні елементи для зручного моніторингу та аналізу лог-даних.

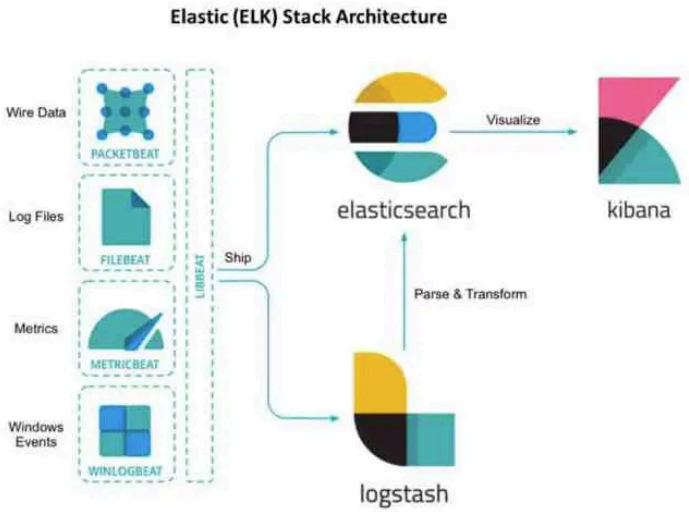


Рисунок 1.5 – Приклад архітектури ELK

Основна перевага ELK Stack полягає в його здатності збирати, агрегувати та візуалізувати дані з різних джерел, що дозволяє здійснювати комплексний моніторинг систем. Велика гнучкість конфігурації дозволяє адаптувати стек до різних вимог та середовищ.

Однак, важливо враховувати, що розгортання та конфігурація ELK Stack може бути складною задачею, особливо для користувачів з обмеженим досвідом в області системного адміністрування та аналізу даних. Також, великий обсяг даних може вимагати значних ресурсів для ефективної роботи стеку.

1.5 Система моніторингу Prometheus

Сучасні інформаційні технології передбачають великий обсяг роботи з комп'ютерними системами та інфраструктурою. Важливим елементом у забезпеченні надійності та ефективності цих систем є їхній моніторинг. Prometheus - це одна з передових систем моніторингу, яка виникла у відповідь на ростучі потреби в моніторингу складних інфраструктур великих організацій.

Prometheus був розроблений у 2012 році інженерами компанії SoundCloud, що є однією з провідних платформ для обміну та відтворення музики. На той момент, інженери виявили потребу у високоефективному інструменті моніторингу

для їхньої розподіленої інфраструктури, яка включала тисячі серверів та компонентів. Під керівництвом Матіаса Клауза та Юліуса Вейса, команда розробників розпочала роботу над Prometheus, спрямовуючи свої зусилля на створення інструменту, який би відповідав вимогам їхньої розмаїтої інфраструктури.

Prometheus ґрунтується на кількох ключових принципах, які роблять його потужним і ефективним інструментом моніторингу. Це модель збору даних, яка базується на принципі "pull" (запит на отримання даних), а не "push" (надсилання даних сервером). Кожен моніторинговий об'єкт, як правило, є експортером, який використовує HTTP для надання своїх метрик Prometheus. Кожен об'єкт, який ми хочемо моніторити, може бути розглянутий як моніторинговий об'єкт, або експортер. Експортер - це компонент, який надає метрики Prometheus. Він може бути вбудованим у додаток чи окремим процесом, і його основна функція - використовувати HTTP для надання своїх метрик серверу Prometheus. Це робить систему вкрай гнучкою, оскільки будь-який компонент може бути інтегрованим у моніторинг. Архітектура Prometheus спрямована на оптимізацію ресурсів та ефективний збір даних. Сервер Prometheus ретельно управляє ресурсами та використовує їх обмежено, що особливо важливо в умовах великої інфраструктури. Компактність та швидкодія сервера сприяють швидкому збору, зберіганню та обробці великої кількості метрик. Для аналізу та використання зібраних метрик, Prometheus використовує мову запитів PromQL. Ця мова дозволяє виконувати різноманітні аналізи даних, включаючи агрегації, фільтрації та групування. PromQL забезпечує користувачам гнучкість у взаємодії з даними та дозволяє створювати складні запити для вивчення різних аспектів функціонування системи. Відкритий код дозволяє розробникам спільно працювати над вдосконаленням інструменту, додавати нові функції та усувати можливі недоліки.

Архітектура Prometheus вражає своєю гнучкістю та можливостями, що дозволяють забезпечувати надійний моніторинг різноманітних систем. Ця система є важливою для технічних фахівців та інженерів, які відповідають за забезпечення

стабільності та ефективності інфраструктури. Давайте докладніше розглянемо ключові компоненти Prometheus та їх роль у створенні потужної системи моніторингу. Один із фундаментальних елементів архітектури Prometheus - це сервер Prometheus, який виконує центральну роль у зборі та обробці метрик. Сервер забезпечує механізми для зчитування конфігураційних файлів та налаштування правил збору метрик. Важливо відзначити, що Prometheus використовує модель pull, коли сам сервер ініціює звернення до систем та отримує від них метрики. Це дозволяє забезпечити актуальні дані та ефективно керувати навантаженням на систему. Крім того, сервер Prometheus може динамічно налаштовуватися з використанням мови запитів PromQL (Prometheus Query Language). PromQL дозволяє виконувати різноманітні запити та агрегації над метриками, що надає адміністраторам та інженерам глибокий інсайт у функціонування їх систем. Клієнтські бібліотеки Prometheus грають ключову роль у вбудованні засобів збору метрик безпосередньо в додатки та служби. Ці бібліотеки, доступні для різноманітних мов програмування, дозволяють розробникам легко інтегрувати моніторинг без значного зусилля. Автоматична реєстрація метрик та їх оновлення роблять процес моніторингу більш простим та ефективним.

Alertmanager є ключовим компонентом системи моніторингу Prometheus, який відповідає за обробку та маршрутизацію сповіщень, що генеруються Prometheus сервером під час виявлення аномалій або проблем в інфраструктурі. Цей інструмент розроблений для того, щоб надавати гнучкі та ефективні засоби управління сповіщеннями, щоб оператори та адміністратори систем могли ефективно реагувати на події та забезпечувати стабільну роботу своєї інфраструктури. Однією з ключових функцій Alertmanager є здатність групування сповіщень. Він може об'єднувати повідомлення, які вказують на одну і ту ж саму проблему, щоб зменшити кількість надходжень та уникнути засмічення. Це зроблено для того, щоб оператори мали змогу ефективно працювати з великою кількістю подій та взаємодіяти з ними шляхом прийняття відповідних рішень.

Alertmanager підтримує також низку різних інтеграцій для подальшого розширення його функціоналу. Зокрема, він може взаємодіяти з різними каналами сповіщень, такими як електронна пошта, Slack, PagerDuty та інші. Це дозволяє адміністраторам інтегрувати Alertmanager з іншими інструментами та процесами у своїй робочій екосистемі. Однією з важливих характеристик є гнучкість налаштувань правил сповіщень. Alertmanager використовує вирази для визначення умов, при яких сповіщення мають бути відправлені. Це дозволяє точно визначити, які події вважатимуться критичними та вимагатимуть негайної реакції, а які можуть бути ігноровані або оброблені в зручний час. Однією з ключових переваг є підтримка повторюваних сповіщень. Це означає, що Alertmanager може нагадувати про проблему, якщо вона не була вирішена протягом певного періоду часу. Це корисно для випадків, коли оператор може не встигнути реагувати на сповіщення або коли проблема вимагає повторної уваги. Alertmanager також має можливість сортування та фільтрації сповіщень за різними критеріями, що робить його інструментом зручним та дружнім до користувача. Оператори можуть легко визначити пріоритети, налаштовувати фільтри та пристосовувати поведінку системи до своїх конкретних потреб. У разі великих інфраструктур або розподілених систем Alertmanager може бути розгорнутий у конфігурації з багатьма екземплярами, що дозволяє оптимізувати обробку великої кількості сповіщень та розподілювати навантаження для підтримки високої доступності та стійкості системи.

Інтеграція між Grafana та Prometheus є популярним рішенням у сфері моніторингу, оскільки ці два інструменти доповнюють один одного. Prometheus надає надійний та ефективний механізм для збору метрик, тоді як Grafana відповідає за їх візуалізацію та аналіз. Користувачі можуть створювати гнучкі дашборди, які відображають ключові аспекти стану системи та дозволяють ефективно взаємодіяти з даними. Крім того, Grafana пропонує різноманітні опції для налаштування вигляду та поведінки графіків, що робить його ідеальним інструментом для адаптації до різних потреб та стилів користувачів. Від кругових

діаграм до теплових карт, Grafana надає широкі можливості для створення індивідуальних та ефективних візуалізацій даних. Окрім того, Grafana забезпечує можливість використання різних джерел даних, не тільки Prometheus. Це означає, що користувачі можуть комбінувати дані з різних джерел та систем моніторингу, щоб отримати повний образ стану їх інфраструктури. Ця гнучкість дозволяє підтримувати різноманітні сценарії використання та задовольняти унікальні вимоги користувачів. Налаштування інтеграції між Grafana та Prometheus також відносно просте. Grafana надає готові конфігураційні файли для підключення до Prometheus, і користувачам лише потрібно вказати адреси та порти свого екземпляра Prometheus. Це робить процес встановлення та налаштування об'єднаної системи моніторингу досить безпечним та доступним.

Процес безпеки в архітектурі Prometheus теж заслуговує уваги. Важливо застосовувати механізми автентифікації та авторизації для забезпечення безпеки збережених метрик та конфігураційних даних. Забезпечення конфіденційності та цілісності даних є важливим аспектом для будь-якої системи моніторингу, особливо у випадку великих та критичних інфраструктур. Архітектура Prometheus також відзначається можливістю розширення та інтеграції. Відкритий характер системи дозволяє розробникам створювати власні розширення та модулі, що відповідають їхнім конкретним потребам. Це робить Prometheus ідеальним інструментом для різноманітних середовищ, від невеликих стартапів до великих корпоративних інфраструктур.

Завдяки своїй відкритій архітектурі та активній спільноті користувачів та розробників, Prometheus продовжує розвиватися та вдосконалюватися. Спільнота вносить нові можливості, покращення та інновації, що дозволяє системі залишатися сучасною та конкурентоспроможною в світі технологій моніторингу.

Враховуючи всі ці аспекти, можна зробити висновок, що архітектура Prometheus - це не просто набір компонентів для збору метрик, а повноцінна система моніторингу, яка працює на кшталт нейронної системи, де кожен

компонент є клітинкою, що виконує свою чітко визначену роль для забезпечення високоякісного та ефективного моніторингу інфраструктури.

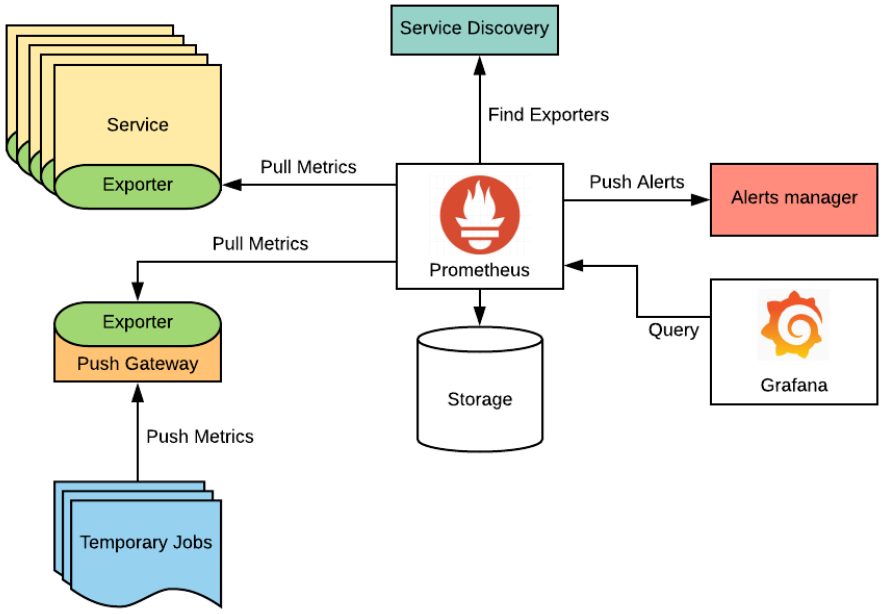


Рисунок 1.6 – Приклад архітектури Prometheus

Архітектура Prometheus підтримує масштабованість, що робить її ефективною для великих інфраструктур. Сервери Prometheus можуть бути організовані в кластери, що дозволяє розділити навантаження та забезпечити високу доступність системи. Кластеризація дозволяє забезпечити неперервний моніторинг навіть у випадку виходу з ладу окремих компонентів. Здатність кластеризації серверів Prometheus сприяє горизонтальному масштабуванню, що робить систему більш адаптованою до зростання обсягу оброблюваних метрик. Це зокрема важливо для компаній, які постійно розширюють свою інфраструктуру та потребують надійного і ефективного моніторингу на всій шкалі. Prometheus підтримує різноманітні типи метрик, включаючи лічильники, гістограми та сумірні показники. Це розширює можливості моніторингу, дозволяючи збирати та аналізувати різні аспекти функціонування системи. Наприклад, гістограми можуть надати інформацію про розподіл часу відповіді системи на різні категорії, що допомагає виявити та оптимізувати ділянки збоїв. Зберігання даних у Prometheus

реалізовано з урахуванням потреб користувача. Дані зберігаються у внутрішніх базах даних, і користувач може налаштувати період ретенції для керування обсягом збережених даних. Це робить систему гнучкою та адаптованою до конкретних потреб проекту. Системи моніторингу повинні легко впроваджуватися та взаємодіяти з різноманітними сервісами. Prometheus підтримує різні механізми автоматичного виявлення сервісів (service discovery), такі як Kubernetes service discovery, EC2 service discovery та інші. Це дозволяє системі динамічно адаптуватися до змін у складі інфраструктури та автоматично визначати нові цілі для моніторингу. Надійність є ключовим аспектом архітектури Prometheus. Система вміє ефективно впоратися із збоями та відновлюватися після відключень. Застосування реплікації даних у кластері дозволяє зберегти дані в надійній формі, навіть якщо один із серверів вийде з ладу. У разі втрати з'єднання з метричними об'єктами, Prometheus може кешувати дані та надсилати їх пізніше, коли з'єднання відновиться. Це робить систему стійкою до тимчасових втрат зв'язку, що особливо важливо в умовах розподіленого середовища.

Успіх Prometheus полягає у його розширюваності та широкій екосистемі інструментів. Доступність різноманітних експортерів, готових інтеграцій та розширень дозволяє користувачам адаптувати систему до конкретних потреб їхніх проектів. За допомогою таких розширень, Prometheus може легко інтегруватися з іншими інструментами та сервісами для вирішення конкретних завдань моніторингу. Експортери дозволяють інтегрувати різноманітні системи та сервіси з Prometheus, роблячи їхні метрики доступними для системи моніторингу. Ця архітектура дозволяє легко взаємодіяти з різними компонентами і максимально використовувати можливості моніторингу. Один із ключових аспектів експортерів - це можливість адаптації до різних технологій та сервісів. Наприклад, для веб-серверів існують HTTP експортери, які надають зручний інтерфейс для збору та виведення метрик, таких як кількість запитів, час відповіді, або статусні коди. Популярні експортери, такі як Node Exporter, дозволяють моніторити метрики різних операційних систем, включаючи вільні ресурси, використання пам'яті та

інші характеристики системи. Важливою перевагою експортерів є їхній високий рівень налаштування та можливість визначення власних правил обробки метрик. Це дає можливість адміністраторам точно визначити, які метрики є критичними для їхнього конкретного середовища та як їх слід використовувати для моніторингу та аналізу. Така гнучкість робить експортери ефективними інструментами для різноманітних сценаріїв моніторингу. Не менш важливим є питання безпеки при використанні експортерів. Багато експортерів надають можливості автентифікації та авторизації для захисту від несанкціонованого доступу. Це надає додатковий рівень захисту для забезпечення конфіденційності та цілісності зібраних метрик. Деякі експортери також підтримують шифрування для безпечного передавання метрик між компонентами системи моніторингу. Розглядаючи аспекти масштабування, експортери можуть бути розглянуті як механізм для горизонтального масштабування системи моніторингу. Завдяки їхній здатності працювати паралельно та обробляти великий обсяг метрик, експортери дозволяють легко розширювати інфраструктуру моніторингу відповідно до зростаючих потреб компанії чи проекту. Ефективність експортерів також виявляється в їхній здатності інтегруватися з іншими інструментами та сервісами. Наприклад, інтеграція з алертінговими системами дозволяє автоматизувати реакцію на виявлені проблеми та події. Така взаємодія покращує загальну ефективність системи моніторингу та забезпечує швидкий відгук на потенційні проблеми. Спільнота користувачів та розробників є невід'ємною частиною успіху Prometheus. Активна спільнота допомагає вирішувати проблеми, розробляти нові функції та підтримувати постійний розвиток системи. Обмін досвідом, відгуки та співпраця з іншими користувачами додають цінності використанню Prometheus в різних сценаріях. Архітектура Prometheus знаходить застосування в різних областях індустрії. Від моніторингу хмарних сервісів до керування великими кластерами контейнерів, від великих корпоративних мереж до невеликих стартапів - Prometheus забезпечує високоякісний моніторинг для будь-якої

інфраструктури. Усі ці аспекти архітектури Prometheus роблять її потужним та універсальним інструментом для моніторингу.

І тому для реалізації даної кваліфікаційної роботи мною був обраний сервер моніторингу Prometheus. Ключову роль в цьому виборі грає те що Prometheus являється opensource проектом та його модульність за рахунок інших мов програмування, завдяки чому є можливість будь якого маштабування.

2 АНАЛІЗ ПРОТОКОЛІВ VPN ТА СИСТЕМ З UNIX-ПОДІБНИМИ ЯДРАМИ

2.1 Віртуальні приватні мережі

Віртуальна приватна мережа (Virtual Private Network) - це технологія, яка забезпечує безпеку та конфіденційність при передачі даних через мережу, таку як Інтернет. Вона дозволяє створити зашифрований тунель між вашим пристроєм та сервером VPN, що приховує вашу реальну IP-адресу та захищає вашу приватність від сторонніх осіб. Однією з ключових функцій VPN є захист від перехоплення даних. Коли ви підключаєте свій пристрій до VPN, всі дані, які ви передаєте через Інтернет, шифруються. Це означає, що, навіть якщо хтось може перехопити ваші дані, вони залишаються незрозумілими і непридатними для використання без ключа шифрування. Коли ви підключаєтеся до сервера VPN, ваш IP-адреса маскується, а ваш трафік прокладається через сервер, роблячи важчим відстеження вашого місцезнаходження та ідентифікацію. Використання VPN також може допомогти у обході обмежень в Інтернеті. Деякі веб-сайти та сервіси можуть бути недоступні в певних країнах чи регіонах через обмеження або цензуру. За допомогою VPN ви можете обходити ці обмеження, підключаючись до сервера в іншій країні, де такі обмеження не існують. Ще однією важливою функцією VPN є забезпечення безпеки від зловмисних атак. У великих мережах, таких як громадські Wi-Fi мережі, існує ризик перехоплення чутливої інформації, такої як паролі або фінансові дані. Використання VPN дозволяє зашифровувати дані, що пересилаються через такі ненадійні мережі, зменшуючи ризик зловмисних атак. Крім того, VPN може бути використаний для забезпечення конфіденційності ідентифікації.

VPN-тунель являє собою наскрізне з'єднання, встановлене між двома пристроями - як правило, між VPN-сервером і вашим пристроєм. Тунелювання дає змогу помістити трафік у стандартні пакети TCP/IP і безпечно передавати його через Інтернет.

Рисунок 2.1 – Простий приклад VPN-з’єднання між сервером та клієнтом

Ваш інтернет-постачальник (ISP) може відстежувати та записувати вашу діяльність в Інтернеті. VPN дозволяє приховати цю діяльність, запобігаючи збору і використанню даних про ваші онлайн-звички. Важливою характеристикою VPN є вибір сервера. Ви можете вибрати сервер у будь-якій країні, де ваша служба VPN має сервери. Це може бути корисним у випадку, якщо вам потрібен доступ до ресурсів, які доступні тільки для жителів певної країни або якщо ви хочете обходити географічні обмеження. Нарешті, слід зазначити, що хоча VPN допомагає захищати вашу конфіденційність та безпеку в Інтернеті, вона не є універсальним засобом.

У добу швидкого розвитку технологій та зростання важливості цифрового світу, використання VPN стає невід'ємною частиною безпеки та конфіденційності в Інтернеті. Ця технологія забезпечує не лише анонімність і захист персональної інформації користувачів, але й відкриває нові можливості для доступу до обмежених контентів та забезпечення безпеки в публічних мережах.

Загалом, VPN виконує низку важливих функцій у сфері забезпечення безпеки та конфіденційності в Інтернеті. Використання цієї технології стає необхідністю для тих, хто прагне захистити свої дані від небажаних переглядів, отримати доступ до обмежених ресурсів та забезпечити безпеку в громадських мережах. Правильне використання VPN дозволяє користувачам насолоджуватися

безпечним та анонімним Інтернет-досвідом, що є особливо важливим у світі, де цифрові загрози стають все більш серйозними.

Але є і свої мінуси. Низька швидкість Інтернет-з'єднання може бути відчутною при використанні віртуальної приватної мережі (VPN). Зашифрування даних, яке використовується для забезпечення конфіденційності, може вимагати додаткового часу на обробку, що впливає на загальну швидкість передачі даних. Більше того, віддалені сервери VPN можуть призводити до збільшення відстані, яку потрібно подолати трафіку, що також впливає на продуктивність.

Важливим нюансом є періодичні втрати з'єднання VPN та неочікуване виходження трафіку в публічну мережу. Розриви можуть стати вразливістю для конфіденційності, оскільки користувач може навіть не помітити такі події, що призводить до можливої утечі даних. Також важливо, щоб VPN-з'єднання відновлювалося автоматично, але ця можливість не завжди доступна без спеціальних налаштувань.

Проблема також полягає в тому, що IPv6 рідко підтримується VPN. Коли використовується IPv6 в публічній мережі та ресурс підтримує його, трафік може автоматично рухатися через відкриту IPv6-мережу. Відключення IPv6 в операційній системі може захистити від таких ситуацій.

Питання DNS-протікань є ще однією трудністю. DNS-запити часто обробляються публічними DNS-серверами, а не віртуальними, що може викликати проблеми конфіденційності. Некоректні відповіді можуть призвести до отримання фальшивої інформації, а також розкриття геолокації та постачальника Інтернету користувача.

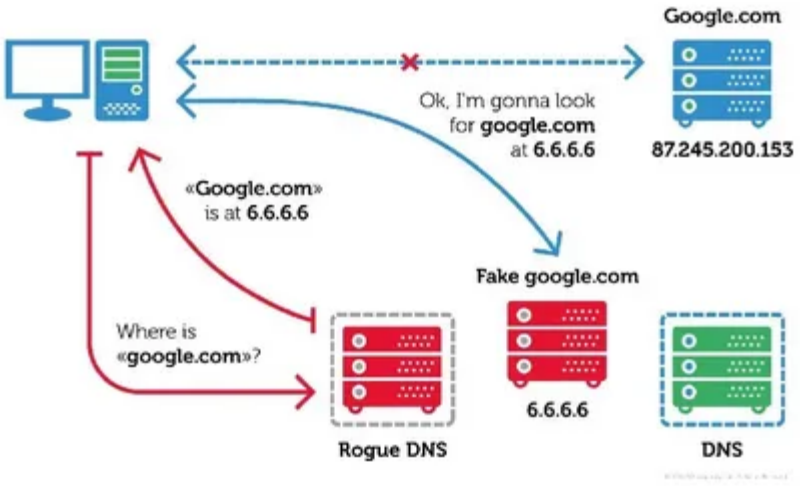


Рисунок 2.2 – Простий приклад підміни DNS відповіді

Юридичні аспекти також варто враховувати. Різниці в законодавстві різних країн, де можуть розташовуватися VPN-клієнт і VPN-сервер, можуть створити правові нев'язки. Також, якщо трафік проходить через третю країну, існує ризик збереження копії даних для подальшого вивчення. Важливо враховувати ці аспекти при використанні VPN для забезпечення не лише технічної, але й юридичної безпеки.

2.2 Популярні протоколи VPN

Протоколи VPN є ключовою складовою для забезпечення безпеки та конфіденційності під час передачі даних через Інтернет. Існує кілька популярних протоколів VPN, кожен з яких має свої характеристики та застосування в залежності від конкретних потреб користувача або організації. Давайте розглянемо деякі з найбільш популярних протоколів VPN:

L2TP/IPsec (Layer 2 Tunneling Protocol with IPsec) є одним із широко використовуваних протоколів VPN, призначеним для створення безпечного тунелю між користувачем та сервером VPN. Цей протокол, поєднуючи в собі L2TP та IPsec, надає ефективний механізм для забезпечення високого рівня безпеки та конфіденційності в мережі.Однією з ключових особливостей L2TP/IPsec є його

здатність до створення тунелю, що дозволяє безпечно транспортувати дані від користувача до VPN-сервера. Процес створення тунелю починається з L2TP, який відповідає за встановлення логічного каналу з'єднання між користувачем та сервером. Такий логічний канал стає основою для подальшої передачі даних. Забезпечення безпеки і конфіденційності є критичним аспектом будь-якого VPN-з'єднання. У випадку L2TP/IPsec, IPsec використовується для шифрування та захисту передаваних даних. IPsec забезпечує безпеку на рівні мережі, гарантуючи, що інформація, що проходить через тунель, залишається конфіденційною та недоступною для неповажних осіб. Однак, не дивлячись на вагомі переваги L2TP/IPsec, існують певні обмеження та проблеми, які важливо враховувати. Зокрема, деякі проблеми з безпекою можуть виникнути в разі використання L2TP без IPsec. Також, іноді можуть виникати труднощі в конфігурації та використанні, що може ускладнити впровадження даного протоколу в деяких сценаріях. Ще однією важливою характеристикою L2TP/IPsec є його сумісність з різними платформами та операційними системами. Цей протокол може бути використаний на різних пристроях, від комп'ютерів до мобільних пристроїв, що робить його відмінним вибором для різноманітних користувачів. У той час як L2TP/IPsec забезпечує високий рівень безпеки, важливо також звертати увагу на можливі проблеми та вразливості, які можуть виникнути в конкретних сценаріях використання. При впровадженні цього протоколу важливо дотримуватися кращих практик забезпечення, включаючи регулярне оновлення програмного забезпечення та правильну конфігурацію параметрів безпеки. Загалом L2TP/IPsec залишається одним із важливих протоколів VPN, здатних забезпечити не тільки ефективну та безпечну передачу даних, але й сумісність з різними платформами. Правильне використання цього протоколу може забезпечити користувачам надійний та конфіденційний доступ до мережі віддалено.

SSTP (Secure Socket Tunneling Protocol) є захоплюючим та важливим інструментом в сфері VPN, пропонуючи унікальний підхід до створення безпечного тунелю між користувачем та сервером VPN. Цей протокол

використовує SSL/TLS для забезпечення безпеки та конфіденційності даних, що передаються через мережу. Однією з ключових особливостей SSTP є його використання SSL/TLS для шифрування трафіку та створення тунелю. SSL/TLS - це криптографічні протоколи, які гарантують безпеку передачі даних шляхом шифрування та аутентифікації. Із застосуванням цих протоколів, SSTP створює безпечний тунель, що дозволяє користувачам безпечно обмінюватися інформацією з віддаленим сервером VPN. Ще однією перевагою SSTP є використання порту 443, який є стандартним для HTTPS трафіку. Це робить його ефективним і універсальним, оскільки багато систем та мереж фільтрують чи блокують трафік, який несе на собі порт 443. Такий підхід дозволяє обходити обмеження та фільтрацію трафіку, що робить SSTP привабливим вибором для користувачів, які знаходяться в умовах з жорсткими обмеженнями мережі. Особливо слід відзначити, що SSTP є популярним протоколом серед користувачів Windows. Вбудована підтримка SSTP у цій операційній системі дозволяє зручно та легко встановлювати VPN-з'єднання без додаткових програм чи складних налаштувань. Це забезпечує велику розповсюдженість та використання SSTP серед користувачів, що вибирають Windows для своїх потреб. Проте, важливо враховувати, що не зважаючи на свої переваги, SSTP має свої обмеження та особливості. Наприклад, деякі аспекти його реалізації можуть бути обмеженими на платформах, відмінних від Windows. Крім того, питання безпеки завжди залишається ключовим, і важливо регулярно оновлювати та моніторити систему для запобігання потенційним загрозам. У світі швидкозмінюваних технологій SSTP залишається актуальним та ефективним протоколом для забезпечення безпеки та конфіденційності в мережах VPN. З його допомогою користувачі можуть не лише обходити обмеження та фільтрацію трафіку, але й забезпечити безпечність своїх даних під час взаємодії з віддаленими ресурсами.

WireGuard представляє собою інноваційний протокол VPN, який вирізняється своєю високою ефективністю та простотою в налаштуванні. Завдяки своєму сучасному підходу до тунелювання та безпеки, WireGuard швидко

завоював популярність серед користувачів та адміністраторів мереж. Однією з ключових особливостей WireGuard є його висока ефективність. Протокол пропонує швидший та більш ефективний тунелінг, порівняно з багатьма традиційними VPN-протоколами. Це досягається завдяки використанню сучасних криптографічних алгоритмів та ефективному управлінню сесійними ключами. Швидкість та продуктивність WireGuard зробили його привабливим вибором для тих, хто цінує ефективність своєї мережі. Ще однією перевагою WireGuard є його простота в налаштуванні. Відкритий характер протоколу дозволяє легко втілювати його на різних платформах та в різних середовищах. Використання сучасних технологій та простий дизайн дозволяють швидко налаштовувати та впроваджувати WireGuard, що робить його привабливим для широкого кола користувачів, від досвідчених адміністраторів до звичайних користувачів. Важливою рисою WireGuard є його відкритість та легкість у реалізації. Будучи відкритим проектом, WireGuard надає можливість для співпраці та внесення внесків від різних розробників та спеціалістів. Це стимулює розвиток та вдосконалення протоколу, а також робить його прозорим та достовірним для спільноти. Дизайн WireGuard спрощує процес автентифікації та обміну ключами. У порівнянні з багатьма традиційними протоколами, WireGuard має менше етапів установки тунелю та менше витрат на обробку даних. Це не лише покращує швидкість тунелювання, але і зменшує навантаження на мережеве обладнання, що робить його ефективним для використання в різних сценаріях. Однак, важливо враховувати, що, незважаючи на всі переваги, які пропонує WireGuard, він може не підходити для всіх випадків використання. Враховуючи різноманітність сценаріїв та вимог користувачів, слід уважно вибирати протокол VPN відповідно до конкретних потреб та умов. Загалом WireGuard визначається як передовий та майбутнєорієнтований протокол VPN, який об'єднує в собі високу ефективність, простоту в налаштуванні та відкритість для спільної розробки та вдосконалення. Його унікальні характеристики роблять його цікавим вибором для тих, хто шукає передові рішення в області віртуальних приватних мереж.

Internet Key Exchange version 2 (IKEv2) є протоколом безпеки мережі, який використовується для створення та управління безпечними з'єднаннями VPN. Цей протокол забезпечує ефективний та безпечний обмін ключами, автентифікацію та управління безпекою в мережі. За допомогою IKEv2 користувачі можуть забезпечити конфіденційність, цілісність та доступність своєї інформації при передачі через ненадійні мережі, такі як Інтернет. Однією з ключових характеристик IKEv2 є його здатність створювати та управляти безпечними з'єднаннями, відомими як Security Associations (SA). SA визначає параметри безпеки для з'єднань, такі як алгоритми шифрування та аутентифікації. Протокол надає можливість автоматично оновлювати SA, що забезпечує стабільні та безпечні VPN-з'єднання впродовж тривалого часу. Іншою важливою функцією IKEv2 є його здатність працювати в різних режимах, таких як режим тунелювання (Tunnel Mode) та режим транспорту (Transport Mode). Режим тунелювання використовується для захищення та шифрування всього IP-трафіку між двома кінцевими точками, тоді як режим транспорту застосовується для безпечності між двома кінцевими точками, залишаючи зовнішній IP-трафік незахищеним. Аутентифікація грає важливу роль у забезпеченні безпеки мережі, і IKEv2 пропонує кілька методів аутентифікації. До них входять методи на основі приватного ключа, пароля, сертифіката, а також методи EAP (Extensible Authentication Protocol), що дозволяють реалізувати різноманітні механізми аутентифікації. Протокол також має вбудовану підтримку для мобільних платформ, що робить його ідеальним вибором для користувачів, які використовують VPN на своїх смартфонах або планшетах. Із застосуванням технології Mobility and Multihoming (MOBIKE), IKEv2 може забезпечити стійке VPN-з'єднання при зміні точок доступу або переході між різними мережами.

истеми безпеки та криптографії, використовані в IKEv2, забезпечують високий рівень захисту даних. Наприклад, для шифрування може використовуватися Advanced Encryption Standard (AES), що вважається одним із найбільш надійних алгоритмів шифрування. Однак, як і у будь-якого протоколу, є свої виклики та

обмеження. Деякі з них включають проблеми, пов'язані з NAT (Network Address Translation) і необхідність використання додаткових пристосувань для подолання цих проблем. Ще однією важливою аспектом є підтримка IKEv2 на різних платформах. Багато операційних систем, включаючи Windows, macOS, Android, та iOS, вже вбудовано підтримку IKEv2, що полегшує встановлення та конфігурацію VPN-з'єднань для різних користувачів. Загалом, Internet Key Exchange version 2 є потужним та різноманітним протоколом VPN, який забезпечує високий рівень безпеки, ефективність та мобільність. З його допомогою користувачі можуть створювати безпечні з'єднання через небезпечні мережі, зберігаючи конфіденційність своєї інформації та забезпечуючи надійний обмін даними.

Generic Routing Encapsulation (GRE) є протоколом тунелювання, який використовується для передачі різних типів мережевого трафіку між двома вузлами VPN чи двома віддаленими мережами через мережу, що не підтримує протокол маршрутизації. GRE надає здатність упаковувати різнорідний трафік в оболонку тунелю, що дозволяє йому перетинати різні типи мереж та пристосовуватися до різноманітних сценаріїв використання. Однією з ключових характеристик GRE є його універсальність у тунелюванні різних протоколів та мережевих пакетів. GRE може обгортати пакети різних протоколів, таких як IPv4, IPv6, IPX (Internetwork Packet Exchange), і навіть мережеві кадри Ethernet. Це забезпечує гнучкість та адаптабельність для передачі різних видів даних через вузькі тунелі. Протокол GRE оперує на другому рівні моделі OSI (Data Link Layer) і не має власних механізмів шифрування чи безпеки. Однак його основною метою є надання механізму тунелювання, а не шифрування даних. В сценаріях, де потрібна безпека, часто GRE використовується разом із іншими протоколами, такими як IPsec (Internet Protocol Security), для надання шару шифрування та аутентифікації. Однією з основних переваг GRE є його простота та легкість в конфігурації. Зазвичай налаштовування тунелю GRE включає в себе вказання точки на початку та кінці тунелю, а також параметрів маршрутизації. Це робить GRE привабливим вибором для операторів мереж, які шукають простий та

ефективний спосіб підключення віддалених мереж без великої складності налаштувань. Ще однією перевагою GRE є його здатність працювати з різними протоколами маршрутизації, такими як RIP (Routing Information Protocol), OSPF (Open Shortest Path First) та EIGRP (Enhanced Interior Gateway Routing Protocol). Це робить GRE витриманим та сумісним з різноманітними мережевими середовищами. Однак, не дивлячись на свої переваги, GRE також має свої обмеження та виклики. Наприклад, через відсутність власних механізмів шифрування, GRE не надає конфіденційності даних, що передаються через тунель. Це робить його менш підходящим для сценаріїв, де конфіденційність є критичною.Також важливо враховувати, що GRE тунелює всі пакети без вибірковості, що може призводити до надмірного навантаження на мережевий канал у разі великої кількості різнорідних пакетів. Це може впливати на ефективність мережі, особливо у випадках обмежених пропускних здатностей. Усупереч своїм обмеженням, GRE залишається одним з популярних протоколів тунелювання, особливо у великих корпоративних мережах та вирізняється своєю простотою та універсальністю. Здатність GRE транспортувати різнорідний трафік через мережі зробила його важливим елементом для забезпечення зв'язку та інтеграції великих мережевих інфраструктур.

2.3 Протокол OpenVPN

OpenVPN є відкритим рішенням для створення VPN серверу, що забезпечує безпечне та шифроване з'єднання з Інтернетом. З моменту його створення в 2001 році проект отримав широку підтримку від численних розробників та співавторів. Завдяки активній участі спільноти з відкритим кодом, OpenVPN став визнаним стандартом на полі мереж з відкритим кодом. З більш як 60 мільйонами завантажень це широко використовувана VPN-технологія, яка дозволяє користувачам створювати безпечні та зашифровані канали зв'язку. Важливо відзначити, що OpenVPN ефективно працює навіть при підключенні до

потенційно небезпечних або ненадійних мереж, таких як громадські точки доступу Wi-Fi. OpenVPN забезпечує захист з'єднань за допомогою спеціального протоколу безпеки, який базується на використанні SSL/TLS для обміну ключами. Цей підхід робить дані, які передаються через Інтернет, шифрованими та конфіденційними. Система надає гнучкість і масштабованість завдяки підтримці різних конфігурацій мережі. Вона може працювати з динамічними загальнодоступними кінцевими точками, мережами, що проходять через брандмауери, а також мережами, які використовують NAT. OpenVPN використовується для розгортання масштабованих VPN-серверів з збалансованим навантаженням. Вам доступні різні методи шифрування, включаючи традиційне шифрування на основі статичного ключа або шифрування з відкритим ключем на основі сертифіката від OpenVPN, для налаштування розширюваного сервера VPN з збалансованим навантаженням.

Оскільки OpenVPN є проектом з відкритим кодом, будь-хто може мати доступ до його коду та оцінити його. Це призвело до створення екосистеми програмістів, інженерів і ентузіастів, які систематично проводять тести, розвивають і оновлюють протокол. Ця спільнота сприяє безпеці, надійності та стабільності OpenVPN на протязі тривалого періоду часу.

OpenVPN – це програмне забезпечення та протокол, які створюють зашифровані з'єднання для забезпечення безпечного обміну даними через мережу та Інтернет. Головним призначенням OpenVPN є забезпечення технології VPN. VPN використовується для створення приватного та безпечного каналу зв'язку через незахищену або загальнодоступну мережу.

OpenVPN дозволяє користувачам безпечно та конфіденційно отримувати доступ до ресурсів, обмінюватися інформацією та переглядати Інтернет. За допомогою методів і протоколів шифрування, він кодує пакети даних, зробивши їх непрочитуваними для сторонніх осіб. Цей шифрувальний процес гарантує, що навіть у випадку перехоплення даних під час передачі, вони залишаються нерозшифрованими без відповідних ключів.

OpenVPN створює зашифровані тунелі між пристроями або мережами, забезпечуючи секретність, цілісність та автентичність передачі даних між кінцевими точками. Ці тунелі ефективно ізолюють пакети даних від впливу загальнодоступної мережі, створюючи безпечний шлях для комунікації. Взаємна аутентифікація вузлів відбувається за допомогою різних методів, таких як попередньо спільні секретні ключі, сертифікати або пари ім'я користувача-пароль. У випадку системи із кількома клієнтами та серверами, OpenVPN дозволяє серверу видачувати сертифікати для автентифікації кожного клієнта, використовуючи цифрові підписи та центр сертифікації. Програма працює через протоколи транспорту користувача (UDP) або керування передачею (TCP), мультиплексуючи створені SSL-тунелі на одному порті TCP/UDP.

OpenVPN визнаний як передовий вибір для VPN-підключень завдяки своїм різноманітним функціям. Клієнти мають можливість індивідуально налаштувати OpenVPN згідно зі своїми потребами, контролюючи параметри, такі як шифрування, шифри, структура мережі та інші. Це економічне рішення, яке сумісне з основними операційними системами.

OpenVPN вважається одним з найнадійніших протоколів VPN завдяки своїм покращеним заходам захисту. Застосування 256-бітних ключів шифрування та вдосконалених шифрів робить перехоплення та маніпулювання пакетами даних важчим завданням для зловмисників. OpenVPN підтримує різноманітні методи шифрування та надає складні механізми безпеки, такі як автентифікація HMAC, OpenSSL і використання спільних ключів. Його використання ідеальної прямої секретності гарантує, що навіть у випадку отримання закритого ключа, зловмисники не зможуть розшифрувати попередні повідомлення. Модель клієнт-сервер допомагає створити безпечний канал між VPN-клієнтом та сервером, забезпечуючи стабільне та безпечне підключення. OpenVPN ефективно обходить брандмауери завдяки своїй адаптивності до протоколів TCP і UDP, особливо якщо використовується порт TCP 443, що робить трафік VPN майже невідличим від звичайної активності в Інтернеті.

Протягом понад двох десятиліть OpenVPN встановив стандарти в галузі VPN-протоколів. Привертаючи до себе значну та активну спільноту користувачів і розробників, він став надійним та популярним вибором для багатьох. Завдяки тривалій історії та підтримці спільноти, OpenVPN вважається стійким та довіреним рішенням. Користувачі можуть розраховувати на велику групу фахівців, які готові надавати допомогу та ресурси для вирішення питань та налаштування.

Код OpenVPN пройшов аудит і систематично виправляються всі виявлені проблеми безпеки, що робить його одним з найбільш надійних варіантів. Цей протокол часто використовується для налаштування маршрутизаторів, надаючи клієнтам можливість захистити всю свою мережу. Завдяки своєму популярному та широко визнаному статусу, OpenVPN залишається одним із провідних інструментів для забезпечення безпеки та конфіденційності в інтернет-з'єднаннях.

OpenVPN ставить в центр своєї політики конфіденційності прозорість, захист особистих даних та забезпечення користувачів контролю над своєю особистою інформацією. Сервіс не реєструє деталі трафіку чи вміст повідомлень користувачів, не обмежує швидкість Інтернет-підключення та не проводить глибоку або поверхневу перевірку пакетів трафіку, за винятком ситуацій, коли це необхідно для виконання вимог брандмауера.

Збір та використання особистих даних в OpenVPN спрямовані на забезпечення та удосконалення послуг. Ці дані надходять від користувачів добровільно чи представлені анонімними загальними даними, зокрема за допомогою файлів cookie та аналітики веб-сайту. OpenVPN зберігає особисті дані лише стільки часу, скільки необхідно для виконання цілей політики конфіденційності. Забезпечуються відповідні технічні та організаційні заходи безпеки для захисту та забезпечення конфіденційності даних клієнтів.

OpenVPN не передає та не розкриває інформацію про користувачів третім особам, якщо це необхідно згідно з вимогами закону. Користувачі мають право вимагати доступу, виправлення, оновлення чи видалення своїх особистих даних.

Вони також можуть відмовитися від обробки своїх особистих даних, обмежити її обробку або здійснити перенесення цих даних.

Відкритий вихідний код дизайну OpenVPN підвищує рівень прозорості та безпеки, оскільки його код відкритий для громадськості. Це гарантує відсутність схованих дефектів або ризикованих конструкцій, що можуть порушити конфіденційність користувачів. Проект дотримується Закону про захист конфіденційності дітей в Інтернеті (COPPA) і свідомо не збирає особисту інформацію від осіб віком до тринадцяти років. Щоб захистити особисту інформацію, OpenVPN реалізує стандартні умови договору для передачі даних з Європейської економічної зони (ЄЕЗ).

Політика конфіденційності OpenVPN гарантує, що конфіденційна чи кредитна інформація передається за допомогою технології Secure Socket Layer (SSL) і подальше шифрується в базі даних постачальника платіжного шлюзу. Доступ до цієї інформації надається лише особам, які мають спеціальні права доступу і зобов'язані дотримуватися вимог конфіденційності. OpenVPN також надає рекомендації з безпеки, такі як захист облікового запису користувача root, регулярне оновлення сервера доступу, захист користувача-адміністратора для веб-інтерфейсу адміністратора, встановлення сертифіката SSL для веб-інтерфейсу та посилення рядка набору веб-шифрів.OpenVPN, незважаючи на свою популярність та визнану безпеку, має деякі недоліки, які варто враховувати:

Через використання вдосконалених методів шифрування, може виникати проблема повільності в порівнянні з іншими протоколами, такими як WireGuard.

Процес завантаження та налаштування клієнтського програмного забезпечення може бути не зручним для всіх категорій користувачів, що вимагає уважності та часу.

Інтеграція OpenVPN не є універсальною для всіх пристроїв, і користувачам доводиться встановлювати сторонні клієнтські програми VPN.

Використання TCP-порту 443 може призвести до блокування з боку брандмауерів, особливо у навчальних закладах та підприємствах, оскільки цей порт використовується також для підключень HTTPS.

Можливість значних накладних витрат та проблем з проксі може впливати на ефективність використання OpenVPN.

OpenVPN в основному безпечний протокол для VPN. Визначаючи методи шифрування та обмін ключами, він дозволяє налаштовувати VPN-з'єднання. Представлене програмне забезпечення OpenVPN надає можливість індивідуально конфігурувати сервер чи мережу VPN.

Один з основних приводів популярності OpenVPN – використання відкритих методів шифрування. Він утворює безпечні точка-точка або мережеві VPN-з'єднання, використовуючи свій власний протокол безпеки на основі SSL/TLS для обміну ключами. Операції обміну ключами SSL/TLS забезпечують застосування шифрування при створенні VPN-тунелів. Стандартно використовується 256-бітне шифрування, що забезпечує високий рівень безпеки. Протокол підтримує передові методи шифрування, такі як Blowfish, AES і CAST-128. Це робить передачу даних майже неуразливою для зовнішніх атак.

OpenVPN використовує транспортні протоколи UDP або TCP для тунелювання. Це дозволяє зробити веб-трафік важко відрізнити від звичайного трафіку HTTPS через SSL, ускладнюючи виявлення та блокування. Протокол також застосовує Perfect Forward Secrecy (PFS), генеруючи для кожного сеансу окремий ключ шифрування. Використання PFS робить вкрай важким викрадення ключів та обхід шифрів шифрування.

Тепер порівняємо OpenVPN проти інших протоколів.

SSTP і OpenVPN подібні за деякими параметрами, такими як використання SSL 3.0 та можливість працювати на порту 443. Обидва протоколи забезпечують високий рівень безпеки, використовуючи 256-бітове шифрування та шифр AES. Однак різниці між ними можуть визначати вибір для користувача. OpenVPN відзначається своєю відкритою вихідною кодовою базою, що робить його набагато

надійнішим порівняно з SSTP, який є продуктом виключно Microsoft. Останній, відомий співпрацею з різними урядовими агентствами, може викликати певні сумніви щодо конфіденційності. У плані сумісності з брандмауерами OpenVPN виявляється більш привабливим, оскільки SSTP, за даними Microsoft, не підтримує автентифіковані веб-проксі. Це може робити SSTP вразливим до втручання мережевого адміністратора через проксі-сервер. Щодо швидкості, хоча стверджується, що SSTP швидший за OpenVPN, об'єктивних доказів цьому надто багато немає. OpenVPN може використовувати порт UDP, що, за загальними висновками, пропонує швидший обмін даними. Ще однією важливою різницею є міжплатформенна сумісність. Тут OpenVPN виходить переможцем, оскільки він працює на більшій кількості платформ порівняно з SSTP, який доступний переважно на Windows, Linux, Android і маршрутизаторах. Однак важливо відзначити, що SSTP вбудовано в платформи Windows, що робить його легше налаштовуємим порівняно з OpenVPN. Таким чином, вибір між SSTP і OpenVPN може залежати від конкретних потреб користувача, зокрема, його переконань у справі конфіденційності, потреби в міжплатформенній сумісності та вимог до швидкості обміну даними.

OpenVPN і WireGuard представляють дві різні парадигми щодо використання криптографії в протоколах VPN. OpenVPN використовує бібліотеку OpenSSL для реалізації різноманітних криптографічних алгоритмів, і основним серед них є популярний AES-256. Використання OpenSSL дозволяє користувачам вибирати різні алгоритми шифрування та налаштовувати їх за необхідність. OpenVPN славиться своєю здатністю надати високий рівень безпеки завдяки ретельному вибору криптографічних методів. Наступний є WireGuard, який використовує сучасні фіксовані алгоритми для уникнення можливих помилок у конфігурації, які можуть стати джерелом безпекових вразливостей. WireGuard відрізняється від OpenVPN своєю легкістю та ефективністю. Зменшена кодова база (приблизно 4000 рядків порівняно з 70 000–600 000 рядків в OpenVPN) робить WireGuard більш легким для аудиту та обслуговування. Щодо

продуктивності, WireGuard визначається своєю значною швидкістю порівняно з OpenVPN. Його ефективність використання ядра ЦП та легка кодова база дозволяють йому працювати швидше, особливо при використанні OpenVPN через UDP. У результаті проведених тестів WireGuard показав вражаючу швидкість передачі даних.

Protocol	Configuration
WireGuard	256-bit ChaCha20, 128-bit Poly1305
IPsec #1	256-bit ChaCha20, 128-bit Poly1305
IPsec #2	256-bit AES, 128-bit GCM
OpenVPN	256-bit AES, HMAC-SHA2-256, UDP mode

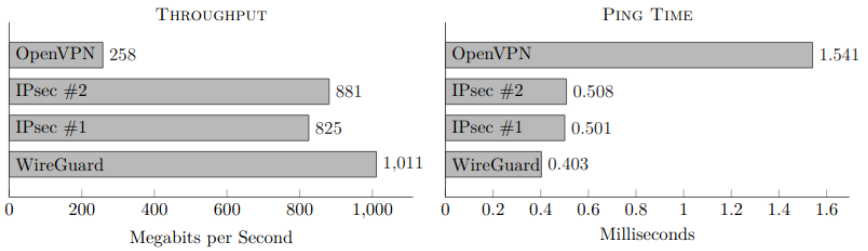


Рис.2.3 - Порівняння продуктивності різних VPN від розробника

У обох тестах, які включали вимірювання пропускної здатності і часу відгуку пінгу, WireGuard показав значно кращі результати, ніж OpenVPN і два різновиди IPsec. Ці тести також показали, що використання OpenVPN і IPsec сильно навантажує центральний процесор, і в деяких випадках відбувається велика витрата ресурсів процесора. WireGuard, натомість, працює більш ефективно і не так сильно навантажує процесор, що дає можливість більш повною мірою використовувати можливості мережевої карти GigabitEthernet.

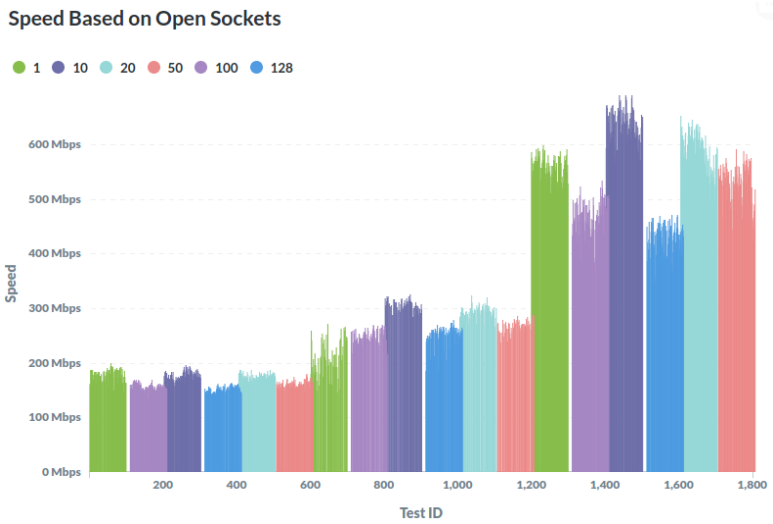


Рис.2.4 - Порівняння продуктивності OpenVPN та Wireguard від незалежного тестувальника

Отже, обидва протоколи забезпечують високий рівень безпеки, але їхні особливості використання криптографії та продуктивність роблять їх придатними для різних сценаріїв використання.

L2TP/IPSec є доступним на багатьох платформах, що робить його налаштування значно простішим порівняно з OpenVPN. Однак, коли ви вже використовуєте службу VPN, різницю між ними ви, ймовірно, не відчуєте. L2TP/IPSec використовує менше портів, ніж OpenVPN, і відсутність використання порту 443 робить його менш вразливим перед брандмауерами NAT. Незважаючи на те, що L2TP/IPSec не є повною власністю Microsoft (оскільки був розроблений також Cisco), він все ж не досягає рівня надійності, який може запропонувати OpenVPN, який є відкритим кодом. Треба також відзначити, що за словами Едварда Сноудена, L2TP був навмисно послаблений АНБ. Одним з ключових аспектів безпеки є те, що L2TP сам по собі не надає шифрування. Це зумовило його поєднання з IPSec для забезпечення надійного рівня захисту. Крім того, навіть не зважаючи на те, що OpenVPN на TCP може вимагати ресурсів, L2TP/IPSec також може бути високонавантаженим (залежно від потужності вашого пристрою), оскільки він подвійно інкапсулює дані. Отже, вибір між

OpenVPN і L2TP/IPSec варіюється в залежності від ваших потреб у зручності налаштування та рівня надійності та безпеки, які ви шукаєте.

IPSec часто використовується в парі з протоколами L2TP та IKEv2, але існують постачальники VPN, які дозволяють використовувати цей протокол самостійно.У порівнянні з протоколом OpenVPN, IPSec також забезпечує високий рівень безпеки. Однак його налаштування вимагає обережності, оскільки невірно виконані кроки можуть підірвати його захист. Цей протокол також працює у просторі ядра, що може обмежити його безпеку в залежності від налаштувань постачальника. В порівнянні з OpenVPN, який використовує простір користувача, IPSec менш портативний.

Щодо доступності, IPSec підтримується на багатьох платформах, тоді як OpenVPN може вимагати ручного налаштування на деяких з них. Треба відзначити, що деякі брандмауери можуть блокувати трафік IPSec, в той час як OpenVPN UDP або TCP зазвичай не має таких проблем.

Щодо швидкості та стабільності, обидва протоколи є прийнятними, за умови наявності достатньої пропускної здатності та потужного пристрою. Але важливо враховувати, що ініціалізація тунелю IPSec може займати більше часу, ніж у випадку OpenVPN.

Протоколи OpenVPN і IKEv2 вважаються безпечними, але важливо враховувати деякі відмінності між ними. OpenVPN використовує TLS/SSL для захисту даних на транспортному рівні, тоді як IKEv2 захищає дані на мережевому. Ця різниця, хоча й невелика, може бути значущою в певних сценаріях. Зазначимо, що, хоча існують відкриті реалізації IKEv2, розроблені Cisco та Microsoft, вони не такі поширені, як OpenVPN.

OpenVPN видається більш універсальним у сенсі сумісності між платформами. З іншого боку, IKEv2 зазвичай є популярним серед мобільних користувачів, особливо тих, що використовують пристрої BlackBerry, оскільки він інтегрований у ці пристрої. Важливо відзначити, що IKEv2 може забезпечити

кращу стабільність в умовах зміни мережі, дозволяючи зберігати з'єднання при перемиканні між різними типами підключень.

Щодо швидкості, IKEv2 зазвичай працює швидше, але його можна легше заблокувати, оскільки він використовує UDP-порт 500. У порівнянні OpenVPN, який часто використовує порт 443, це може зробити його більш вразливим до блокування брандмауерами.

Загалом, якщо ви часто користуєтеся мобільним телефоном, особливо в подорожах, то IKEv2 може бути кращим вибором. В інших випадках OpenVPN може бути більш практичним і універсальним варіантом.

2.4 Визначення ролі Python в моніторингу

Python відіграє одну з ключових ролей в моніторингу сервера OpenVPN через інтеграцію з Prometheus, забезпечуючи автоматизований збір метрик заснованих на аналізі файлів сервера. В даній кваліфікаційній роботі сценарії python будуть створювати кастомні метрики на основі даних з файлів та виводі утиліт командного рядка сервера. Це дозволить покращити відмовостійкість та роботу сервера OpenVPN. В даній роботі буде розглянуто два сценарії python, які будуть за допомогою стандартних модулів бібліотеки prometheus_client створювати метрики для Prometheus, а також буде реалізовано захист новостворених метрик за допомогою базової аутентифікації та перенесенню даних у веб-інтерфейсі на TLS/SSL на основі фреймворку Flask. Використовуючи утиліти командного рядка сценарій буде перевіряти скільки часу залишилось до закінчення строку дії кореневого, серверного сертифікатів OpenVPN , а також буде реалізована перевірка можливості підключення до OpenVPN сервера з віддаленого присторю. Дані сценарії не являються широкодоступними і тому, можливо, в даній роботі будуть використані вперше.

Створення і підтримка з'єднання з сервером OpenVPN є критичною для ефективної роботи високонавантажених інфраструктур. Перевірка доступності

цього з'єднання важлива для негайного реагування на будь-які виникнення проблем. Використання мови програмування Python у поєднанні з інструментами моніторингу, такими як Prometheus, дозволяє автоматизувати цей процес і надає зручний і ефективний спосіб слідкувати за станом OpenVPN-з'єднань.

В свою чергу перевірка строку закінчення дії сертифікатів допоможе уникнути виникненню несподіваних інцидентів і допоможе завчасно підготуватись до них.

2.5 Unix-подібні операційні системи

Операційна система (ОС) - це комплекс взаємопов'язаних системних програм, які забезпечують контроль використання та розподіл ресурсів комп'ютерної системи і надають можливість взаємодії користувача з комп'ютером. ОС виконує ряд ключових функцій, включаючи керування процесами, введенням-виведенням, пам'яттю, файловою системою та мережами.

ОС дозволяє користувачам та програмам взаємодіяти з апаратним забезпеченням комп'ютера, надаючи абстракцію від конкретних деталей апаратури. Вона також забезпечує ефективне використання ресурсів, планування виконання завдань, забезпечення безпеки та координацію роботи всіх компонентів системи. ОС можуть бути різних типів, таких як Windows, macOS, Linux, або Unix-подібні системи. Кожна з них має свої особливості та призначення, і вони використовуються в різних областях, включаючи персональні комп'ютери, сервери, мобільні пристрої та інші вбудовані системи. ОС є складною системою взаємопов'язаних програм, які відповідають за управління ресурсами комп'ютера та взаємодію з користувачем. Сучасні Unix-подібні ОС, такі як BSD, AIX, IRIX, HP-UX, GNU/Linux та інші, працюють на різних архітектурах.

Unix-подібні ОС відзначаються своєю гнучкістю, стабільністю та масштабованістю. Розвиток Unix почався в 1960-х роках в лабораторії Bell в AT&T. Однією з ключових особливостей Unix-подібних ОС є їхня здатність

працювати на різних архітектурах комп'ютерів. Це робить їх універсальними та придатними для використання на різних пристроях, починаючи від персональних комп'ютерів і закінчуючи вбудованими системами та суперкомп'ютерами. У світі сучасних технологій Unix-подібні ОС необхідні для функціонування різних пристроїв та систем. Вони використовуються в інтернет-серверах, хмарних обчисленнях, мобільних платформах та інших областях. Крім того, їхня стабільність та безпека роблять їх популярними в корпоративному середовищі. Unix-подібні ОС також активно розвиваються та підтримуються великою спільнотою розробників. Вони надають можливість налаштовувати систему під конкретні потреби та впроваджувати нові технології.

Однак, варто відзначити, що для деяких користувачів використання командного рядку може стати викликом, і тому деякі Unix-подібні системи включають графічний інтерфейс для полегшення взаємодії користувача з ОС.

Linux, вийшовши в офіційний реліз в середині 1990-х років, швидко завоював популярність та поширення в усьому світі. Ця операційна система не лише присутня в комп'ютерах, але й в телефонах, автомобілях, телевізорах та інших пристроях.

В сучасному світі операційні системи конкурують за лідерство на ринку персональних комп'ютерів. Windows є найпопулярнішою серед домашніх користувачів завдяки зручному графічному інтерфейсу. Linux, хоча і має графічний інтерфейс, акцентується на командному рядку, що може бути викликом для новачків. Проте, на серверних платформах Linux виявляється дуже потужним завдяки ефективному управлінню ресурсами та високій надійності.

2.6 Популярні Unix системи

Під час еволюції операційної системи Linux виникло значна кількість варіацій у вигляді різних дистрибутивів. Кожен з цих дистрибутивів вирізняється своїми унікальними можливостями, які відображають погляди та потреби різних

розробників. Однак з часом виникла потреба у створенні єдиної реалізації, яка б об'єднала ключові новації та покращення з різних дистрибутивів.

Операційна система MacOS є однією з найпоширеніших та динамічно розвиваючихся систем у сучасному світі, визначаючи нові стандарти у користувацькому досвіді та функціональності. Заснована на відкритому ядрі Darwin, MacOS інтегрує ключові аспекти Unix-систем, такі як надійність та стабільність, з унікальними рішеннями, розробленими спеціально для пристроїв компанії Apple. Однією з найбільш визначних особливостей MacOS є його користувацький інтерфейс Aqua, який пропонує не тільки ефективне управління системою, але й естетично приємний зовнішній вигляд. Інтуїтивно зрозумілий та легко використовуваний інтерфейс є однією з причин популярності MacOS серед широкого кола користувачів, включаючи як досвідчених інженерів, так і новачків. Важливо відзначити, що MacOS активно використовується в екосистемі компанії Apple, де висока інтеграція між апаратним та програмним забезпеченням забезпечує оптимальну продуктивність. Завдяки цьому, пристрої лінійки Mac відрізняються не тільки потужністю, але й довговічністю, що робить їх популярними вибором для різних професійних сегментів, включаючи дизайнерів, редакторів відео та розробників програмного забезпечення. Велика увага приділяється інноваціям, що визначають MacOS як передову операційну систему. Голосовий асистент Siri входить в численні функції, дозволяючи користувачам ефективно керувати пристроями за допомогою голосових команд. Це допомагає використовувачам отримати доступ до інформації, керувати розкладами та виконувати завдання безпосередньо за допомогою голосу, роблячи взаємодію з системою більш природною та зручною. Важливим аспектом безпеки MacOS є система Gatekeeper, яка відповідає за контроль над програмним забезпеченням, що встановлюється на пристроях. Gatekeeper забезпечує безпечне завантаження та встановлення додатків, що допомагає запобігти можливим загрозам безпеки та зберегти цілісність системи. Однією з ключових переваг MacOS є також підтримка нової файлової системи APFS (Apple File System). APFS визначається високою

швидкодією та надійністю, а також здатністю працювати зі сучасними технологіями зберігання даних, такими як SSD (Solid State Drive). Ця файлова система забезпечує ефективне управління даними та покращує продуктивність системи в цілому. Важливо відмітити, що MacOS є комерційним продуктом, і його вихідний код закритий. Це ставить операційну систему в рядок платних рішень, але в той же час дозволяє забезпечити високий рівень підтримки та якості від компанії-розробника. Узагальнюючи, MacOS не лише представляє собою оптимально відшліфовану операційну систему, але й є екосистемою, що включає в себе апаратні та програмні рішення, які працюють в гармонії. Завдяки цьому MacOS залишається популярним вибором для тих, хто цінує високий рівень якості та вдалу інтеграцію функціональності з елементами дизайну та інновацій.

Debian, який визначається як один із найдавніших та популярних дистрибутивів операційної системи GNU/Linux, є результатом спільної розробки великої відкритої спільноти і базується на ядрі Linux. Debian вирізняється своєю відкритою філософією, забезпечуючи безкоштовне програмне забезпечення з відкритим кодом. Це означає, що користувачі мають можливість вільно використовувати, модифікувати та розповсюджувати Debian, спираючись на принципи вільності та доступності. Однією з визначальних характеристик Debian є його стабільність, яка є результатом уважного контролю якості та тестування перед випуском нових версій. Ця особливість дозволяє користувачам впевнено використовувати Debian в серйозних та вимогливих обчислювальних середовищах. Однак Debian також славиться своєю гнучкістю та адаптабельністю. Завдяки великій базі пакетів, яка включає тисячі програм та інструментів, користувачі можуть легко налаштовувати свої системи відповідно до своїх унікальних потреб та вимог. Ще однією значущою рисою Debian є його роль в якості основи для численних інших дистрибутивів, у тому числі популярного Ubuntu. Ubuntu використовує Debian як вихідну точку, додаючи власні інновації та інтерфейси користувача, але водночас спадаючи на базову стабільність та надійність Debian. Поміж різноманіттям функцій та переваг Debian варто

відзначити його активну та велику спільноту розробників, яка надає підтримку, вносить внески до розвитку та регулює процеси прийняття рішень у спільноті. У величезному світі операційних систем GNU/Linux, Debian відіграє значущу роль, надаючи користувачам надійність, стабільність та вільність у використанні. Його постійна еволюція та вдосконалення відображають принципи спільноти вільного програмного забезпечення, роблячи Debian ключовим учасником у світі відкритих технологій.

Ubuntu – це відомий та широко використовуваний дистрибутив операційної системи GNU/Linux, який базується на Debian. Розроблений та підтримується компанією Canonical, Ubuntu отримав визнання завдяки своїй активній спільноті та легкому доступу до різноманітного безкоштовного програмного забезпечення. Його філософія визначається бажанням зробити Linux доступним та зручним для різних користувачів. Однією з ключових особливостей Ubuntu є його походження від Debian, що надає йому стабільну та надійну основу. Canonical взяла базовий фундамент Debian і внесла власні інновації, забезпечуючи вищий рівень зручності та користувацької дружби. Серед переваг Ubuntu слід відзначити активну спільноту, що активно підтримує та розвиває дистрибутив. Спільнота вносить свої внески до розвитку системи, надає підтримку користувачам і сприяє поширенню знань про використання Ubuntu. Ключова філософія Ubuntu полягає в тому, щоб робити операційну систему Linux доступною для широкого кола користувачів, незалежно від їхнього рівня експертизи. Простий та інтуїтивно зрозумілий інтерфейс Ubuntu дозволяє новачкам швидко адаптуватися та використовувати систему, тоді як для досвідчених користувачів зберігається висока гнучкість налаштувань. Важливою рисою Ubuntu є також його швидка інсталяція та простий процес оновлення системи. Це робить дистрибутив привабливим для тих, хто цінує ефективне використання часу та мінімізацію зусиль при встановленні та підтримці операційної системи. Ubuntu активно використовується як на робочих станціях, так і на серверах. Для більшості користувачів Ubuntu надає повноцінний функціонал, включаючи офісні програми, мультимедійні інструменти,

інтернет-браузери та інші необхідні додатки. Окрім того, Ubuntu є популярним вибором для розробників програмного забезпечення. Інтеграція з рядом інструментів для розробки, включаючи підтримку різних мов програмування та інструменти контролю версій, робить його зручним середовищем для створення програмних продуктів.

Linux Mint представляє собою інший дистрибутив операційної системи GNU/Linux, що базується на Debian та Ubuntu. Відрізняючись своєю спрощеним використанням та стильним дизайном, Linux Mint націлено на створення зручної та добре підтримуваної операційної системи для користувачів різних рівнів досвіду. Цей дистрибутив визначається своєю особливою фокусованістю на простоті використання, що робить його особливо привабливим для новачків у світі Linux, а також для тих, хто шукає ефективний та стабільний варіант операційної системи. Linux Mint будується на фундаменті Debian та Ubuntu, використовуючи їхні основи та додаючи власні розробки та інновації. Такий підхід дозволяє поєднати надійність Debian з додатковими можливостями, які приносить світ Ubuntu. Однією з ключових переваг Linux Mint є його естетичний дизайн, який включає в себе яскравий та інтуїтивно зрозумілий інтерфейс. Це сприяє створенню приємного користувацького досвіду та полегшує перехід від інших операційних систем. Дистрибутив активно розвивається спільнотою, що вносить свої внески до розширення функціоналу та покращення загальної продуктивності. Велика кількість корисних програм та функцій, які інтегровані в Linux Mint, дозволяють користувачам ефективно використовувати операційну систему для різних завдань, від роботи з офісними документами до розваг та редагування мультимедійного контенту. Важливим аспектом Linux Mint є його прагнення надати повноцінний робочий стіл для користувачів, що охоче переходять з інших операційних систем. Тут враховуються потреби користувачів у зручності та доступності, що робить дистрибутив дуже конкурентоспроможним.

Дистрибутиви, які були описані, займають значне положення на ринку операційних систем і забезпечують високий рівень функціональності,

відповідаючи на потреби різних користувачів. Кожен з них володіє унікальними особливостями та можливостями, що дозволяє вибрати оптимальний варіант в залежності від конкретних потреб і вимог. Linux є операційною системою, яка використовується на різних платформах, включаючи популярну мобільну платформу Android. Ця операційна система необхідна для ефективного управління апаратними ресурсами комп'ютера та забезпечення взаємодії між програмним забезпеченням та апаратурою. Linux складається з кількох ключових компонентів, які спільно забезпечують його функціональність. Початковий етап - це завантажувач, який управляє процесом запуску комп'ютера, представлений користувачам у вигляді заставки під час запуску системи. Найважливішою частиною є ядро, що керує процесором, пам'яттю та периферійними пристроями. Воно представляє собою найнижчий рівень операційної системи. Після завантаження ядра діє система ініціалізації, яка завантажує користувацький простір та управляє демонами - фоновими службами, що працюють під час завантаження або після входу на робочий стіл. Демони, такі як служби друку, звуку, та інші, автоматично запускаються під час завантаження системи або після входу користувача. Графічний сервер (X-сервер або X) відповідає за відображення графічного інтерфейсу на моніторі. Середовище робочого столу - це та частина, з якою користувачі фактично взаємодіють. Різноманітні варіанти, такі як GNOME, KDE, Xfce, мають унікальні риси та додаткові програми. Крім того, Linux пропонує різноманіття програм для встановлення через вбудовані центри програм у більшості дистрибутивів.

Операційна система Linux впроваджує гармонійну взаємодію ключових компонентів, спрямованих на забезпечення зручності, надійності та ефективності. Кожен з цих елементів відповідає за конкретний аспект функціонування системи, створюючи злагоджене середовище для задоволення різних потреб користувачів. У додаток до основних компонентів, Linux має велику кількість додаткових елементів, що розширюють функціональні можливості та підвищують продуктивність. Ці компоненти працюють у співробітництві, сприяючи

різноманітній функціональності операційної системи. Всі різновиди Linux-дистрибуцій володіють власною унікальною системою керування пакетами, яка є необхідною для забезпечення ефективного управління програмним забезпеченням. Ця інтегрована система спрощує процеси встановлення, оновлення та видалення програм та пакетів з великого асортименту доступного програмного забезпечення. Наприклад, Ubuntu використовує передовий інструмент для упаковки APT (Advanced Packaging Tool), в той час як Fedora користується DNF (Dandified YUM) для забезпечення ефективного управління пакетами. Цей пакетний менеджер є важливою складовою, що дозволяє користувачам легко та швидко встановлювати необхідне програмне забезпечення, проводити оновлення для забезпечення безпеки та отримання нових функцій, а також безболісно видаляти програми, які вже не потрібні. Це сприяє зручності в управлінні системою та її функціональністю, забезпечуючи високий рівень користувацького досвіду. Незважаючи на те, що основна мета пакетного менеджера полягає в ефективному розподілі та керуванні програмами, його функціональні можливості виявляються набагато ширшими. Він також грає ключову роль у забезпеченні сумісності програм між різними версіями операційної системи, а також у розв'язанні можливих конфліктів у процесі встановлення та оновлення. Це робить пакетний менеджер невід'ємною частиною структури Linux, забезпечуючи гнучкість та надійність у використанні операційної системи. Лінукс надає можливість взаємодії з операційною системою через інструмент, відомий як термінал або командний рядок. Цей текстовий інтерфейс відкриває перед користувачем широкі можливості для контролю над системою та налаштувань, роблячи операційну систему надзвичайно гнучкою та потужною.

Термінал Linux є ключовим засобом для взаємодії з операційною системою. Завдяки йому користувач може вводити команди, що спрощує виконання різноманітних завдань та операцій. Такий підхід дозволяє отримати більший рівень контролю над системою порівняно з графічним інтерфейсом, що може бути

особливо важливим для досвідчених користувачів та системних адміністраторів. Термінал Linux може бути використаний для виконання широкого спектру завдань, включаючи встановлення та оновлення програм, налаштування системних параметрів, адміністрування мережі та багато інших. Відкритий доступ до командного рядка дозволяє користувачам глибше вникнути в структуру операційної системи, використовуючи різноманітні команди та сценарії для досягнення своїх цілей. Важливо підкреслити, що термінал в Linux є не лише інструментом для введення команд, але й потужним засобом автоматизації завдань. Сценарії команд можуть бути складені для виконання послідовності дій, що робить його не тільки зручним для ручного введення команд, але і для автоматизації повторюваних завдань та оптимізації робочого процесу. Застосування терміналу в Linux також важливе з точки зору управління ресурсами та відладки. Користувачі можуть використовувати різноманітні команди для моніторингу системних ресурсів, аналізу даних та виявлення можливих проблем. Одним з важливих аспектів термінала є його можливість працювати у віддаленому режимі. Це дозволяє адміністраторам або користувачам взаємодіяти з системою, навіть якщо вони фізично знаходяться в іншому місці. Ця функція стає важливою для великих підприємств або серверних конфігурацій, де важливо мати дистанційний доступ та управління системою. У Лінукс використовуються різноманітні бібліотеки, які є ключовими елементами для розвитку та функціонування операційної системи. Ці бібліотеки є невід'ємною частиною програмної інфраструктури Linux, надаючи розробникам доступ до різноманітних функцій та процедур, які допомагають оптимізувати роботу програм та покращити їх ефективність. Однією з ключових характеристик бібліотек в Linux є їх здатність розширювати функціональність операційної системи. Це досягається завдяки включенню в бібліотеки різноманітних функцій, які можуть бути використані програмами для взаємодії з різними аспектами операційної системи. Такий підхід створює основу для розширення можливостей операційної системи та забезпечує сприятливе середовище для розробки різноманітних програм. Бібліотеки в Linux є

важливим елементом для покращення ефективності розробки програм. Розробники можуть використовувати ці бібліотеки для виклику готових функцій та процедур, що спрощує процес розробки та дозволяє уникнути дублювання коду. Використання бібліотек також сприяє створенню більш структурованих та модульних програм, що полегшує їх обслуговування та розширення в майбутньому. Ще однією важливою характеристикою бібліотек є їх роль у забезпеченні стандартизації та сумісності програм. Багато з цих бібліотек відповідають за реалізацію стандартів, які забезпечують уніфікований підхід до розробки програм на платформі Linux. Це дозволяє забезпечити, що програми, розроблені з використанням цих бібліотек, будуть сумісні з різними дистрибутивами Linux та різними версіями операційної системи. Важливо також відзначити, що бібліотеки в Linux розвиваються та оновлюються разом з розвитком операційної системи. Це означає, що розробники мають доступ до нових функцій та можливостей, що дозволяє їм підтримувати високий рівень актуальності своїх програм. Linux, як операційна система, володіє внутрішніми інструментами, спеціально розробленими для керування та оптимізації мережевого середовища. Ці вбудовані засоби надають користувачам та системним адміністраторам ряд потужних інструментів для налаштування, моніторингу та взаємодії з мережевими ресурсами, сприяючи ефективному управлінню мережевими підключеннями. Однією з ключових характеристик операційної системи Linux є наявність інструментів для налаштування мережі. Це включає в себе можливість встановлення IP-адрес, налаштування маршрутизації та керування мережевими інтерфейсами. До того ж, користувачі можуть використовувати ці інструменти для здійснення різноманітних мережевих налаштувань, таких як налаштування DNS-серверів чи визначення параметрів DHCP. Окрім того, операційна система Linux забезпечує інструменти для ефективного моніторингу мережевого трафіку. Це стає особливо важливим в сучасному світі, де ефективне використання мережевих ресурсів є ключовим аспектом для багатьох організацій та користувачів. Інструменти для моніторингу

трафіку дозволяють вчасно виявляти проблеми та оптимізувати роботу мережі. Окрім цього, Linux надає інструменти для налагодження мережевих підключень. Це означає можливість діагностики та виправлення проблем, пов'язаних з мережевими підключеннями. Такі інструменти можуть виявляти проблеми зі з'єднанням, тестувати швидкість передачі даних та виконувати інші операції, спрямовані на покращення надійності та швидкості мережевого підключення.

Linux виявляє вражаючий рівень підтримки для різноманітних технологій віртуалізації, що відкриває безмежні можливості для оптимізації та управління обчислювальними ресурсами. Однією з важливих технологій в цьому контексті є KVM (Kernel-based Virtual Machine), яка є вбудованою в ядро операційної системи і надає низькорівневий доступ до віртуалізації. KVM є потужним інструментом для віртуалізації, який дозволяє користувачам створювати та управляти віртуальними машинами без значних накладних витрат. Використовуючи цю технологію, можна запускати віртуальні операційні системи на фізичному сервері, що вирішує задачі розділення ресурсів та ізоляції відокремлених середовищ. Окрім KVM, Linux підтримує технологію віртуалізації на рівні операційної системи з використанням контейнерів, таких як Docker. Контейнеризація надає легкість та ефективність у використанні ресурсів, приводячи до більш швидкого впровадження та запуску програм в порівнянні з традиційною віртуалізацією. Контейнери дозволяють ізолювати додатки та їх залежності, забезпечуючи консистентність у різних середовищах та полегшуючи процес розгортання та управління програмами. Віртуалізація в Linux також допомагає розв'язувати проблеми, пов'язані з тестуванням, розгортанням та управлінням програмним забезпеченням. Вона спрощує процеси розробки та забезпечує ефективність у використанні ресурсів. Більше того, віртуалізація може слугувати економічно ефективним рішенням для оптимізації апаратного забезпечення, оскільки дозволяє більш ефективно використовувати наявні обчислювальні ресурси. Надто важливо відзначити, що вбудовані можливості віртуалізації в Linux знаходять широке застосування у сферах великих даних, хмарних обчислень, мережевих рішень та

багатьох інших областях. Це підкреслює універсальність та важливість віртуалізаційних технологій в операційній системі Linux.

Безпека в операційній системі Linux є однією з визначальних її характеристик, і вона досягається завдяки активній участі та співпраці громадськості, регулярним оновленням та гнучкій системі налаштування прав доступу. Ця безпекова архітектура створює надійну оболонку для операційної системи, забезпечуючи захист від різноманітних загроз та небезпек. Велика роль у забезпеченні високого рівня безпеки Linux належить активній спільноті користувачів та розробників. Колективний підхід до виявлення та виправлення потенційних ураз дозволяє швидко реагувати на нові загрози та забезпечує широкий спектр експертізи для вирішення проблем безпеки. Спільнота забезпечує також постійний моніторинг безпекових аспектів системи та вдосконалення її архітектури для протистояння сучасним викликам у сфері кібербезпеки. Регулярні оновлення є ключовим елементом у забезпеченні безпеки Linux. Виробники постійно виправляють виявлені урази та уразливості, що забезпечує користувачам актуальні та безпечні версії операційної системи. Система оновлень також дозволяє швидко реагувати на нові види загроз та ефективно усувати їхні наслідки. Одним з важливих аспектів безпеки Linux є можливість налаштовування прав доступу та обмежень. Користувачі мають можливість точно визначити, хто та як може використовувати різні ресурси системи. Ця гнучкість дозволяє адміністраторам систем точно налаштовувати безпекові політики, щоб вони відповідали конкретним потребам та вимогам. Окрім того, Linux підтримує такі технології, як SELinux (Security-Enhanced Linux), які надають додаткові рівні безпеки шляхом розширення контролю доступу та обмеження привілеїв користувачів.

Важливо також врахувати, що безпека Linux є стійкою до вірусів та шкідливого програмного забезпечення, завдяки своїй архітектурі та стійкості до багатьох типів атак. Це робить Linux популярним вибором для сфер, де безпека є критичним аспектом, таким як серверні системи та мережеві пристрої.

Новаторські зміни та вдосконалення стали важливою частиною еволюції операційної системи Linux. Ці різноманітні компоненти роблять її потужною, гнучкою та придатною для задоволення різних потреб користувачів. Відкритий вихідний код є ключовим фактором, що дозволяє глобальній спільноті активно співпрацювати над розвитком і вдосконаленням системи, забезпечуючи її постійну інноваційність та актуальність. Багато осіб розглядають питання про доцільність використання Linux, враховуючи той факт, що більшість комп’ютерів, ноутбуків і серверів постачаються з іншими операційними системами, які виглядають як задовільні. Однак, важливо розглянути, що Linux не просто конкурує за установлені позиції, але представляє собою альтернативу, що пропонує численні переваги та інновації. Однією з ключових переваг є гнучкість та адаптивність Linux. Здатність користувачів налаштовувати та модифікувати систему відповідно до своїх унікальних потреб стає важливим аспектом. Крім того, система відзначається високою стійкістю до вірусів та зловмисного програмного забезпечення, забезпечуючи надійний захист. Важливо врахувати і глобальний характер спільноти, яка працює над розвитком Linux. Відкритий вихідний код створює можливість для тисяч розробників та ентузіастів спільно внести свій внесок, що призводить до швидкої інноваційної реакції на виклики та потреби сучасного світу. Для відповіді на це запитання важливо розглянути переваги, які пропонує Linux в порівнянні з іншими операційними системами. При зіткненні з проблемами, такими як віруси, шкідливе програмне забезпечення, низька продуктивність, системні збої або високі витрати на обслуговування та ліцензії, Linux виокремлюється як одна з найбільш надійних комп’ютерних екосистем на планеті, і його популярність стабільно зростає. Однією з ключових переваг Linux є його безкоштовність, що дозволяє встановлювати цю операційну систему на будь-яку кількість комп'ютерів без додаткових витрат на програмне забезпечення чи ліцензії. Це робить Linux економічно вигідним вибором, особливо для великих підприємств або організацій, що мають великі парки комп'ютерів. Однак важливо також відзначити, що Linux вирізняється високою стійкістю до вірусів і

шкідливого програмного забезпечення. Ця операційна система має солідний рекорд безпеки, що дозволяє уникнути багатьох загроз, які можуть впливати на інші операційні системи. Крім того, Linux славиться своєю високою продуктивністю та стабільністю. Вона часто використовується на серверах та великих обчислювальних системах завдяки своїй здатності ефективно обробляти завдання в реальному часі та запускати різноманітні застосунки.

Отже, враховуючи всі ці переваги, Linux виявляється не лише надійним рішенням у контексті проблем, але і економічно вигідним та продуктивним вибором для користувачів та організацій. Linux надає стабільну та безпечну платформу, що залишається стійкою до впливу вірусів та шкідливого програмного забезпечення. Більше того, користувачам доступний широкий вибір дистрибутивів, які відповідають різним потребам та вимогам користувачів. Linux надає можливість користувачам збільшити продуктивність завдяки різноманітним оптимізаційним і настроювальним можливостям. Відповідно до унікальних потреб користувачів, вони можуть використовувати Linux для різних завдань, починаючи від серверних платформ і закінчуючи інтернет-пристроями та вбудованими системами. Крім того, Linux відкриває двері для різноманітних можливостей використання, надаючи платформу, яка адаптується до різних сценаріїв. Це означає, що система може бути ефективно використана від розгалужених серверних конфігурацій до енергоефективних інтернет-пристроїв та інтегрованих систем, демонструючи гнучкість та універсальність. Використання Лінукс може сприяти покращенню продуктивності, забезпечити високий рівень безпеки та значно зменшити витрати на програмне забезпечення та обслуговування.

Давайте порівняємо вартість сервера з операційною системою Linux і Windows Server 2016. Вартість ліцензії на Windows Server 2016 Standard складає приблизно 900 доларів США, що представляє собою лише витрати на саму операційну систему. До цього слід додати додаткові витрати на ліцензії користувачів (CAL) та інше програмне забезпечення, таке як бази даних,

веб-сервери, поштові сервери і інше. Наприклад, CAL для одного користувача Windows Server 2016 оцінюється приблизно в 40 доларів США. Якщо вам потрібно ліцензувати 10 користувачів, це призведе до додаткових витрат у розмірі 400 доларів США.

Використання сервера з операційною системою Linux усуває всі ці витрати, оскільки Linux є безкоштовним та має відкритий вихідний код. Встановлення повноцінного веб-сервера (включаючи сервер баз даних) на Linux зводиться лише до кількох простих клацань або введення команд.

Крім того, що Linux має нульову вартість, він виявляється значно менш вразливим до атак і вірусів порівняно з Windows Server, надаючи вищий рівень стабільності та надійності в роботі. Зазвичай, перезавантаження сервера Linux необхідні лише під час оновлення ядра, а сам сервер може працювати безперервно протягом років.

Застосування регулярних оновлень важливе для забезпечення стабільної та безпечної роботи сервера з Linux. Всі ці переваги роблять Linux привабливим вибором для тих, хто шукає економічно ефективну та надійну операційну систему для своїх серверів.

2.7 Базова архітектура Unix систем

Ядро - це центральна частина операційної системи, відповідальна за управління ресурсами та надання середовища для виконання програм. У ньому зосереджені ключові функції, необхідні для взаємодії між апаратним і програмним забезпеченням.

Розглянемо функції ядра. Перш за все, ядро відповідає за управління процесами. Воно визначає порядок виконання завдань, розподіляє ресурси та визначає пріоритети виконання програм. Крім того, ядро взаємодіє з апаратним

забезпеченням, забезпечуючи доступ до пристроїв та реалізуючи обробку переривань.

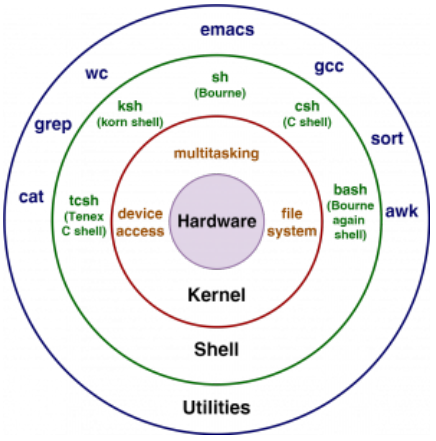


Рисунок 2.5 - Схематичне зображення рівнів ядра

Ядро може бути розширене за допомогою модулів, які можна завантажувати або вивантажувати в робочий процес ядра за запитом. Це робить систему більш гнучкою та адаптованою до конкретних потреб. Конфігурація ядра визначає, які функції включені або виключені, що дозволяє вам налаштовувати систему під свої завдання.

Ядро відповідає за керування процесами в системі. Воно створює, видаляє та призначає пріоритети процесам. Планування виконання процесів є важливою функцією ядра, оскільки воно визначає, який процес отримає доступ до ЦП у конкретний момент часу.

Розуміння роботи ядра дозволить вам оптимізувати роботу системи, ефективно використовувати ресурси та забезпечити стабільну та продуктивну роботу вашої робочої станції чи сервера.

Системні виклики є ключовою складовою операційних систем, забезпечуючи взаємодію між програмами та ядром. Це механізм, який програми використовують для звертання до операційної системи з метою отримання доступу до різних ресурсів чи послуг. Вони є важливим засобом взаємодії для

програмістів, дозволяючи їм використовувати функціонал операційної системи для виконання своїх завдань.

Кожен системний виклик відповідає конкретній операції, яку програма може викликати. Наприклад, системний виклик для вводу-виводу може бути використаний для читання чи запису даних в файл, або для взаємодії з пристроями вводу-виводу. Інші системні виклики можуть включати створення нового процесу, керування пам'яттю, аутентифікацію та багато інших операцій.

Системні виклики є містком між користувацьким простором та ядром операційної системи. Коли програма викликає системний виклик, виклик виконується в просторі ядра, що забезпечує безпеку та управління ресурсами. Системні виклики реалізовані через спеціальні інструкції процесора, які здійснюють перехід з режиму користувача в режим ядра для виконання необхідної операції.

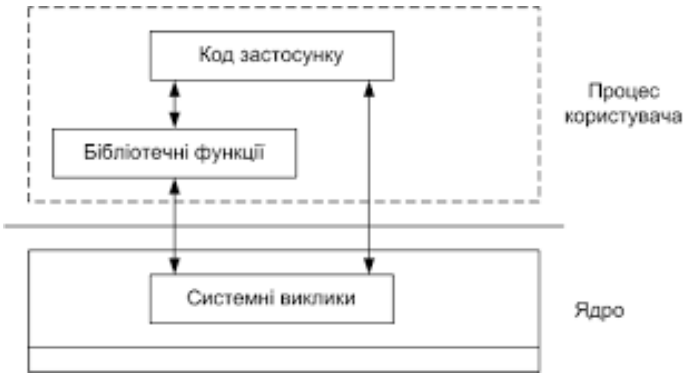


Рисунок 2.6 – Схематичне відображення взаємодії викликів

Кожна операційна система має свій набір системних викликів, але деякі з них є стандартними та є частиною інтерфейсу POSIX. Це дозволяє розробникам створювати переносимий код, який може працювати на різних системах, дотримуючись стандартів. Системні виклики стають основою для розробки програм та додатків, забезпечуючи їм доступ до функціоналу операційної системи.

Крім того, системні виклики відіграють важливу роль у безпеці та обробці помилок. Операційні системи встановлюють права доступу до різних системних

викликів, контролюючи тим самим, які дії можуть виконувати користувачі та програми. Це сприяє забезпеченню конфіденційності та цілісності даних, а також управлінню ресурсами системи.

Важливим аспектом є інтерфейс між мовами програмування та системними викликами. Деякі мови, такі як C, надають декларації для системних викликів, що дозволяє програмістам взаємодіяти з операційною системою на більш високому рівні абстракції, спрощуючи роботу з операційними системами та забезпечуючи більш високий рівень переносимості для програм.

Файлова система Linux є однією з фундаментальних частин операційної системи, визначаючи структуру та організацію збереження даних на дисках. Вона грає ключову роль у взаємодії між ядром операційної системи та користувачем, забезпечуючи управління файлами та каталогами.

Файлова система Linux організована у вигляді дерева каталогів, де кожен вузол може бути або файлом, або каталогом. Корінь цього дерева відомий як кореневий каталог, позначений символом "/". Каталоги вміщують файли та інші каталоги, утворюючи ієрархію, яка полегшує організацію та доступ до даних.

Властивості файлів та каталогів Linux такі, де кожен файл або каталог має ім'я, що служить унікальним ідентифікатором всередині свого каталогу. Однак важливо враховувати, що Linux враховує регістр при наданні імен файлам, тобто "File.txt" і "file.txt" розглядаються як різні файли.

Кожен файл також має власника та групу, які визначають права доступу до цього файлу. Права доступу включають можливість читання, запису та виконання, і визначають, які дії можна виконати щодо файла.

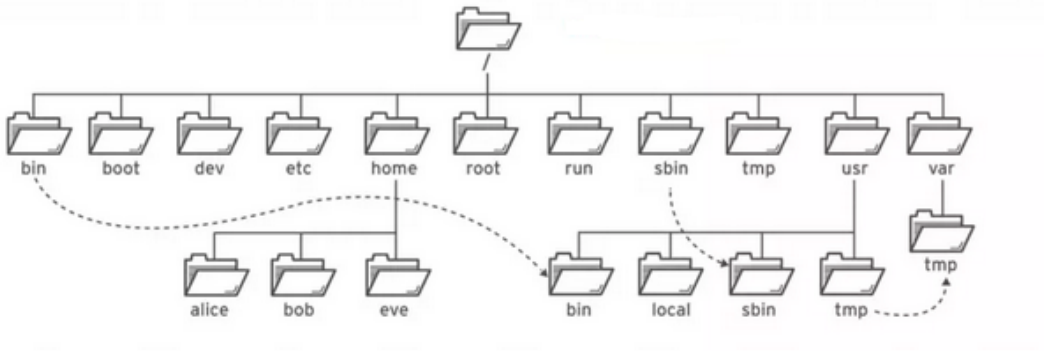


Рисунок 2.7 - Ієрархія файлової системи Linux

Linux розрізняє різні типи файлів. Зокрема, є звичайні файли, каталоги, символьні та блочні пристрої, сокети та FIFO (іменовані канали). Кожен з цих типів відіграє свою роль у системі. Наприклад, каталоги використовуються для організації ієрархії файлів, а символьні та блочні пристрої представляють собою інтерфейси для звернення до апаратного забезпечення.

Linux підтримує різні типи файлових систем, такі як ext4, xfs, btrfs та інші. Кожна з цих файлових систем має свої особливості та переваги. Файлові системи можна монтувати у різні точки в ієрархії файлової системи, що дозволяє користувачам та системі працювати з різними пристроями чи розділами, як однією злагодженою системою.

Linux надає ряд інструментів та команд для роботи з файловою системою. Команди, такі як ls для перегляду вмісту каталогу, cp для копіювання файлів, mv для переміщення файлів та rm для видалення файлів, забезпечують базовий функціонал для роботи з файлами та каталогами.

Linux має логічне та фізичне подання файлової системи. Файлова система має логічне та фізичне подання. Логічне подання визначає, як файли та каталоги виглядають для користувача та програм. Фізичне подання вказує, які структури даних використовуються для збереження цих файлів та каталогів на диску. Розуміння цих аспектів допомагає оптимізувати ефективність та використання простору на диску.

Однією з функцій також є журналювання та відновлення. Багато файлових систем Linux підтримують журналювання, що забезпечує додатковий рівень стійкості та відновлення в разі відмови. Журнал зберігає записи про всі зміни, які вносяться в файлову систему, дозволяючи системі відновитися після непередбачуваних ситуацій, таких як відключення живлення чи аварія.

Синхронізація та доступ до файлів, де Linux використовує механізми блокування та синхронізації для забезпечення правильного доступу до файлів у многозадачних середовищах. Це важливо для уникнення конфліктів та забезпечення консистентності даних.

Linux також підтримує спеціальні файлові системи, такі як procfs та sysfs, які надають інформацію про систему та конфігурацію ядра через інтерфейс каталогів та файлів. Це дозволяє програмам та системним інструментам отримувати доступ до важливих параметрів та статистики ядра.

Процеси та керування пам'яттю є важливими аспектами операційної системи Linux, визначаючи спосіб виконання програм та управління доступом до пам'яті. Розуміння цих концепцій є ключовим для оптимізації використання ресурсів та забезпечення ефективності роботи системи.

Процес в Linux - це програма у виконанні, яка має свій власний адресний простір пам'яті, ідентифікатор процесу (PID), контекст виконання та набір ресурсів. Операційна система Linux підтримує багатозадачність, що означає, що одночасно може виконуватися багато процесів. Процеси можуть взаємодіяти один з одним через міжпроцесовий обмін, але кожен має свій власний простір пам'яті та робочий контекст.

Щоб створити новий процес, Linux використовує системний виклик fork(). В результаті цього виклику створюється копія батьківського процесу, включаючи код, дані та стан виконання. Після цього може бути використаний системний виклик exec() для заміни коду нового процесу кодом іншої програми. Процес може бути призупинений, відновлений або припинений за допомогою сигналів.

Процеси мають свій життєвий цикл, який включає створення, виконання, блокування, відновлення та завершення. Системні виклики та розкладання процесів забезпечують управління цим циклом. Процес може бути в пасивному стані, очікуючи подій, або активно виконувати інструкції в своєму коді.

Кожен процес має власний адресний простір пам'яті, який поділяється на сегменти, такі як код, дані та стек. Операційна система відповідає за управління цим адресним простором та вирішення конфліктів доступу. Кожен процес має також власні дескриптори файлів, таблицю процесів та інші ресурси, які визначають його стан та поведінку.

Керування пам'яттю в Linux включає управління фізичною та віртуальною пам'яттю. Віртуальна пам'ять дозволяє процесам використовувати більше пам'яті, ніж є фактично доступно на фізичному диску. Це досягається за допомогою механізму обміну даними між RAM та диском.

Linux використовує механізми керування віртуальною пам'яттю, такі як сегментація та стронування. Сегментація дозволяє розділити віртуальний адресний простір на логічні сегменти, такі як код, дані та стек. Стронування розбиває фізичну та віртуальну пам'ять на сторінки, що полегшує роботу з великими обсягами даних та ефективно використовує пам'ять.

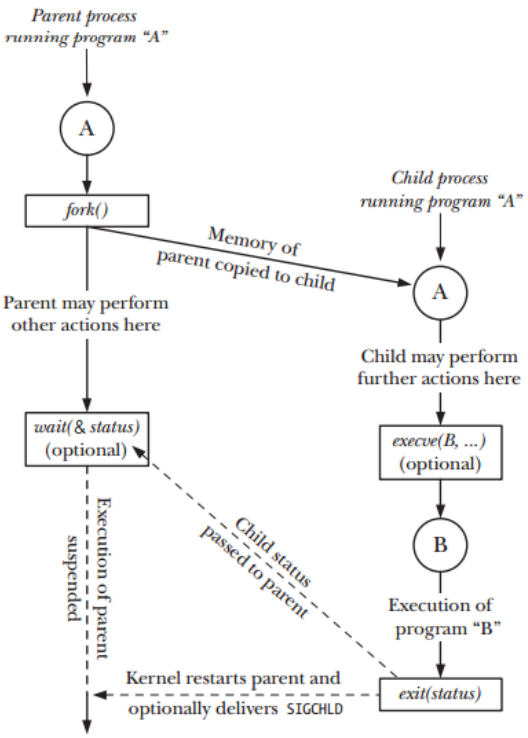


Рисунок 2.8 – Взаємодія процесів в Linux

Системні виклики, такі як brk() та mmap(), дозволяють процесам динамічно адаптувати розмір свого віртуального адресного простору, забезпечуючи ефективне використання пам'яті. Системні виклики fork() та exec() також впливають на управління пам'яттю, оскільки вони створюють нові адресні простори для нових процесів.

Linux дозволяє спільне використання пам'яті між процесами за допомогою механізмів, таких як розділені сегменти та мапування пам'яті. Це може бути використано для обміну даними між процесами чи для спільного доступу до областей пам'яті.

Для оптимізації використання пам'яті Linux надає інструменти, такі як утиліта top або htop, які дозволяють моніторити використання пам'яті процесами. Програмісти можуть використовувати інструменти, такі як Valgrind, для виявлення помилок в роботі з пам'яттю, таких як витікання пам'яті чи неправильне використання.

Системи Linux також враховують аспекти безпеки та обмеження ресурсів для процесів. Обмеження ресурсів дозволяє управляти доступом до CPU, пам'яті та інших системних ресурсів. Дозволяючи адміністраторам контролювати та обмежувати використання ресурсів, Linux забезпечує стабільність та безпеку системи.

Введення та виведення (I/O) в операційній системі Linux грає ключову роль у взаємодії користувача та програм з системою. Це обов'язковий елемент для виконання завдань, забезпечення взаємодії з різними пристроями та забезпечення ефективної обробки даних.

Ввод-вивід у Linux реалізується через файлові дескриптори та системні виклики. Кожен процес має свою таблицю файлових дескрипторів, що відображає відкриті файли, сокети, пристрої та інші ресурси. Системні виклики, такі як read(), write(), open(), та close(), надають інтерфейс для взаємодії з цими ресурсами. Наприклад, read() використовується для читання даних з файлового дескриптора, а write() для запису даних.

У кожного процесу в Linux є три стандартні потоки введення/виведення: STDIN (стандартний вхід), STDOUT (стандартний вихід) та STDERR (стандартна помилка). Вони зазвичай пов'язані з клавіатурою, екраном та потоком помилок, але можуть бути перенаправлені на інші пристрої чи файли за допомогою командного рядка.

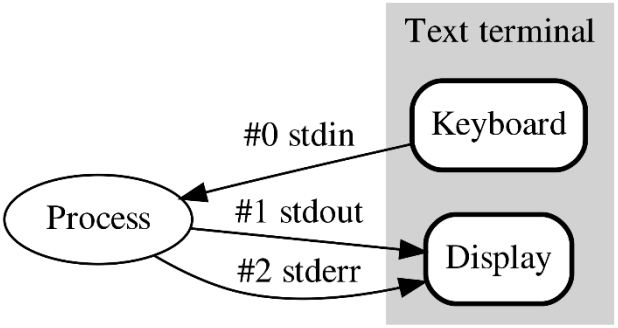


Рисунок 2.9 – Стандартні потоки

Операційна система Linux надає широкі можливості роботи з файлами та каталогами. Команди, такі як ls для перегляду вмісту каталогу, cp для копіювання файлів, mv для переміщення файлів та rm для видалення файлів, дозволяють користувачам ефективно керувати файловою системою.

Для зчитування даних з клавіатури використовуються функції з бібліотеки введення-виведення C, такі як scanf() або fgets(). Вони дозволяють програмам чекати на введення від користувача та обробляти ці дані для подальшого використання. Утиліти командного рядка також можуть отримувати введення з клавіатури під час виконання.

Для виведення даних на екран використовуються функції виведення, такі як printf() та puts() в мові програмування C. Ці функції дозволяють форматувати та виводити рядки, числа та інші дані на екран.

Linux підтримує асинхронне введення/виведення через системні виклики, такі як select(), poll(), або epoll(). Це дозволяє програмам працювати з декількома джерелами введення/виведення одночасно без блокування виконання інших завдань.

Linux підтримує роботу з мережевими пристроями та сокетами для взаємодії з мережею. Системні виклики, такі як socket(), bind(), connect(), send(), та recv(), дозволяють створювати, конфігурувати та взаємодіяти з мережевими з'єднаннями.

У командному рядку (шелі) введення/виведення грає важливу роль. Перенаправлення виведення та введення, конвеєри (pipes), та команди обробки тексту (awk, sed) дозволяють зручно обробляти дані та управляти виведенням програм.

Введення/виведення може бути блокуючим чи неблокуючим. У блокуючому режимі програма чекає завершення операції вводу/виводу, що може спричинити затримки. У неблокуючому режимі програма продовжує виконання інших завдань, не чекаючи завершення введення/виведення.

Управління потоками введення/виведення та буферизація грають важливу роль в оптимізації продуктивності. Буферизація дозволяє системі зберігати дані в

буферах та відправляти їх партіями, що зменшує кількість системних викликів та поліпшує ефективність.

Безпека введення/виведення важлива у веб-додатках та системах обробки даних. Використання безпечних функцій введення, обмеження прав доступу до файлів та правильне управління ресурсами допомагають уникнути атак, таких як переповнення буфера чи зловживання введенням.

Linux надає інструменти для моніторингу та налагодження введення/виведення. Утиліти, такі як strace або lsof, дозволяють відстежувати системні виклики та роботу з файловими дескрипторами. Це допомагає виявити проблеми та оптимізувати введення/виведення для покращення продуктивності системи.

Мережеві функції в операційній системі Linux є ключовим компонентом для забезпечення взаємодії різних комп'ютерів та пристроїв в мережі. Вони включають в себе широкий спектр можливостей, від створення з'єднань до обміну даними та управління мережевими ресурсами.

Одним із ключових аспектів мережевих функцій є використання сокетів та мережевих портів. Сокети представляють собою абстракцію для взаємодії між процесами через мережу, де кожен сокет пов'язаний із конкретним мережевим портом. Мережеві порти дозволяють ідентифікувати різні служби або процеси на конкретному пристрої в мережі.

Linux підтримує різні мережеві протоколи, такі як TCP/IP, UDP, ICMP і багато інших. Сполучення протоколів утворює стек протоколів, що визначає, як дані передаються та отримуються в мережі. Наприклад, TCP (Transmission Control Protocol) надає забезпечення передачі даних без втрат та упорядкування, в той час як UDP (User Datagram Protocol) передає дані без гарантії доставки та упорядкування.

IP-адресація визначається унікальними числовими ідентифікаторами для пристроїв у мережі. Вона може бути IPv4 або IPv6, залежно від конфігурації

мережі. Підмережі дозволяють розбити велику мережу на менші частини, що полегшує управління та забезпечує ефективне використання IP-адрес.

Linux надає системні виклики для роботи з мережею. socket() використовується для створення нового сокета, bind() для прив'язки сокета до конкретної IP-адреси та порта, listen() для прийому вхідних з'єднань, а accept() для прийняття нового з'єднання. Інші виклики, такі як send(), recv(), connect(), та close(), використовуються для передачі даних через сокети.

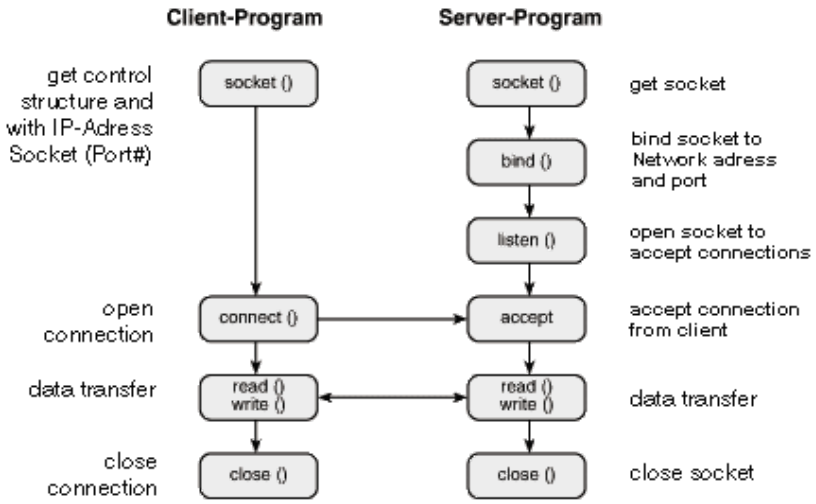


Рисунок 2.10 – Приклад взаємодії мережевих сокетів на клієт-серверній архітектурі

Програмування сокетів в Linux використовує мови програмування, такі як C або Python. Програми можуть створювати сервери, які чекають на з'єднання, або клієнти, які підключаються до існуючих серверів. Сокети дозволяють обмінюватися даними між різними пристроями та програмами через мережу.

Linux надає ряд утиліт та команд для роботи з мережею. Наприклад, ifconfig дозволяє налаштовувати інтерфейси мережі, ping використовується для перевірки доступності пристроїв у мережі, а traceroute дозволяє відстежувати шлях до певного пристрою через мережу.

Linux підтримує різні мережеві сервіси та демони, які працюють на фоні та надають різноманітні функції. Наприклад, dhcpd відповідає за надання IP-адрес

автоматично, sshd дозволяє віддалене підключення до системи через протокол SSH.

Firewall в Linux дозволяє контролювати доступ до мережі, фільтруючи та блокуючи певний мережевий трафік. Утиліти, такі як iptables чи ufw, дозволяють адміністраторам конфігурувати правила для захисту системи від несанкціонованого доступу та зловживання.

Мережеві протоколи, такі як DNS (Domain Name System), грають ключову роль у визначенні IP-адрес для доменних імен. Linux використовує конфігураційні файли та утиліти, такі як /etc/resolv.conf та nslookup, для налаштування та тестування роботи DNS.

Віртуалізація в Linux дозволяє створювати віртуальні мережеві інтерфейси та мости. Мережеві мости дозволяють об'єднувати різні мережі та пересилати пакети між ними, що важливо для конфігурацій віртуальних мереж.

Linux підтримує різні мережеві програми та сервіси, які можна використовувати для різних цілей. Наприклад, Apache є популярним веб-сервером, Samba дозволяє обмінюватися файлами між Linux та Windows системами, а Nginx є альтернативним веб-сервером.

Мережеві функції є важливим елементом для розробки різноманітних додатків, таких як веб-додатки, віддалені сервіси та інші. Розуміння роботи мережі та використання мережевих функцій допомагає розробникам створювати ефективні та стабільні додатки.

3 ВПРОВАДЖЕННЯ МОНІТОРИНГУ СЕРВЕРА OPENVPN ЧЕРЕЗ ІНТЕГРАЦІЮ ІЗ СИСТЕМОЮ PROMETHEUS ТА PYTHON

3.1 Встановлення пакетів та налаштування

В даній кваліфікаційній роботі буде використано два сервера на операційній системі Debian. На одному буде знаходитись сервер моніторингу Prometheus, а на іншому OpenVPN сервер.

Спочатку налаштуємо OpenVPN сервер:

Оновимо дані з репозиторіїв за допомогою команди та перевіримо чи є необхідні нам пакети:

apt update && apt-cache search openvpn

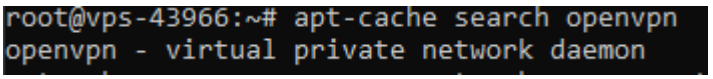


Рисунок 3.1 – Пошук пакета openvpn в репозиторії

Далі встановимо необхідні для нас пакети:

apt install -y openvpn prometheus-node-exporter iptables iptables-persistent

Пакет openvpn включає в себе виконуючі файли сервера та клієнта, котрі нам необхідні для виконання роботи.

Пакет prometheus-node-exporter включає в себе виконуючі файли експортера, за допомогою нього ми зможемо моніторити такі дані із сервера як оперативна пам’ять, завантаженність центрального процесора, кількість вільного місця на диску та інша корисна інформація.

Пакет iptables являє собою міжмережевий екран, который буде використаний для забезпечення базової мережевої гігієни сервера. В свою чергу iptables-persistent допомагає зберігати правила міжмережевого екрана.

Створимо нову папку для директорії OpenVPN сервера та перейдемо в неї:

mkdir -p /etc/openvpn/others && cd /etc/openvpn/others

Дана папка буде слугувати для збереження файлів сертифікатів та bash скрипта на openssl, который сертифікати генерує та підписує, а також може створювати файли конфігурації OpenVPN. Код bash скрипта можна знайти в Додаток А. Згенеруємо новий рутовий сертифікат, щоб підписати ним інші та створимо сертифікат для OpenVPN сервера.

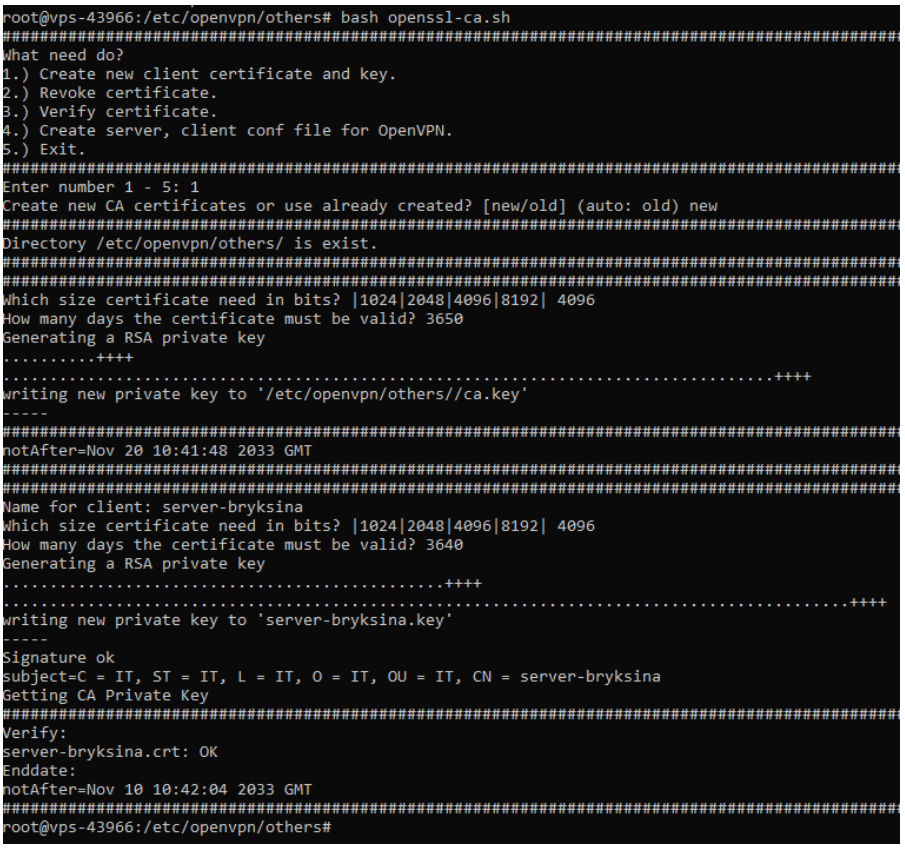


Рисунок 3.2 – Вивід bash скрипта при генерації перших сертифікатів

Таким же чином створимо ще один сертифікат для підключення сервера Prometheus.

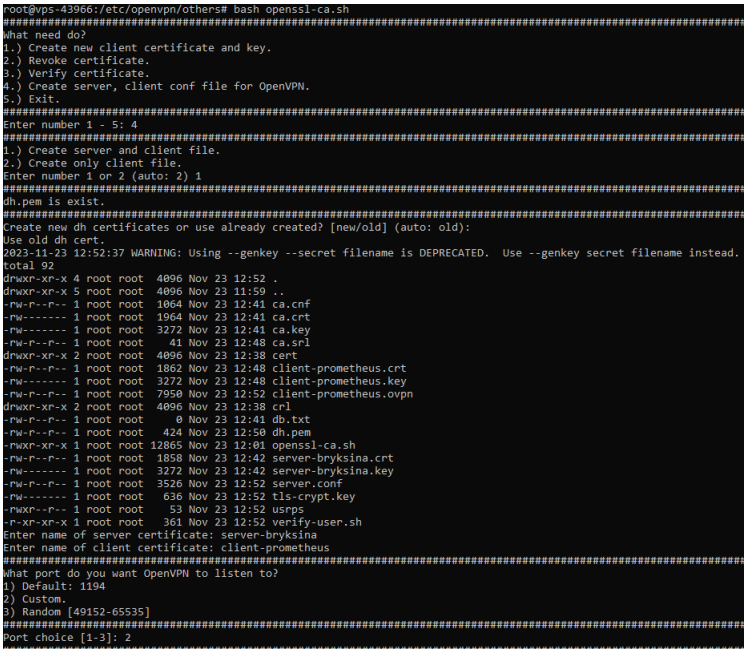


Рисунок 3.3 – Генерація сертифіката сервера Prometheus

Після генерації сертифікату, скрипт запропонує створити файл конфігурації сервера на що ми погоджуємось.

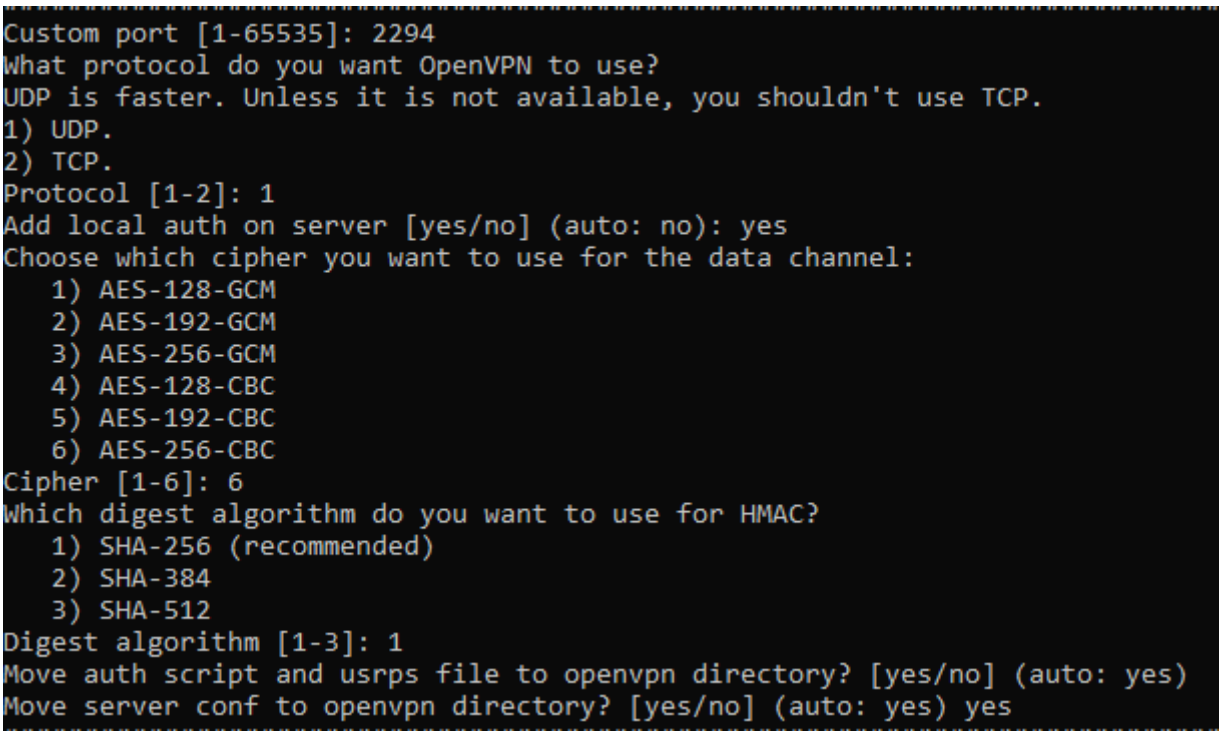


Рисунок 3.4 – Вибір алгоритмів та чіпсета для OpenVPN сервера

Після цього ми отримаємо файли конфігурації сервера та клієнта для цього сервера.

```
root@vps-43966:/etc/openvpn# cat server.conf
port 2294
proto udp
dev tun
user nobody
group nogroup
duplicate-cn
persist-key
persist-tun
keepalive 10 120
topology subnet
server 10.8.0.0 255.255.255.0
status /var/log/openvpn/status.log
verb 3
push "dhcp-option DNS 1.1.1.1"
push "dhcp-option DNS 8.8.8.8"
push "redirect-gateway def1 bypass-dhcp"
script-security 2
auth-user-pass-verify /etc/openvpn/verify-user.sh via-file
username-as-common-name
auth SHA256
cipher AES-256-CBC
tls-server
```

Рисунок 3.5 – Приклад базової конфігурації сервера

Перейдемо до налаштування експортера на python. Для його роботи необхідний python не нижче версії 3.9. Так як в сбірку більшості дистрибутивів Linux python входить перевіримо, яка версія встановлена на сервері: python3 --version

```
root@vps-43966:/etc/openvpn# python3 --version
Python 3.9.2
```

Рисунок 3.6 – Версія python

Після цього встановлюємо необхідні для роботи коду модулі:

pip3 install flask flask_httpauth prometheus_client

Далі для зручності керування цим експортером створюємо unit-файл для systemctl. Він має виглядати наступним чином:

```
root@vps-43966:~# cat /etc/systemd/system/py-exporter.service
[Unit]
Description=Custom python script for create metrics
Before=network-online.target
Wants=network-online.target
[Service]
Type=simple
ExecStart=/etc/py-exporter/py-exporter start
ExecStop=/etc/py-exporter/py-exporter stop
[Install]
WantedBy=multi-user.target
root@vps-43966:~#
```

Рисунок 3.7 – Unit-файл py-exporter

Після цього створимо директорію для py-exporter /etc/py-exporter/ та додамо в неї виконуючий файл з котрим можна ознайомитись в Додаток Б. Константи для даного серверу виглядають наступним чином:

```
exporter_port = 9101
exporter_ip = "185.25.116.46"
exporter_login = "py-exporter"
exporter_password = "DC6xFm3bat926mnp9J82KnMHA6j6MCa9"
exporter_certificate_ssl = "/etc/py-exporter/crt/exporter.crt"
exporter_key_ssl = "/etc/py-exporter/crt/exporter.key"
check_ca_paths = [exporter_certificate_ssl, "/etc/openvpn/others/ca.crt", "/etc/openvpn/others/server-bryksina.crt", "/etc/openvpn/others/client-prometheus.crt"]
```

Рисунок 3.8 – Константи py-exporter

Далі створимо ще одну папку в директорії /etc/py-exporter/crt та згенеруємо ще один сертифікат за допомогою bash скрипта і перемістимо його в цю папку.

На даному сервері OpenVPN також було реалізована аутентифікація по логіну і паролю за допомогою bash скрипів. Приклад скрипта аутентифікації verify-user.sh можна знайти в Додаток А. Даний скрипт парсить файл з логіном і паролем та повертає ACCEPT або REJECT в залежності від даних, котрі отримує.

```
root@vps-43966:/etc/openvpn# cat usrps
bryksina:+GixnJeAeQly5cfnAAywb2DiR/bLvrQ0+eHVdcX5FfU=
```

Рисунок 3.9 – Файл с користувацькими даними

Для коректної роботи маршрутизації клієнтів також необхідно внести зміни в конфігурацію ядра: echo ‘net.ipv4.ip_forward=1’ >> /etc/sysctl.conf && sysctl -p
Та додати правила до мережевого екрану, котрі дозволяють підключення до даного сервера:

iptables -A INPUT -p udp -m udp –dport 2294 -m comment –comment “openvpn-server”-j ACCEPT

iptables -A INPUT -s 5.61.32.169/32 -p tcp -m tcp –dport 9101 -m comment –comment “py-exporter” -j ACCEPT

Збережемо правила: netfilter-persistent save

На цьому процес налаштування OpenVPN сервера завершено – запускаємо служби:systemctl start openvpn@server prometheus-node-exporter py-exporter && systemctl enable openvpn@server py-exporter

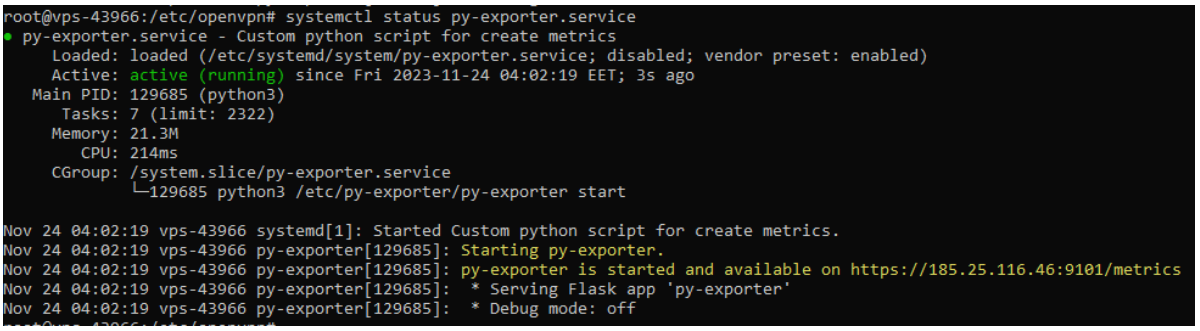


Рисунок 3.10 – Успішний запуск py-exporter

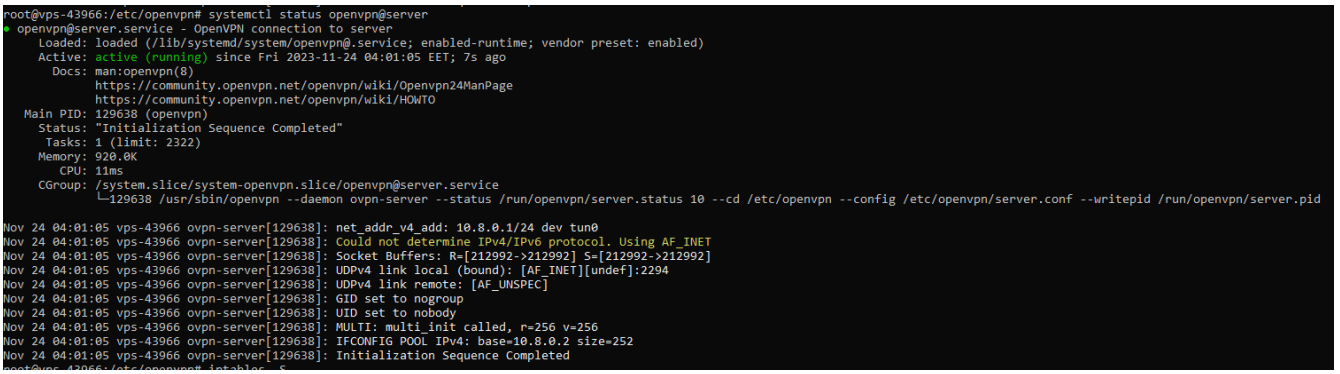


Рисунок 3.11 – Успішний запуск OpenVPN сервера

Далі буде приведено приклад налаштування серверу з моніторингом.

Виконаємо наступні команди: apt update && apt-cache search prometheus

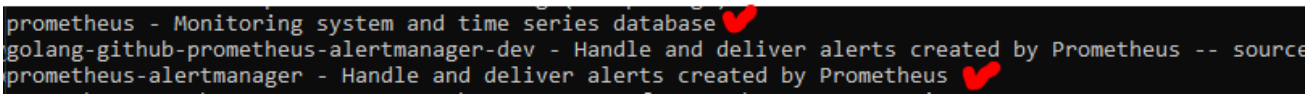


Рисунок 3.12 – Необхідні пакети prometheus в репозиторії

Встановимо необхідні пакети: apt install -y prometheus prometheus-alertmanager prometheus-node-exporter apt-transport-https software-properties-common wget iptables iptables-persistent

Для встановлення сервера grafana необхідно додати репозиторій до свого source файлу. Даний репозиторій працює по ключу і тому для нього необхідно провести деякі маніпуляції:

mkdir -p /etc/apt/keyrings/

wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | tee etc/apt/keyrings/grafana.gpg > /dev/null

echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | tee -a /etc/apt/sources.list.d/grafana.list && apt update

Після даних дій в репозиторії повинен з’явитись необхідний пакет.

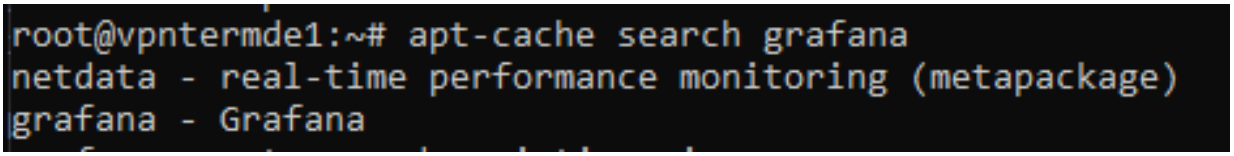


Рисунок 3.13 – Необхідні пакети grafana в репозиторії

Встановлюємо його: apt install grafana -y

Аналогічним способом налаштовуємо py-exporter, як і на OpenVPN сервері, тільки код тепер із додатка В.

Для реалізації моніторингу можливості підключення до сервера OpenVPN створимо папку configs в директорії py-exporter та додамо туди три файли з наступним змістом:

1. Файл constants – необхідний для повернення констант сервера з яким не вдалось створити з’єднання:

```
{
   "client-prometheus": {
      'server': "bryksina",
      'ip': "185.25.116.46",
```

```
        'port': "2294"
    }
}
```

2. Файл credential в ньому зберігаються користувацькі дані для підключення:

bryksina

+GixnJeAeQly5cfnAAywb2DiR/bLvrQ0+eHVdcX5FfU=

3. Файл конфігурації OpenVPN сервера до якого ми будемо підключатись:

```
root@vpntermde1:~# cat /etc/py-exporter/configs/client-prometheus.ovpn
client
proto udp
remote 185.25.116.46 2294
dev tun
nobind
persist-key
persist-tun
topology subnet
auth-nocache
auth-user-pass
verb 4
cipher AES-256-CBC
```

Рисунок 3.14 – Файл конфігурації клієнта OpenVPN сервера

Константи py-exporter на даному сервері виглядають наступним чином:

```
exporter_port = 9101
exporter_ip = "5.61.32.169"
exporter_login = "py-exporter"
exporter_password = "DC6xFm3bat926mnp9J82KnMHA6j6MCa9"
exporter_certificate_ssl = "/etc/py-exporter/crt/exporter.crt"
exporter_key_ssl = "/etc/py-exporter/crt/exporter.key"
check_ca_paths = [exporter_certificate_ssl]

openvpn_config_file = ["/etc/py-exporter/configs/client-prometheus.ovpn"]
openvpn_credential_file = "/etc/py-exporter/configs/credential"
openvpn_constant_file = "/etc/py-exporter/configs/constants"

py_metrics = Gauge('py_check_ssl_ca_ovpn_expiry', 'Custom metric on python.', ['ip', 'instance', 'ca'])
py_metrics_openvpn_connection = Gauge('py_check_ovpn_connection', 'Custom metric on python.',
                                      ['ip', 'instance', 'server_name', 'server_ip', 'server_port'])
```

Рисунок 3.15 – Константи py-exporter на сервері моніторингу

Далі перейдемо до налаштування самого Prometheus. Для цього відредагуємо файл конфігурації Prometheus /etc/prometheus/prometheus.yml наступним чином:

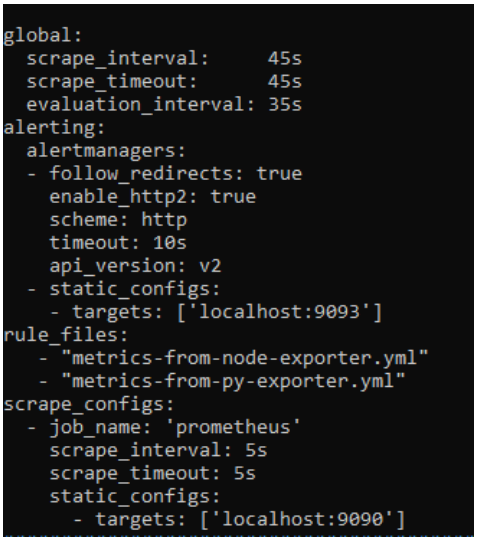


Рисунок 3.16 – Стандартні параметри файлу конфігурації Prometheus

Ця конфігурація Prometheus визначає параметри для збору, оцінки та сповіщення на основі метрик. Давайте прокоментуємо кожен розділ:

scrape_interval: Цей параметр визначає інтервал часу між запусками збору метрик для кожного об'єкта (job). У цьому випадку, метри збираються кожні 45 секунд.

scrape_timeout: Максимальний час, який Prometheus виділений для завершення одного запиту збору метрик. У випадку, якщо збір метрик не завершується за цей час, він буде перервано.

evaluation_interval: Інтервал часу між двома оцінками правил в системі сповіщень. У цьому випадку, оцінка правил відбувається кожні 35 секунд.

alertmanagers: Вказує на конфігурацію Alertmanager, до якого Prometheus відправляє алерти для обробки та надсилання сповіщень.

rule_files: Вказує на шляхи до файлів правил, де визначені правила для генерації алертів.

job_name: Назва робочої групи, яку будуть використовувати конфігурації збору метрик. У цьому випадку, Prometheus буде збирати метрики для робочої групи "prometheus".

scrape_interval та scrape_timeout: Параметри для конфігурації інтервалу та таймауту збору метрик для конкретного робочого групи.

static_configs: Визначає статичний список цілей (targets), з яких буде здійснюватися збір метрик. У цьому випадку, метрики будуть збиратися з локальної адреси localhost:9090, яка відповідає самому Prometheus.

```
- job_name: 'node-exporters'
  scrape_timeout: 60s
  scrape_interval: 80s
  scheme: https
  tls_config:
    insecure_skip_verify: true
  basic_auth:
    username: prometheus
    password: 0O2g6YUktgSBE5Uo5wg80mZ4eBs=
  static_configs:
    - targets:
      - "5.61.32.169:9100@prometheus-server"
      - "185.25.116.46:9100@openvpn-server"
  relabel_configs:
    - source_labels: [ __address__ ]
      regex: '.*@(.*)'
      replacement: $1
      target_label: instance
    - source_labels: [ __address__ ]
      regex: '(.*):.*'
      replacement: $1
      target_label: IP
    - source_labels: [ __address__ ]
      regex: '(.*)@.*'
      replacement: $1
      target_label: __address__
```

Рисунок 3.17 – Параметри для моніторингу node-exporters

```
- job_name: 'py-exporter'
  scrape_timeout: 30s
  scrape_interval: 40s
  scheme: https
  metrics_path: /metrics
  tls_config:
    insecure_skip_verify: true
  basic_auth:
    username: py-exporter
    password: DC6xFm3bat926mnp9J82KnMHA6j6MCa9
  static_configs:
    - targets:
      - "5.61.32.169:9101@prometheus-server"
      - "185.25.116.46:9101@openvpn-server"
  relabel_configs:
    - source_labels: [ __address__ ]
      regex: '.*@(.*)'
      replacement: $1
      target_label: instance
    - source_labels: [ __address__ ]
      regex: '(.*):.*'
      replacement: $1
      target_label: IP
    - source_labels: [ __address__ ]
      regex: '(.*)@.*'
      replacement: $1
      target_label: __address__
```

Рисунок 3.18 – Параметри для моніторингу py-exporters

Переходимо до налаштування правил повідомлень із сервера Prometheus в telegram сам процес створення бота не буде висвітленний. Конфігуруємо файл /etc/prometheus/alertmanager.yml наступним чином:

```
global:
  resolve_timeout: 5m

route:
  group_by: ['alertname', 'job', 'service']
  group_wait: 10s
  group_interval: 5m
  repeat_interval: 3h
  receiver: 'telegram'
  routes:
  - match:
      severity: critical
    receiver: telegram

receivers:
- name: 'telegram'
  telegram_configs:
  - bot_token: '6276741927:AAEK6P5g-sk774BnzrSAphiHCtAVR35[redacted]'
    chat_id: 53560[redacted]
```

Рисунок 3.19 – Файл конфігурації alertmanager

resolve_timeout: 5m: Вказує максимальний час на вирішення (закриття) алертів. Якщо алерт не вирішується протягом цього часу, він вважається невирішеним.

group_by: ['alertname', 'job', 'service']: Групування алертів за іменем, робочою групою та службою.

group_wait: 10s: Час очікування перед тим, як групувати алерти.

group_interval: 5m: Інтервал часу між повторними спробами об'єднання груп алертів.

repeat_interval: 3h: Інтервал повторення для відправки повторних алертів, навіть якщо попередній алерт ще не було вирішено.

receiver: 'telegram': Вказує стандартний отримувач для алертів.

routes: Визначає додаткові правила маршрутизації. У цьому випадку, якщо алерт має severity 'critical', він буде направлений на отримувач 'telegram'.

name: 'telegram': Визначає ім'я отримувача, яке використовується в розділі receiver та routes.

telegram_configs: Вказує на конфігурацію для відправки алертів в Telegram.

bot_token: Токен вашого Telegram-бота, який отримано від BotFather.

chat_id: Ідентифікатор чату, куди будуть відправлятися алерти.

Далі створимо два файли з правилами для alertmanager. В перший /etc/prometheus/metrics-from-node-exporter.yml додамо правила повязані з CPU, RAM та вільним місцем на сервері.

```
root@vpntermde1:/etc/prometheus# cat metrics-from-node-exporter.yml
groups:
- name: node-exporter
  rules:
  - alert: 'Host DOWN'
    expr: up == 0
    for: 90s
    labels:
      severity: critical
    annotations:
      description: "Instance {{ $labels.instance }} with IP:({{ $labels.IP }}) has been down
        for more than 90 seconds."
      summary: Instance {{ $labels.instance }} is DOWN
  - alert: Low RAM
    expr: node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 < 10
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "On instance {{ $labels.instance }} out of RAM memory"
      description: "RAM memory on instance {{ $labels.instance }} with IP:({{ $labels.IP }}) is filling up (< 10% left). Free RAM level in right now
  - alert: openvpn@server.service
    expr: node_systemd_unit_state{name="openvpn@server.service",state="active"} == 0
    for: 1s
    labels:
      severity: critical
    annotations:
      summary: "Instance {{ $labels.instance }} is down"
      description: "{{ $labels.name }} on {{ $labels.instance }} of job {{ $labels.job }} is down."
  - alert: Low Disk Space
    expr: (node_filesystem_avail_bytes * 100) / node_filesystem_size_bytes < 10 and ON (instance, device, mountpoint) node_filesystem_readonly == 0
    for: 2m
    labels:
      severity: critical
    annotations:
      summary: "On instance {{ $labels.instance }} out of disk space"
      description: "Disk space device ({{ $labels.device }}) on instace {{ $labels.instance }} with IP:({{ $labels.IP }}) is almost full (< 10% left)
  - alert: High CPU Load
    expr: 100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[2m])) * 100) > 80
    for: 0m
    labels:
      severity: warning
    annotations:
      summary: "On instance {{ $labels.instance }} high CPU load"
      description: "Load CPU on instance {{ $labels.instance }} is (> 80%). CPU load in right now is: {{ $value }} %."
```

Рисунок 3.20 – Файл з правилами metrics-from-node-exporter.yml

В /etc/prometheus/metrics-from-py-exporter.yml додамо правила повязані на py-exporter: строк дії сертифікату та можливість підключення до серверу OpenVPN

```
groups:
- name: py-exporter
  rules:
    - alert: OpenVPN connection failed.
      expr: py_check_ovpn_connection == 0
      for: 1s
      labels:
        severity: critical
      annotations:
        summary: "On instance {{ $labels.exported_instance }} has problem"
        description: "From instance {{ $labels.exported_instance }} with IP:({{
    - alert: SSL certificate will expire soon.
      expr: py_check_ssl_ca_ovpn_expiry < 30
      for: 1m
      labels:
        severity: "critical"
      annotations:
        summary: "On instance {{ $labels.exported_instance }} has problem."
        description: "Certificate: {{ $labels.ca }} on instance {{ $labels.expo
```

Рисунок 3.21 – Файл з правилами metrics-from-py-exporter.yml

Перевіримо нашу конфігурації Prometheus за домогою утиліти яка іде разом з пакетом сервера: promtool check config /etc/prometheus/prometheus.yml

```
root@vpntermde1:/etc/prometheus# promtool check config /etc/prometheus/prometheus.yml
Checking /etc/prometheus/prometheus.yml
  SUCCESS: 2 rule files found

Checking /etc/prometheus/metrics-from-node-exporter.yml
  SUCCESS: 5 rules found

Checking /etc/prometheus/metrics-from-py-exporter.yml
  SUCCESS: 2 rules found
```

Рисунок 3.22 – Успішна перевірка синтаксису утилітою promtool

Після цього вже можна запускати служби на сервері: systemctl start prometheus prometheus-alertmanager prometheus-node-exporter grafana-server && systemctl enable prometheus prometheus-alertmanager prometheus-node-exporter grafana-server

3.2 Приклади роботи

Веб-інтерфейс Prometheus в даній роботі доступний за адресою http://5.61.32.169:9090/

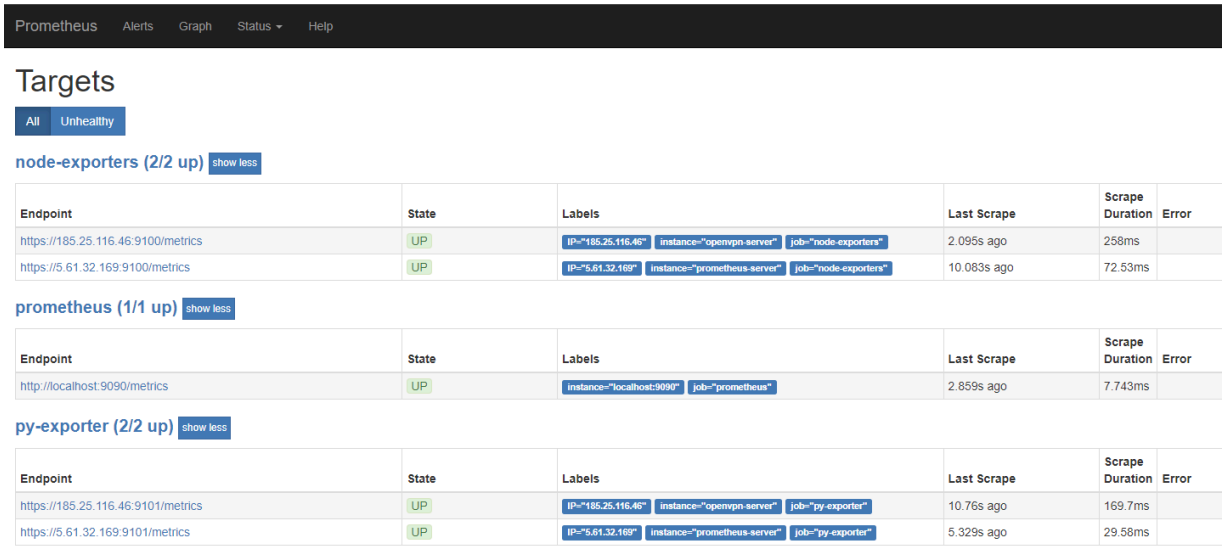


Рисунок 3.23 – Веб-інтерфейс Prometheus вкладка Targerts

Як можна побачити із скриншоту всі вузли які ми додавали у файл конфігурації Prometheus успішно розпізнані та знаходяться під спостереженням сервера Prometheus. Перевіримо правила які ми додавали для Alertmanager перейшовши у вкладку Alerts.

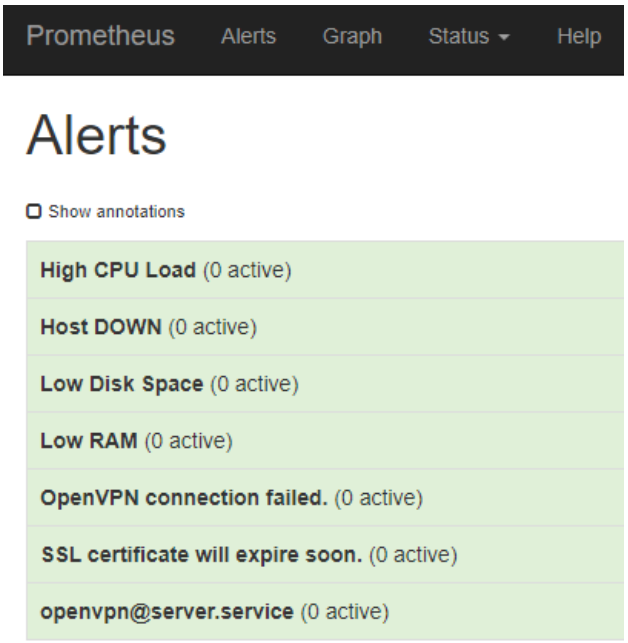


Рисунок 3.24 – Веб-інтерфейс Prometheus вкладка Alerts

Правила також успішно додані. Перевіримо метрики, котрі повинен створювати py-exporter перейшовши у вкладку Graph.

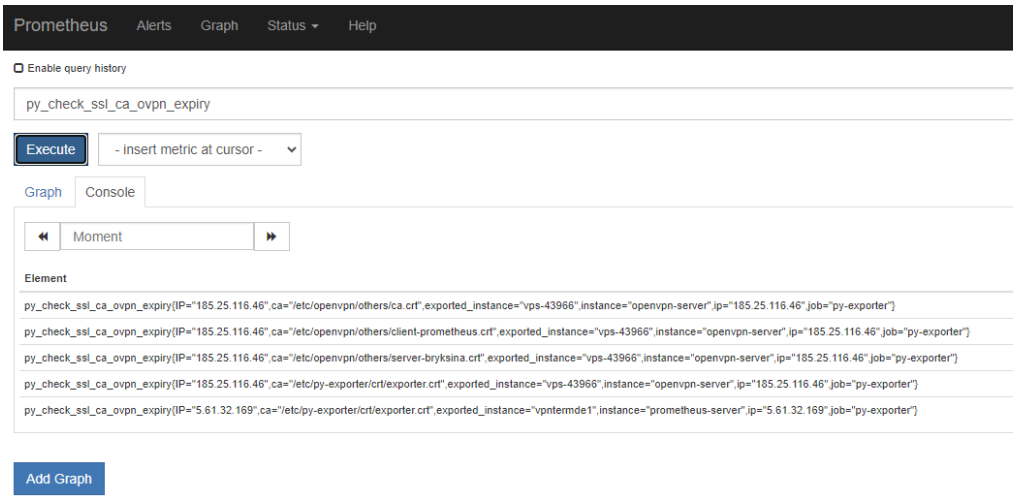


Рисунок 3.25 – Веб-інтерфейс Prometheus вкладка Graph метрика check_ssl

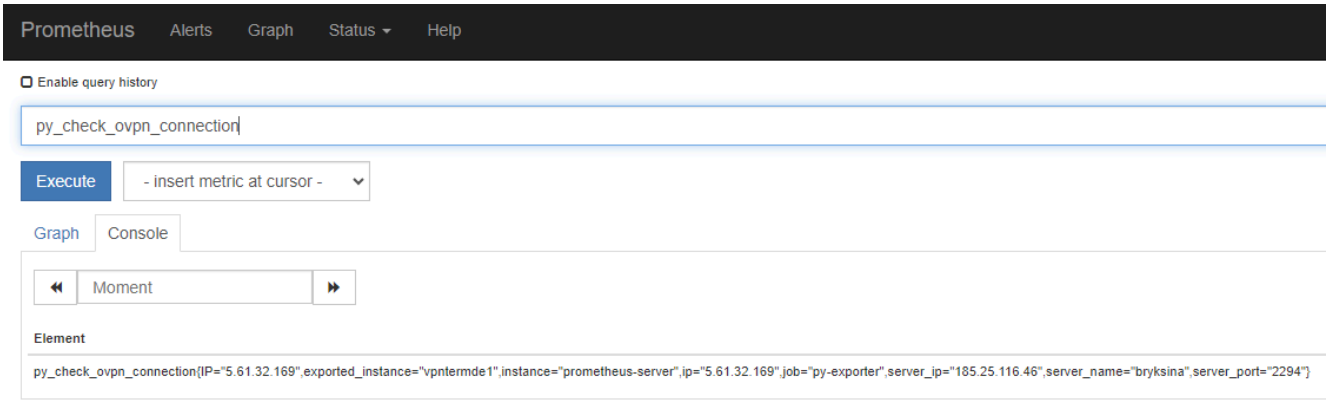


Рисунок 3.26 – Веб-інтерфейс Prometheus вкладка Graph метрика check_ovpn

З метриками також все впорядку. Перейдемо у веб-інтерфейс Grafana http://5.61.32.169:3000/. Стандартним паролем та логіном при першому вході являється admin admin, сервер відразу запропонує змінити пароль.

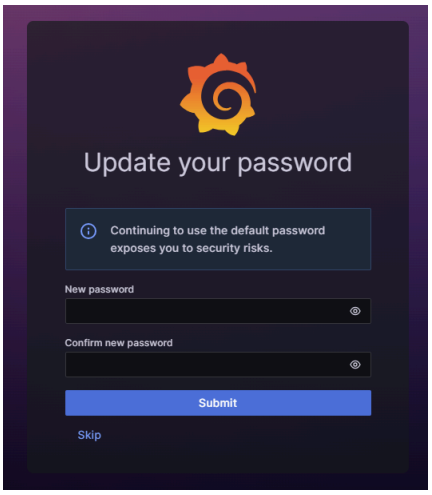


Рисунок 3.27 – Веб-інтерфейс Grafana стронінка логіну

Відразу додамо Data Source до Grafana перейшовши у вкладку Connections, а там в Add new connection та в пошуку знайдіть Prometheus та Prometheus Alertmanager.

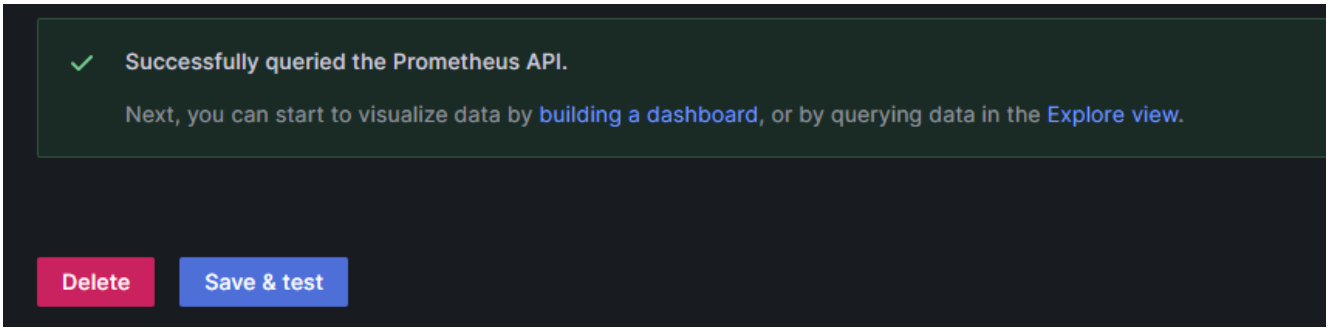


Рисунок 3.28 – Успішне додавання Data Source

Далі створимо Dashboard перейшовши у вкладку Dashboards и там натиснувши Create Dashboard, а після нього Add visualization.

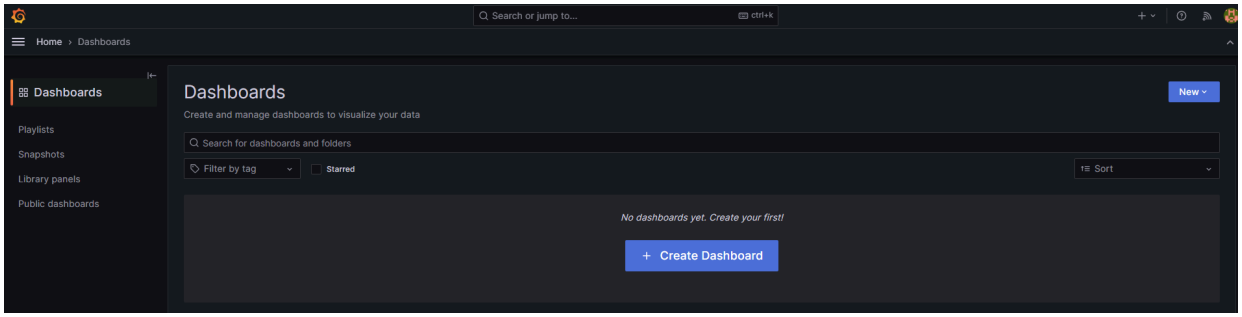


Рисунок 3.29 – Додавання дашбоарду

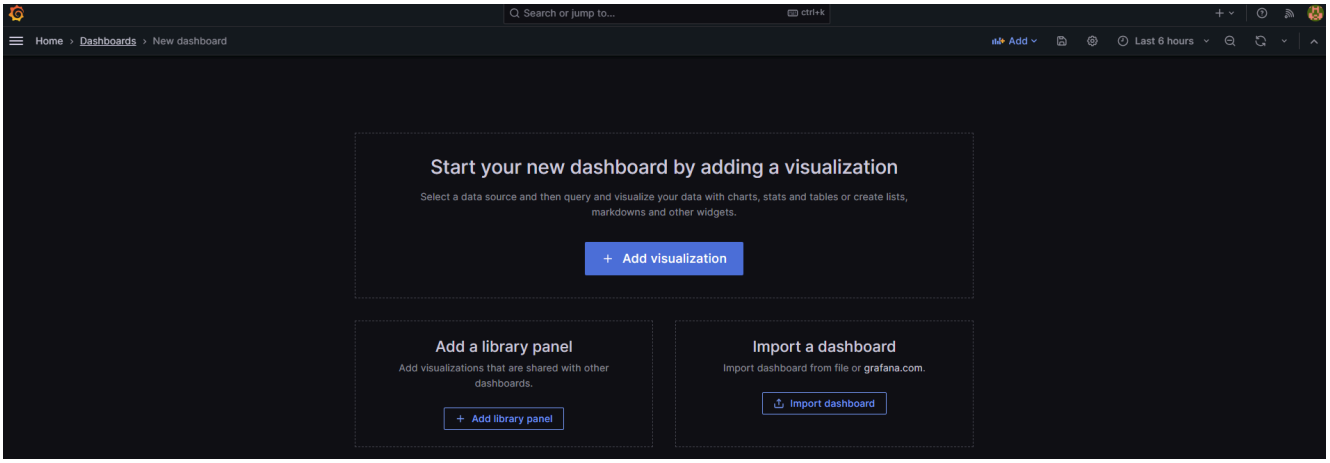


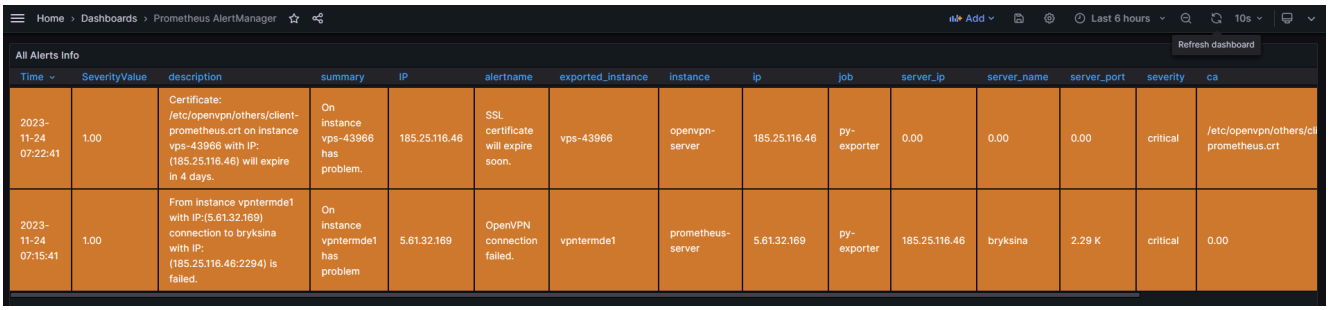
Рисунок 3.30 – Додавання візуалізації для дашбоарду

В наступній вкладці обираємо prometheus alertmanager, як Data Source, додаткових налаштувань не потрібно та Table(old) або Alert list для візуалізації повідомлень. Для тесту змінимо пароля для користувача bryksina на OpenVPN сервері та замінимо сертифікат який моніторимо на той який заекспарситься через 5 днів.

```
root@vps-43966:/etc/openvpn/others# bash openssl-ca.sh
########################################################################################################################
What need do?
1.) Create new client certificate and key.
2.) Revoke certificate.
3.) Verify certificate.
4.) Create server, client conf file for OpenVPN.
5.) Exit.
########################################################################################################################
Enter number 1 - 5: 1
Create new CA certificates or use already created? [new/old] (auto: old)
########################################################################################################################
Name for client: client-5-days
Which size certificate need in bits? |1024|2048|4096|8192| 1024
How many days the certificate must be valid? 5
Generating a RSA private key
.....................+++++
.....................+++++
writing new private key to 'client-5-days.key'
-----
Signature ok
subject=C = IT, ST = IT, L = IT, O = IT, OU = IT, CN = client-5-days
Getting CA Private Key
########################################################################################################################
Verify:
client-5-days.crt: OK
Enddate:
notAfter=Nov 29 05:18:22 2023 GMT
########################################################################################################################
root@vps-43966:/etc/openvpn/others# ls
ca.cnf  ca.key  cert              client-5-days.key      client-prometheus.key  crl      dh.pem         server-bryksina.crt  tls-crypt.key
ca.crt  ca.srl  client-5-days.crt  client-prometheus.crt  client-prometheus.ovpn  db.txt  openssl-ca.sh  server-bryksina.key
root@vps-43966:/etc/openvpn/others# cat client-5-days.crt > client-prometheus.crt
root@vps-43966:/etc/openvpn/others# cat ../usrps
bryksina:+GixnJeAeQly5cfnAAywb2DiR/bLvrQ0+eHVdcX5FfU
```

Рисунок 3.31 – Створення інцидентів для моніторингу

Тепер перейдемо в новостворений дашбоард, щоб перевірити коректність роботи системи.



Time	SeverityValue	description	summary	IP	alertname	exported_instance	instance	ip	job	server_ip	server_name	server_port	severity	ca
2023-11-24 07:22:41	1.00	Certificate: /etc/openvpn/others/client-prometheus.crt on instance vps-43966 with IP: (185.25.116.46) will expire in 4 days.	On instance vps-43966 has problem.	185.25.116.46	SSL certificate will expire soon.	vps-43966	openvpn-server	185.25.116.46	py-exporter	0.00	0.00	0.00	critical	/etc/openvpn/others/cli prometheus.crt
2023-11-24 07:15:41	1.00	From instance vpntermde1 with IP:(5.61.32.169) connection to bryksina with IP: (185.25.116.46:2294) is failed.	On instance vpntermde1 has problem	5.61.32.169	OpenVPN connection failed.	vpntermde1	prometheus-server	5.61.32.169	py-exporter	185.25.116.46	bryksina	2.29 K	critical	0.00

Рисунок 3.32 – Dashboard Alertmanager

Як можна помітити із скриншоту система працює корректно. Ми отримали повідомлення про невдале зєднання із сервером OpenVPN і що у сертифіката незабаром закінчиться строк дії.

Також ми отримали повідомлення від бота в Telegram із аналогічним змістом.

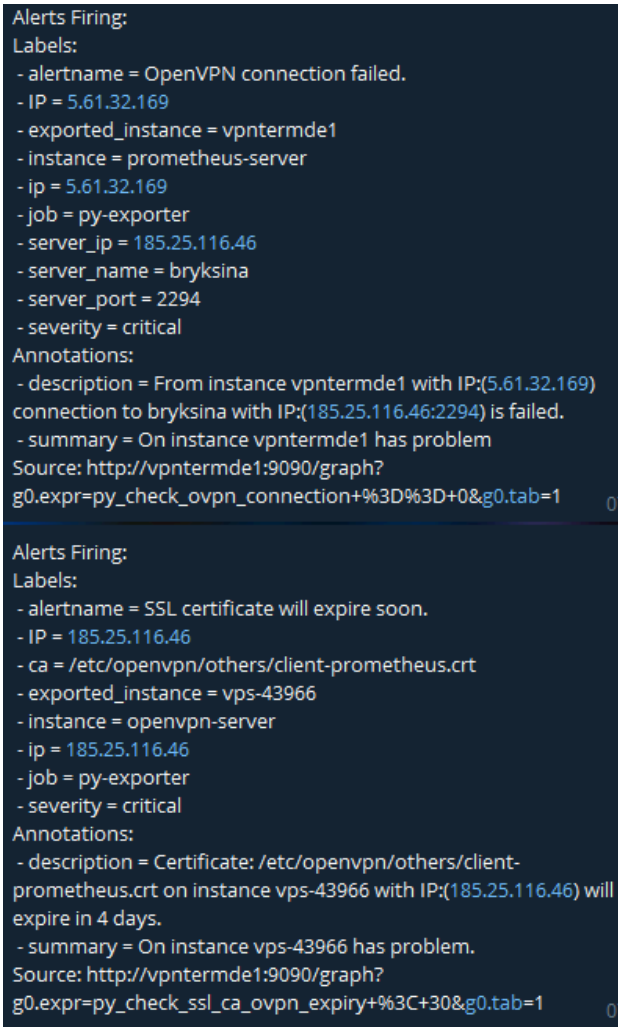


Рисунок 3.33 – Повідомлення від бота в Telegram

На цьому робота завершена.

ВИСНОВОК

Сучасні операційні системи представляють собою складний набір програмних інструментів, які дозволяють користувачам ефективно взаємодіяти з комп'ютером та керувати його процесами. Програмний інтерфейс операційних систем спрощує взаємодію з комп'ютером, уніфікує системні процеси і полегшує роботу з різними компонентами обчислювальної техніки. Кожна операційна система визначає набір функцій для обміну інформацією між процесами, службами та файлами, включаючи відкриття, читання, управління та закриття файлів. На платформі Unix Debian виділяється своєю сучасністю та зручністю. Постійні оновлення і обширний вибір програмного забезпечення в репозиторіях роблять її найбільш сучасною серед сімейства Linux. Це особливо актуально у сфері серверних застосувань, де Debian є найпопулярнішим вибором завдяки своїм перевагам.

OpenVPN потужний та гнучкий інструмент для налаштування віртуальних приватних мереж (VPN). Його відкритий код і кросплатформенність роблять його популярним вибором для забезпечення безпеки з'єднань у різних операційних системах. Технологія визначається високим рівнем шифрування, яке гарантує конфіденційність даних під час їх передачі через мережу. Його здатність працювати на основі різних протоколів, таких як TCP та UDP, надає йому гнучкість у використанні залежно від конкретних потреб користувача чи організації. Вона також володіє великою спільнотою користувачів і розробників, що призводить до швидкого виявлення та виправлення потенційних проблем. Його здатність працювати в різних сценаріях, включаючи використання на серверах та в корпоративних мережах, робить його ідеальним інструментом для створення безпечних та надійних VPN-з'єднань.

Мова програмування Python відома своєю гнучкістю та ефективністю, що робить її ідеальним вибором для початківців і досвідчених розробників. Інтеграція з багатьма бібліотеками та фреймворками дозволяє розробникам працювати над

широким спектром проектів і вирішувати різноманітні завдання, як наприклад ті котрі були вирішені в даній кваліфікаційній роботі.

Prometheus став ключовим гравцем у світі моніторингу завдяки своїй ефективності, гнучкості та здатності адаптуватися до різноманітних сценаріїв використання. Однією з його основних переваг є модель даних, яка дозволяє збирати дані з різних джерел та легко їх опрацьовує. Це дозволяє розробникам та системним адміністраторам отримувати повний обсяг інформації щодо працездатності системи.

Принцип роботи Prometheus полягає в тому, що він збирає дані за допомогою власного механізму для витягування інформації з об'єктів моніторингу. Це важливий аспект, оскільки це робить його легким у використанні та налаштуванні. Крім того, він має підтримку для метрик, що дозволяє визначати і вимірювати різні параметри функціонування системи.

В ході роботи було встановлено, що моніторинг мережі в сучасному інформаційному середовищі є необхідним елементом для забезпечення ефективності, надійності та безпеки інформаційних систем. Важливість систем моніторингу мережі визначається їхньою здатністю забезпечувати розуміння стану інфраструктури, вчасне виявлення проблем та можливість прийняття інформованих рішень. Високий рівень конкуренції та зростання обсягів даних роблять мережевий моніторинг ключовим аспектом ефективного управління IT-ресурсами, що є актуальною проблемою в сучасному світі. Системи моніторингу надають можливість відслідковувати використання ресурсів, виявляти аномалії та попереджати про можливі збої, що сприяє невідкладній реакції та усуненню проблем.

В даній роботі було виконано практичну реалізацію впровадження моніторингу сервера OpenVPN за допомогою Prometheus з метою підвищення швидкості реакції на виникаючі інциденти на сервері. Практичну частину цієї роботи можна використовувати як практичний посібник реалізації данного

рішення, що сприятиме ефективному керуванню мережею та забезпечуватиме її надійність і безпеку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Олифер, В. Г., Олифер Н.А. (2001). Комп'ютерні мережі. Принципи, технології, протоколи: навч. посібник.
2. Соколов, О. В., Шаньгін, В. Ф. (2002). Захист інформації у розподілених корпоративних мережах та системах: навч. посібник.
3. Колесніков, О. В., Хетч Б. (2002). LINUX. Створення віртуальних приватних мереж (VPN): навч. посібник.
4. Петренко С. (2001). Захищена віртуальна приватна мережа: сучасний погляд на захист конфіденційних даних/Світ Internet. - М.: № 2.
5. Мізюк О. (2021) Путівник по Linux. Київ.
6. Гончарова Л., Возненко А., Стасюк О., Коваль Ю. (2013). Основи захисту інформації в телекомунікаційних та комп'ютерних мережах.
7. Стивен Б. (2017). Віртуальні приватні мережі: навч. посібник.
8. OpenVPN Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://openvpn.net/vpn-server-resources/
9. OpenVPN: інструкція по застосуванню [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/articles/665036/
10. What is OpenVPN? [Електронний ресурс] – Режим доступу до ресурсу: https://www.cactusvpn.com/beginners-guide-to-vpn/what-is-openvpn/
11. Prometheus [Електронний ресурс] – Режим доступу до ресурсу: https://prometheus.io/docs/introduction/overview/
12. Моніторинг сервісів з Prometheus [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/companies/selectel/articles/275803/

13. Prometheus Monitoring: Use Cases, Metrics, and Best Practices [Електронний ресурс] – Режим доступу до ресурсу: https://www.tigera.io/learn/guides/prometheus-monitoring/

14. Python для мережевих інженерів [Електронний ресурс] – Режи доступу до ресурсу: https://pyneng.readthedocs.io/ru/latest/#

15. The Python Standard Library [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/library/

16. Centralized Log Management, Monitoring and Analysis Software. [Електронний ресурс] – Режим доступу до ресурсу: https://www.nagios.com/products/nagios-log-server/

17. Additional Documentation // openvpn.net [Електронний ресурс] – Режим доступу до ресурсу: https://openvpn.net/communityresources/how-to/, 15.09.2021.

18. OpenVpn community and Wiki // [Електронний ресурс] – Режим доступу до ресурсу: https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage, 15.09.2021.

19. How a VPN Helps with Network Security // OpenVpn [Електронний ресурс] – Режим доступу до ресурсу: https://openvpn.net/blog/vpns-and-network-security/

20. Моніторинг клієнтського досвіду. [Електронний ресурс] – Режим доступу до ресурсу: https://newrelic.com/resources/datasheets/customer-experience-monitoring

21. Adnan Abdulazeez Comparison of VPN Protocols at Network Layer Focusing on Wire Guard Protocol: Iraq, 2021. // [Електронний ресурс] – Режим доступу до ресурсу: https://www.learntechlib.org/p/218341

22. Monitoring dashboard. [Електронний ресурс] – Режим доступу до ресурсу: https://dashthis.com/monitoring-dashboard/

23. GNU/Linux. [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Linux

24. З’єднання Grafana і Prometheus. [Електронний ресурс] – Режим доступу до ресурсу: https://www.metricfire.com/blog/connecting-prometheus-and-grafana/

25. Конфігурація Prometheus. [Електронний ресурс] – Режим доступу до ресурсу: https://prometheus.io/docs/prometheus/latest/configuration/configuration/

26. Dashboards Grafana. [Електронний ресурс] – Режим до ресурсу: https://grafana.com/grafana/dashboards/ (дата звернення: 04.05.20

27. About Grafana panels. [Електронний ресурс] – Режим доступу до ресурсу: https://grafana.com/docs/grafana/latest/panels/

28. AlertManager. [Електронний ресурс] – Режим доступу до ресурсу: https://prometheus.io/docs/alerting/latest/alertmanager/

29. Create alerts Grafana. [Електронний ресурс] – Режим доступу до ресурсу: https://grafana.com/docs/grafana/latest/alerting/old-alerting/create-alerts/

30. What Is SSH: Understanding Encryption, Ports and Connection. [Електронний ресурс] – Режим доступу до ресурсу: https://www.hostinger.com/tutorials/ssh-tutorial-how-does-sshwork#What_Is_SSH

31. Feilner M. Open VPN Building and Operating Virtual Private Networks. Birmingham : Packt, 2006. 270 c.

32. Suhail, A., & Navarro, L. (2018). "A Survey of Open Source Data Monitoring and Visualization Tools." In International Conference on Open Source Systems. Springer, Cham.

33. Turnbull, J. (2018). "The Prometheus Monitoring System: A Practical Guide to the Most Popular Monitoring System." O'Reilly Media.

34. Ghorbani, A. (2015) Characterization of Encrypted and VPN Traffic using Time-related Features, University of New Brunswick.

35. Markus, F. (2006). Openvpn: Building and Integrating Virtual Private Networks: навч. посібник.

36. Muhammad Iqbal, Imam Riadi (2019) Analysis of Security Virtual Private Network (VPN) Using OpenVPN, International Journal of Cyber-Security and Digital Forensics.

37. Eric F. Crist, Jan Just Keijser (2015). Mastering OpenVPN: навч. посібник.

38. Zhensheng Zhang, Ya-Qin Zhang, Xiaowen Chu & Bo Li (2004) An Overview of Virtual Private Network (VPN): IP VPN and Optical VPN, Photonic Network Communications.

39. Richard C. Harlan (2006) Wireless Sensor Network-Management System An Adaptive Policy-Based Management for Wireless Sensor Networks, WinM

40. Eric F. Crist and Jan Just Keijser. (2015). Mastering OpenVPN, Packt Publishing.

41. Jan Just Keijser (2017). OpenVPN Cookbook - 2nd Edition. Packt Publishing.

42. Canavan J. (2001). Fundamentals of Network Security. Boston, London: Artech House

43. SSL Remote Access VPNs (Network Security). Qiang Huang, Jazib Frahim,. Cisco Press, 2008.

44. Sawalmeh, Hanan, et al. "VPN Remote Access OSPF-based VPN Security Vulnerabilities and Counter Measurements." 2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT). IEEE, 2021.

45. Gerardus Blokdyk. Server Room Environment Monitoring System a Complete Guide, 2020 Edition (Kindle edition).

46. Brian Brazil. Prometheus: Up & Running: Infrastructure and Application Performance, 1st Edition, 2018, O`Reilly Media, Inc.

47. Thomas Kurian Theakanath. Datadog Cloud Monitoring Quick Start Guide, June 2021.

48. David Carasso. Exploring Splunk, 2012, New York.

49. Hoch D. (2001). Linux system and Performance monitoring.

50. Joel Bastos, Pedro Araujo. Hands-on Infrastructure Monitoring with Prometheus: Implement and scale queries, dashboards, and alerting across machines and containers, 2019, Packt.

51. Brian Brazil. Prometheus: Up & Running: Infrastructure and Application Performance, 1st Edition, 2018, O`Reilly Media, Inc.

Додаток А

Код bash скрипта

```
#!/bin/bash
tmpdir=/etc/openvpn/others/
cnfconf=$tmpdir/ca.cnf
function block {
printf "%0$(tput cols)d" 0 | tr '0' '#'
}
function create-client-cert {
block
cd $tmpdir
read -p "Name for client: " FILENAME
read -p "Which size certificate need in bits? |1024|2048|4096|8192| " BITSCLIENT
read -p "How many days the certificate must be valid? " DAYSCLIENT
openssl req -nodes -newkey rsa:$BITSCLIENT -keyout $FILENAME.key -keyform PEM -out $FILENAME.csr -outform PEM -subj "/C=IT/ST=IT/L=IT/O=IT/OU=IT/CN=$FILENAME"
openssl x509 -req -in $FILENAME.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out $FILENAME.crt -days $DAYSCLIENT
block
echo "Verify: "
openssl verify -verbose -CAfile ca.crt $FILENAME.crt
echo "Enddate: "
openssl x509 -enddate -noout -in $FILENAME.crt
block
#chmod 600 $tmpdir/$FILENAME.crt $$ chmod 600 $tmpdir/$FILENAME.key
rm $FILENAME.csr
}
function server-local-users-auth {
echo "script-security 2
auth-user-pass-verify /etc/openvpn/verify-user.sh via-file
username-as-common-name" >> $tmpdir/server.conf
touch $tmpdir/verify-user.sh && chmod 555 $tmpdir/verify-user.sh
touch $tmpdir/usrps && chmod 744 $tmpdir/usrps
```

```
if [[ ! -f
"/var/log/openvpn/openv
pn-auth.log" ]]; then
touch
/var/log/openvpn/openvp
n-auth.log
fi
echo '#!/bin/bash
USERS=`cat
/etc/openvpn/usrps`
vpn_verify() {
if [ ! $1 ] || [ ! $2 ]; then
exit 1
fi
for i in $USERS; do
if [ "$i" = "$1:$2" ]; then
echo `date` auth LOGIN:
$1 >>
/var/log/openvpn/openvp
n-auth.log
exit 0
fi
done
echo `date` auth
WRONG: $1 >>
/var/log/openvpn/openvp
n-auth.log
}
if [ ! $1 ] || [ ! -e $1 ];
then
exit 1
fi
vpn_verify `cat $1`
exit 1' >
$tmpdir/verify-user.sh
echo "client1:$(openssl
rand -base64 32)" >
$tmpdir/usrps
}
function revoke-cert {
block
read -p "Name of the
client certificate: "
CCRT
openssl ca -config
$cnfconf -revoke
$tmpdir/$CCRT.crt
openssl ca -gencrl
-config $cnfconf -out
$tmpdir/crl.pem
block
}
function create-dh {
block
read -p "Which size dh
certificates need in bits?
|2048|4096|8192| "
BITSDH
openssl dhparam -out
$tmpdir/dh.pem
$BITSDH
block
}
function create-CA-cert
{
block
read -p "Which size
certificate need in bits?
|1024|2048|4096|8192| "
BITS
read -p "How many days
the certificate must be
valid? " DAYS
openssl req -new
-newkey rsa:$BITS
-sha512 -x509 -nodes
-days $DAYS -out
$tmpdir/ca.crt -keyout
$tmpdir/ca.key -subj
"/C=IT/ST=IT/L=IT/O=I
T/OU=IT/CN=CA"
#-sha256
block
openssl x509 -enddate
-noout -in $tmpdir/ca.crt
chmod 600
$tmpdir/ca.crt &&
chmod 600
$tmpdir/ca.key
block
}
function verify-crt {
block
read -p "Path to CA
certificate: " CACRT
read -p "Path to client
certificate: "
CLIENTCRT
openssl verify -verbose
-CAfile $CACRT
$CLIENTCRT
block
}
function
create-client-file {
touch
$tmpdir/$CNAME.ovpn
#openvpn --genkey
--secret
$tmpdir/tls-crypt.key
echo "client" >>
$tmpdir/$CNAME.ovpn
if [[ $CPROTOCOL -eq
1 ]]; then
echo "proto udp" >>
$tmpdir/$CNAME.ovpn
elif [[ $CPROTOCOL
-eq 2 ]]; then
echo "proto tcp-client"
>>
$tmpdir/$CNAME.ovpn
else
block
```

```
read -p "Use udp or tcp for client? [tcp/udp] (auto: tcp): " CPROT
if [[ $CPROT == "tcp" || $CPROT == "TCP" || $CPROT == "" ]]; then
echo "proto tcp-client" >> $tmpdir/$CNAME.ovpn
elif [[ $CPROT == "udp" || $CPROT == "UDP" ]]; then
echo "proto udp" >> $tmpdir/$CNAME.ovpn
fi
block
fi
if [[ $APORT -eq 1 ]]; then
CPORT="1194"
elif [[ $APORT -eq 2 ]]; then
CPORT=$PORT
elif [[ $APORT -eq 3 ]]; then
CPORT=$RPORT
else
block
read -p "Which port use for openvpn? (auto: 1194): " -i 1194 CAPORT
CPORT=$CAPORT
block
fi
echo -e "remote $(ip -4 addr | sed -ne 's|^.* inet \([^/]*\)/.* scope global.*$|\1|p' | head -1) $CPORT" >> $tmpdir/$CNAME.ovpn
echo "dev tun
nobind
persist-key
persist-tun
topology subnet
auth-nocache
auth-user-pass
verb 4" >> $tmpdir/$CNAME.ovpn
if [[ $CIPHER == "" ]]; then
echo "without cigher"
else
echo -e "cipher $CIPHER" >> $tmpdir/$CNAME.ovpn
fi
echo '<ca>' >> $tmpdir/$CNAME.ovpn
&& cat $tmpdir/ca.crt >> $tmpdir/$CNAME.ovpn
&& echo '</ca>' >> $tmpdir/$CNAME.ovpn
echo '<cert>' >> $tmpdir/$CNAME.ovpn
&& cat $CNAME.crt >> $tmpdir/$CNAME.ovpn
&& echo '</cert>' >> $tmpdir/$CNAME.ovpn
echo '<key>' >> $tmpdir/$CNAME.ovpn
&& cat $CNAME.key >> $tmpdir/$CNAME.ovpn
&& echo '</key>' >> $tmpdir/$CNAME.ovpn
echo '<tls-crypt>' >> $tmpdir/$CNAME.ovpn
&& cat $tmpdir/tls-crypt.key >> $tmpdir/$CNAME.ovpn
&& echo '</tls-crypt>' >> $tmpdir/$CNAME.ovpn
#rm $tmpdir/tls-crypt-v2-client.key
}
function send-in-file {
until [[ $ANSWSEND =~ ^[1-2]$ ]]; do
block
echo "1.) Create server and client file."
echo "2.) Create only client file."
read -p "Enter number 1 or 2 (auto: 2) " ANSWSEND
done
if [[ $ANSWSEND -eq 2 || $ANSWSEND = "" ]]; then
ls -la $tmpdir
cd $tmpdir
read -p "Enter name of client certificate: " CNAME
create-client-file
block
cat $tmpdir/$CNAME.ovpn
block
elif [[  $ANSWSEND -lt 1 ||  $ANSWSEND -gt 2 ]]; then
echo "Please enter number 1 - 2"
else
block
```

```
if [[ -f  $tmpdir/dh.pem
]]; then
echo "dh.pem is exist."
block
read -p "Create new dh
certificates or use
already created?
[new/old] (auto: old): "
DHANSW
if [[ $DHANSW = 'new'
|| $DHANSW = 'NEW' ||
$DHANSW = 'New' ]];
then
create-dh
elif [[ $DHANSW = "" ||
$DHANSW = 'old' ||
$DHANSW = 'OLD' ||
$DHANSW = 'Old' ]];
then
echo "Use old dh cert."
fi
else
create-dh
fi
openvpn --genkey
--secret
$tmpdir/tls-cryp
ls -la $tmpdir
cd $tmpdir
read -p "Enter name of
server certificate: "
SNAME
read -p "Enter name of
client certificate: "
CNAME
touch
$tmpdir/server.conf
touch
$tmpdir/$CNAME.ovpn
block
echo "What port do you
want OpenVPN to listen
to?"
echo "1) Default: 1194"
echo "2) Custom."
echo "3) Random
[49152-65535]"
block
until [[ $APORT =~
^[1-3]$ ]]; do
read -rp "Port choice
[1-3]: " -e -i 1 APORT
block
done
case $APORT in
1)
echo 'port 1194' >>
$tmpdir/server.conf
;;
2)
until [[ $PORT =~
^[0-9]+$ ]] && [
"$PORT" -ge 1 ] && [
"$PORT" -le 65535 ]; do
read -rp "Custom port
[1-65535]: " -e -i 1194
PORT
done
echo -e "port $PORT"
>> $tmpdir/server.conf
;;
3)
RPORT=$(shuf
-i49152-65535 -n1)
echo "Random Port is:
$RPORT"
echo -e "port $RPORT"
>> $tmpdir/server.conf
;;
esac
echo "What protocol do
you want OpenVPN to
use?"
echo "UDP is faster.
Unless it is not available,
you shouldn't use TCP."
echo "1) UDP."
echo "2) TCP."
until [[ $CPROTOCOL
=~ ^[1-2]$ ]]; do
read -rp "Protocol [1-2]:
" -e -i 1 CPROTOCOL
done
case $CPROTOCOL in
1)
echo "proto udp" >>
$tmpdir/server.conf
;;
2)
echo "proto tcp" >>
$tmpdir/server.conf
;;
esac
echo "dev tun
user nobody
group nogroup
duplicate-cn
persist-key
persist-tun
keepalive 10 120
topology subnet
server 10.8.0.0
255.255.255.0
status
/var/log/openvpn/status.l
og
verb 3" >>
$tmpdir/server.conf
echo 'push "dhcp-option
DNS 1.1.1.1"' >>
$tmpdir/server.conf
```

```
echo 'push "dhcp-option
DNS 8.8.8.8"' >>
$tmpdir/server.conf
echo 'push
"redirect-gateway def1
bypass-dhcp"' >>
$tmpdir/server.conf
read -p "Add local auth
on server [yes/no] (auto:
no): " ANSWLAUTH
if [[ $ANSWLAUTH ==
"yes" || $ANSWLAUTH
== "YES" ||
$ANSWLAUTH == "y"
]]; then
server-local-users-auth
fi
echo "Choose which
cipher you want to use
for the data channel:"
echo "   1)
AES-128-GCM"
echo "   2)
AES-192-GCM"
echo "   3)
AES-256-GCM"
echo "   4)
AES-128-CBC"
echo "   5)
AES-192-CBC"
echo "   6)
AES-256-CBC"
until [[ $CCIPHER =~
^[1-6]$ ]]; do
read -rp "Ciphe
-e -i 6 CCIPHE
done
case $CCIPHER in
1)
CIPHER="AES-128-GC
M"
;;
2)
CIPHER="AES-192-GC
M"
;;
3)
CIPHER="AES-256-GC
M"
;;
4)
CIPHER="AES-128-CB
C"
;;
5)
CIPHER="AES-192-CB
C"
;;
6)
CIPHER="AES-256-CB
C"
;;
esac
echo "Which digest
algorithm do you want to
use for HMAC?"
echo "   1) SHA-256
(recommended)"
echo "   2) SHA-384"
echo "   3) SHA-512"
until [[ $AHMAC =~
^[1-3]$ ]]; do
read -rp "Digest
algorithm [1-3]: " -e -i 1
AHMAC
done
case $AHMAC in
1)
HMACALG="SHA256"
;;
2)
HMACALG="SHA384"
;;
3)
HMACALG="SHA512"
;;
esac
echo -e "auth
$HMACALG
cipher $CIPHER
tls-server" >>
$tmpdir/server.conf
touch
/var/log/openvpn/status.l
og
echo '<ca>' >>
$tmpdir/server.conf &&
cat $tmpdir/ca.crt >>
$tmpdir/server.conf &&
echo '</ca>' >>
$tmpdir/server.conf
echo '<dh>' >>
$tmpdir/server.conf &&
cat $tmpdir/dh.pem >>
$tmpdir/server.conf &&
echo '</dh>' >>
$tmpdir/server.conf
echo '<cert>' >>
$tmpdir/server.conf &&
cat $SNAME.crt >>
$tmpdir/server.conf &&
echo '</cert>' >>
$tmpdir/server.conf
echo '<key>' >>
$tmpdir/server.conf &&
cat $SNAME.key >>
$tmpdir/server.conf &&
echo '</key>' >>
$tmpdir/server.conf
echo '<tls-crypt>' >>
$tmpdir/server.conf &&
cat $tmpdir/tls-crypt.key
>> $tmpdir/server.conf
```

```
&& echo '</tls-crypt>'
>> $tmpdir/server.conf
create-client-file
read -p "Move auth
script and usrps file to
openvpn directory?
[yes/no] (auto: yes) "
ANSWMOVE
if [[ $ANSWMOVE ==
"yes" || $ANSWMOVE
== "YES" ||
$ANSWMOVE == "" ]];
then
mv
$tmpdir/verify-user.sh
/etc/openvpn/
mv $tmpdir/usrps
/etc/openvpn/
read -p "Move server
conf to openvpn
directory? [yes/no]
(auto: yes) "
ANSWMSC
if [[ $ANSWMSC ==
"yes" || $ANSWMSC ==
"YES" ||  $ANSWMSC
== "" ]]; then
mv $tmpdir/server.conf
/etc/openvpn/
fi
fi
block
cat $tmpdir/server.conf
block
cat
$tmpdir/$CNAME.ovpn
block
echo -e "$(tput setaf
1)For openvpn to work,
you still need to enable
net.ipv4.ip_forward = 1
in the /etc/sysctl.conf file
and do sysctl -p.\nYou
also need to add iptables
rule for forwarding and
enable port on openvpn
work:\niptables -A
FORWARD -i tun+ -o
eth0 -j
ACCEPT\niptables -A
FORWARD -i eth0 -o
tun+ -j ACCEPT$(tput
sgr 0)"
block
fi
}
function create-cnf-file {
touch $cnfconf
echo '[ ca ]' >> $cnfconf
echo 'default_ca =
default # The default ca
section' >> $cnfconf
echo '[ default ]' >>
$cnfconf
echo -e "dir = $tmpdir"
>> $cnfconf
mkdir -p $tmpdir/cert
echo -e "certs =
$tmpdir/cert # Where the
issued certs are kept" >>
$cnfconf
mkdir -p $tmpdir/crl
echo -e "crl_dir =
$tmpdir/crl # Where the
issued crl are kept" >>
$cnfconf
touch $tmpdir/db.txt
echo -e "database =
$tmpdir/db.txt #
database index file." >>
$cnfconf
echo -e "new_certs_dir =
$tmpdir/cert # default
place for new certs." >>
$cnfconf
echo -e "certificate =
$tmpdir/ca.crt # The CA
certificate" >> $cnfconf
echo -e "crl =
$tmpdir/crl.pem # The
current CRL" >>
$cnfconf
echo -e "private_key =
$tmpdir/ca.key # The
private key" >> $cnfconf
echo 'default_days =
3650 # how long to
certify for' >> $cnfconf
echo 'default_crl_days =
3650 # how long before
next CRL' >> $cnfconf
echo 'default_md =
sha256 # use SHA-256
by default' >> $cnfconf
echo '[ req ]' >> $cnfconf
echo 'default_bits =
2048' >> $cnfconf
echo 'default_md =
sha256' >> $cnfconf
echo '[
req_distinguished_name
]' >> $cnfconf
echo 'countryName = IT'
>> $cnfconf
echo
'countryName_default =
IT' >> $cnfconf
echo
'stateOrProvinceName =
IT' >> $cnfconf
```

```
echo 'stateOrProvinceName_default = IT' >> $cnfconf
echo 'localityName = IT' >> $cnfconf
echo 'localityName_default = IT' >> $cnfconf
echo '0.organizationName = IT' >> $cnfconf
echo '0.organizationName_default = IT' >> $cnfconf
echo 'organizationalUnitName = IT' >> $cnfconf
echo 'organizationalUnitName_default = IT' >> $cnfconf
echo 'commonName = certificate' >> $cnfconf
echo 'commonName_max = 64' >> $cnfconf
}
until [[ $ANSWOLD =~ ^[1-5]$ ]]; do
block
echo "What need do?"
echo "1.) Create new client certificate and key."
echo "2.) Revoke certificate."
echo "3.) Verify certificate."
echo "4.) Create server, client conf file for OpenVPN."
echo "5.) Exit."
block
read -p "Enter number 1 - 5: " ANSWOLD
if [[ $ANSWOLD -eq 1 ]]; then
read -p "Create new CA certificates or use already created? [new/old] (auto: old) " ANSW
if [[ $ANSW = 'new' || $ANSW = 'NE
$ANSW = 'Ne
 if [[ -d $tmpdir ]]; then
block
echo -e "Directory $tmpdir is exist."
block
cd $tmpdir
create-CA-cert
create-cnf-file
create-client-cert
 else
block
echo -e "Directory $tmpdir is not exist. Create directory."
block
mkdir -p $tmpdir
cd $tmpdir
create-CA-cert
create-cnf-file
create-client-cert
 fi
elif [[  $ANSW = "" || $ANSW = 'old' || $ANSW = 'OLD' || $ANSW = 'Old' ]]; then
create-client-cert
fi
elif [[ $ANSWOLD -eq 2 ]]; then
revoke-cert
elif [[ $ANSWOLD -eq 3 ]]; then
verify-crt
elif [[ $ANSWOLD -eq 4 ]]; then
send-in-file
elif [[ $ANSWOLD -eq 5 ]]; then
exit 0
else
echo -e "$(tput setaf 1)Please enter number from 1 to 5.$(tput sgr 0)"
fi done
```

Додаток Б

Код python скрипта для моніторингу сертифікатів OpenVPN

```
#!/usr/bin/env python3
import ssl
import time
import syslog
import socket
import logging
import threading
import subprocess
```

```
from sys import argv
from flask import Flask
from flask_httpauth import
HTTPBasicAuth
from concurrent.futures import
ThreadPoolExecutor
from prometheus_client import
make_wsgi_app, Gauge
exporter_port = 9101
exporter_ip = "185.25.116.46"
exporter_login = "py-exporter"
exporter_password =
"DC6xFm3bat926mnp9J82KnMHA6j6
MCa9"
exporter_certificate_ssl =
"/etc/py-exporter/crt/exporter.crt"
exporter_key_ssl =
"/etc/py-exporter/crt/exporter.key"
check_ca_paths =
[exporter_certificate_ssl]
py_metrics =
Gauge('py_check_ssl_ca_ovpn_expiry',
'Custom metric on python.', ['ip',
'instance', 'ca'])
app = Flask(__name__)
auth = HTTPBasicAuth()
# Disables all log info from Flask
app.debug = False
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)
app.logger.setLevel(logging.ERROR)
@auth.verify_password
def verify_password(username,
password):
    if username == exporter_login and
password == exporter_password:
        #
syslog.syslog(syslog.LOG_WARNING,
f"User {username} auth accept.")
        return True
    if username == "":
        return False
 syslog.syslog(syslog.LOG_WARNING,
f"User {username} auth failed.")
    return False
@app.route('/metrics')
@auth.login_required
def metrics():
    return make_wsgi_app()
def run_server(ip, port, ssl_cert,
ssl_key):
    context =
ssl.SSLContext(ssl.PROTOCOL_TLSv1
_2)
    context.load_cert_chain(ssl_cert,
ssl_key)
    app.run(host=ip, port=port,
ssl_context=context)
def check_path(path):
    try:
        subprocess.check_output(['ls',
path], stderr=subprocess.DEVNULL)
        return True
    except subprocess.CalledProcessError
as error:
        # syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with cert path:
{error}")
        return False
def check_ca_days(ca_path):
    command = "echo $((($(date -d
\"$(openssl x509 -enddate -noout -in
path | sed 's/notAfter=//')\" +%s) - $(date
+%s)) / 86400))"
    try:
        days =
subprocess.run(command.replace("path"
, ca_path), shell=True,
capture_output=True, text=True)
        return days.stdout
    except subprocess.CalledProcessError
as error:
```

```
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error: {error}.")
def cert_executor():
    try:
        with ThreadPoolExecutor() as
executor:
            for path in check_ca_paths:
                if check_path(path):
py_metrics.labels(ip=exporter_ip,
instance=socket.gethostname(),
ca=path).set(0)
executor.submit(cert_collect_metrics,
path)
                else:
syslog.syslog(syslog.LOG_WARNING,
f"Skipping path {path}. Maybe cert is
not exists.")
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with cert
executor: {error}.")
def cert_collect_metrics(path):
    try:
        py_metrics.labels(ip=exporter_ip,
instance=socket.gethostname(),
ca=path).set(int(check_ca_days(path)))
        time.sleep(120)
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with metrics:
{error}.")
if __name__ == '__main__':
    if len(argv) > 1:
        if argv[1] == "start":

syslog.syslog(syslog.LOG_WARNING,
f"Starting py-exporter.")
            if
check_path(exporter_certificate_ssl) and
check_path(exporter_key_ssl):
threading.Thread(target=run_server,
                             args=(exporter_ip,
exporter_port, exporter_certificate_ssl,
exporter_key_ssl)).start()
 syslog.syslog(syslog.LOG_WARNING,
                        f"py-exporter is
started and available on
https://{exporter_ip}:9101/metrics")
                while True:
                    try:
                        with
ThreadPoolExecutor() as executor:
executor.submit(cert_executor)
except Exception as error:
syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with executors:
{error}.")
            else:
syslog.syslog(syslog.LOG_CRIT,
                        f"py-exporter start is
failed. Certificate:
{exporter_certificate_ssl} or key:
{exporter_key_ssl} is not exists.")
                exit(1)
        elif argv[1] == "stop":

syslog.syslog(syslog.LOG_WARNING,
f"Stopping py-exporter.")
            exit(0)
        else:
            print(f"{argv[1]} is not an
available argument. Try start or stop.")
            exit(1)
    else:
        print("Argument is not found.
Please enter an argument. Available
arguments are: start, stop.")
        exit(1)
```

Додаток В

Код python скрипта для моніторингу з’єднання із OpenVPN сервером

```
#!/usr/bin/env python3
import ssl
import time
import syslog
import socket
import logging
import threading
import subprocess
from sys import argv
from flask import Flask
from flask_httpauth import
HTTPBasicAuth
from concurrent.futures import
ThreadPoolExecutor
from prometheus_client import
make_wsgi_app, Gauge
exporter_port = 9101
exporter_ip = "5.61.32.169"
exporter_login = "py-exporter"
exporter_password =
"DC6xFm3bat926mnp9J82KnMHA6j6
MCa9"
exporter_certificate_ssl =
"/etc/py-exporter/crt/exporter.crt"
exporter_key_ssl =
"/etc/py-exporter/crt/exporter.key"
check_ca_paths =
[exporter_certificate_ssl]
openvpn_config_file =
["/etc/py-exporter/configs/client-promet
heus.ovpn"]
openvpn_credential_file =
"/etc/py-exporter/configs/credential"
openvpn_constant_file =
"/etc/py-exporter/configs/constants"
py_metrics =
Gauge('py_check_ssl_ca_ovpn_expiry',
'Custom metric on python.', ['ip',
'instance', 'ca'])
py_metrics_openvpn_connection =
Gauge('py_check_ovpn_connection',
'Custom metric on python.',
                                ['ip', 'instance',
'server_name', 'server_ip', 'server_port'])
app = Flask(__name__)
auth = HTTPBasicAuth()
# Disables all log info from Flask
app.debug = False
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)
app.logger.setLevel(logging.ERROR)
@auth.verify_password
def verify_password(username,
password):
    if username == exporter_login and
password == exporter_password:
        #
syslog.syslog(syslog.LOG_WARNING,
f"User {username} auth accept.")
        return True
    if username == "":
        return False
syslog.syslog(syslog.LOG_WARNING,
f"User {username} auth failed.")
    return False
@app.route('/metrics')
@auth.login_required
def metrics():
    return make_wsgi_app()
def run_server(ip, port, ssl_cert,
ssl_key):
    context =
ssl.SSLContext(ssl.PROTOCOL_TLSv1
_2)
```

```
    context.load_cert_chain(ssl_cert,
ssl_key)
    app.run(host=ip, port=port,
ssl_context=context)
def servers_constant(constants_file,
config):
    try:
        if (check_path(constants_file)) and
(check_path(config)):
            with open(constants_file, "r") as
file:
                if file.readable():
                    file = eval(file.read())
                    server_name = config
                    server_name =
server_name.replace("/etc/py-exporter/c
onfigs/", "").replace(".ovpn", "")
                    if server_name in file:
                        server =
file[server_name]['server']
                        ip =
file[server_name]['ip']
                        port =
file[server_name]['port']
                        return server, ip, port
        else:
syslog.syslog(syslog.LOG_WARNING,
                      f"py-exporter has error
with openvpn server constants:
{constants_file} or {config} is not
found.")
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with openvpn
server constants: {error}")
def check_openvpn_connection(config,
credential_file):
    command = f"openvpn --config
{config} --auth-user-pass
{credential_file} --connect-retry 1
--connect-retry-max 1 --route-nopull"
    try:
        if check_path(config) and
check_path(credential_file):
            process =
subprocess.Popen(command,
shell=True, stdout=subprocess
stderr=subprocess.PIPE)
            output, _ =
process.communicate(timeout=30)
            if "auth_failed" in
output.decode().lower():
syslog.syslog(syslog.LOG_CRIT,
                          f"py-exporter has
error: connection to openvpn server
return - auth failed")
                raise
subprocess.CalledProcessError(returnco
de=1, cmd=command)
            elif "initialization sequence
completed" in output.decode().lower():
                raise
subprocess.TimeoutExpired(cmd=comm
and, timeout=30)
            else:
                raise
subprocess.CalledProcessError(returnco
de=1, cmd=command)
    except subprocess.CalledProcessError
as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with openvpn
proccess: {error}")
        subprocess.run(f'pkill -f
"{command}"', shell=True)
        return 0
    except subprocess.TimeoutExpired:
        subprocess.run(f'pkill -f
"{command}"', shell=True)
        return 1
    except Exception as error:
```

```
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with openvpn
connection check: {error}")
        subprocess.run(f'pkill -f
"{command}"', shell=True)
        return 0
def check_path(path):
    try:
        subprocess.check_output(['ls',
path], stderr=subprocess.DEVNULL)
        return True
    except subprocess.CalledProcessError
as error:
        # syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with cert path:
{error}")
        return False
def check_ca_days(ca_path):
    command = "echo $(($(date -d
\"$(openssl x509 -enddate -noout -in
path | sed 's/notAfter=//')\" +%s) - $(date
+%s)) / 86400))"
    try:
        days =
subprocess.run(command.replace("path"
, ca_path), shell=True,
capture_output=True, text=True)
        return days.stdout
    except subprocess.CalledProcessError
as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error: {error}.")
def openvpn_executor():
    try:
        with
ThreadPoolExecutor(max_workers=20)
as executor:
            for config in
openvpn_config_file:
                if check_path(config):
                    server, ip, port =
servers_constant(openvpn_constant_file,
config)
executor.submit(openvpn_collect_metric
s, server, ip, port, config)
                else:
syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error: {config} is not
exists.")
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with openvpn
executor: {error}.")
def cert_executor():
    try:
        with ThreadPoolExecutor() as
executor:
            for path in check_ca_paths:
                if check_path(path):
py_metrics.labels(ip=exporter_ip,
instance=socket.gethostname(),
ca=path).set(0)

executor.submit(cert_collect_metrics,
path)
                else:
syslog.syslog(syslog.LOG_WARNING,
f"Skipping path {path}. Maybe cert is
not exists.")
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with cert
executor: {error}.")
def
openvpn_collect_metrics(vpn_server_na
me, vpn_server_ip, vpn_server_port,
vpn_server_config):
    try:
py_metrics_openvpn_connection.labels(
ip=exporter_ip,
instance=socket.gethostname(),
```

```
server_name=vpn_server_name,
server_ip=vpn_server_ip,
server_port=vpn_server_port).set(
int(check_openvpn_connection(vpn_ser
ver_config, openvpn_credential_file)))
        time.sleep(60)
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with openvpn
metrics: {error}.")
def cert_collect_metrics(path):
    try:
        py_metrics.labels(ip=exporter_ip,
instance=socket.gethostname(),
ca=path).set(int(check_ca_days(path)))
        time.sleep(120)
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with metrics:
{error}.")
if __name__ == '__main__':
    if len(argv) > 1:
        if argv[1] == "start":
 syslog.syslog(syslog.LOG_WARNING,
f"Starting py-exporter.")
            if
check_path(exporter_certificate_ssl) and
check_path(exporter_key_ssl):
threading.Thread(target=run_server,
                             args=(exporter_ip,
exporter_port, exporter_certificate_ssl,
exporter_key_ssl)).start()
syslog.syslog(syslog.LOG_WARNING,
                          f"py-exporter is
started and available on
https://{exporter_ip}:9101/metrics")
                while True:
                    try:
                        with
ThreadPoolExecutor() as executor:
executor.submit(cert_executor)
executor.submit(openvpn_executor)
                    except Exception as error:
syslog.syslog(syslog.LOG_CRIT,
f"py-exporter has error with executors:
{error}.")
            else:
 syslog.syslog(syslog.LOG_CRIT,
                          f"py-exporter start is
failed. Certificate:
{exporter_certificate_ssl} or key:
{exporter_key_ssl} is not exists.")
                exit(1)
        elif argv[1] == "stop":
syslog.syslog(syslog.LOG_WARNING,
f"Stopping py-exporter.")
            exit(0)
        else:
            print(f"{argv[1]} is not an
available argument. Try start or stop.")
            exit(1)
    else:
        print("Argument is not found.
Please enter an argument. Available
arguments are: start, stop.")
        exit(1)
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії

Кваліфікаційна робота

ВПРОВАДЖЕННЯ МОНІТОРИНГУ СЕРВЕРА OPENVPN ЧЕРЕЗ ІНТЕГРАЦІЮ ІЗ СИСТЕМОЮ PROMETHEUS ТА PYTHON

Науковий керівник:
Лемешко Андрій Вікторович
Роботу виконала:
студентка групи КСДМ-61
Бриксіна Марія Дмитріївна

ОБ'ЄКТ, ПРЕДМЕТ ДОСЛІДЖЕННЯ ТА МЕТА РОБОТИ

Об'єкт дослідження: сервер OpenVPN.
Предмет дослідження: система моніторингу Prometheus та Python.
Мета роботи: впровадження моніторингу сервера OpenVPN за допомогою Prometheus для підвищення швидкості реакції на виникаючі інциденти на сервері.

АКТУАЛЬНІСТЬ ТЕМИ

Актуальність роботи полягає в можливості використання в будь-якій сфері IT, де використовується серверне обладнання. Всі пакети в роботі являють собою opensource і є безкоштовними.
За рахунок opensource є можливість модифіковувати все програмне забезпечення в залежності від потреби.
Дана робота пропонує рішення для зменшення часу реагування та можливість завчасно підготуватись до інцидентів пов’язаних із OpenVPN.

ВСТАНОВЛЕННЯ ПАКЕТІВ

На даному слайді представлені команди для встановлення пакетів для сервера OpenVPN та Prometheus.Всі необхідні пакети зазвичай є в стандартних репозиторіях і тільки у випадку з Grafana необхідно додавати репозиторій її розробників.

Для сервера OpenVPN:
apt install -y openvpn prometheus-node-exporter iptables iptables-persistent

Для сервера Prometheus:
apt install -y prometheus prometheus-alertmanager prometheus-node-exporter iptables iptables-persistent

Для встановлення Grafana:
mkdir -p /etc/apt/keyrings/ && wget -q -O https://apt.grafana.com/gpg.key | gpg --dearmor | tee etc/apt/keyrings/grafana.gpg && echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | tee -a /etc/apt/sources.list.d/grafana.list && apt update && apt install grafana -y

ПРАВИЛА FIREWALL

Для можливості підключення користувачів до OpenVPN сервера та отримання метрик сервером Prometheus необхідно додати правила на сервері OpenVPN. Одне правило для можливості підключення клієнтів до OpenVPN сервера, інше — для можливості підключення сервера Prometheus до експортерів.

Після додавання зберігаємо правила командою знизу, щоб вони відновлювались після рестарту операційної системи.

```
iptables -A INPUT -p udp -m udp –dport 2294 -m comment –comment “openvpn-server”-j ACCEPT
iptables -A INPUT -s 5.61.32.169/32 -p tcp -m multiport –dports 9100, 9101 -m comment –comment “exporters” -j ACCEPT
netfilter-persistent save
```


ПРОТОКОЛ OPENVPN

OpenVPN є opensource рішенням для створення VPN серверу, що забезпечує безпечне та шифроване з'єднання з Інтернетом за допомогою спеціального протоколу безпеки, який базується на використанні SSL/TLS для обміну ключами.

На скріншотах представлено зміст файлів конфігурації сервера (зверху) та клієнта OpenVPN (внизу).

```
root@vps-43966:/etc/openvpn# cat server.conf
port 2294
proto udp
dev tun
user nobody
group nogroup
duplicate-cn
persist-key
persist-tun
keepalive 10 120
topology subnet
server 10.8.0.0 255.255.255.0
status /var/log/openvpn/status.log
verb 3
push "dhcp-option DNS 1.1.1.1"
push "dhcp-option DNS 8.8.8.8"
push "redirect-gateway def1 bypass-dhcp"
script-security 2
auth-user-pass-verify /etc/openvpn/verify-user.sh via-file
username-as-common-name
auth SHA256
cipher AES-256-CBC
tls-server
```

```
root@vpntermde1:~# cat /etc/py-exporter/configs/client-prometheus.ovpn
client
proto udp
remote 185.25.116.46 2294
dev tun
nobind
persist-key
persist-tun
topology subnet
auth-nocache
auth-user-pass
verb 4
cipher AES-256-CBC
```

ЛОКАЛЬНА АУТЕНТИФІКАЦІЯ OPENVPN

```
#!/bin/bash
USERS=`cat /etc/openvpn/usrps`
vpn_verify() {
if [ ! $1 ] || [ ! $2 ]; then
exit 1
fi
for i in $USERS; do
if [ "$i" = "$1:$2" ]; then
echo `date` auth LOGIN: $1 >> /var/log/openvpn/openvpn-auth.log
exit 0
fi
done
echo `date` auth WRONG: $1 >> /var/log/openvpn/openvpn-auth.log
}
if [ ! $1 ] || [ ! -e $1 ]; then
exit 1
fi
vpn_verify `cat $1`
exit 1
```

```
root@vps-43966:/etc/openvpn# cat usrps
bryksina:+GixnJeAeQly5cfnAAywb2DiR/bLvrQ0+eHVdcX5FfU=
```

В даній роботі на сервері OpenVPN була реалізована локальна аутентифікація по логіну і паролю за допомогою bash-скрипта для підвищення рівня безпеки користувачів.
Даний скрипт парсить файл usrps з логіном і паролем та повертає 0, що означає вдалу аутентифікацію або 1, якщо дані користувача не відповідають даним у файлі.

7

СИСТЕМА PROMETHEUS

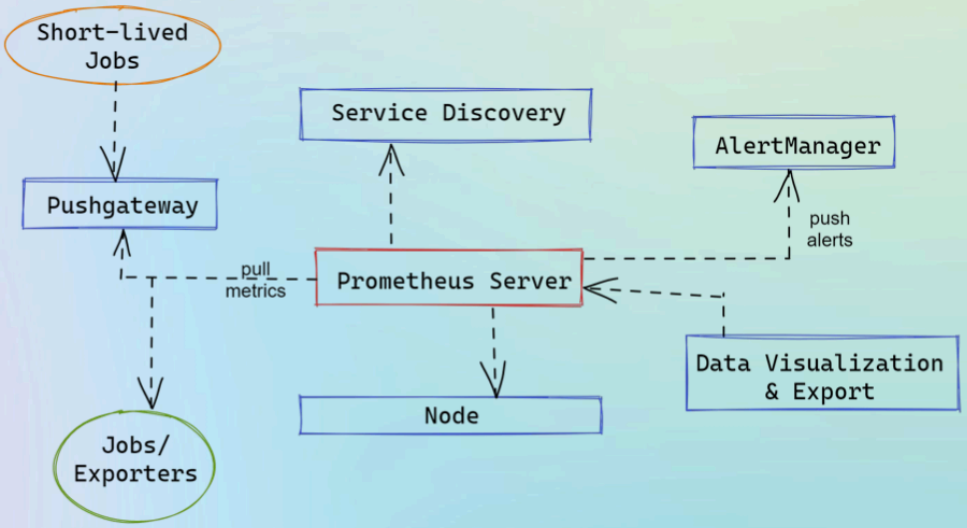
Одна з передових систем моніторингу, призначена для збору та аналізу метрик. Дозволяє виявляти проблеми в реальному часі та сповіщати відповідальних осіб про можливі проблеми в мережі.
Кожен моніторинговий об'єкт, як правило, є експортером, який використовує HTTP/HTTPS для надання своїх метрик Prometheus.

Приведено скріншот стандартної конфігурації Prometheus сервера. В ньому приведено приклад підключення alertmanager та файлів з правилами для нього (metrics-from-*).

```
global:
  scrape_interval:     45s
  scrape_timeout:      45s
  evaluation_interval: 35s
alerting:
  alertmanagers:
  - follow_redirects: true
    enable_http2: true
    scheme: http
    timeout: 10s
    api_version: v2
  - static_configs:
    - targets: ['localhost:9093']
rule_files:
   - "metrics-from-node-exporter.yml"
   - "metrics-from-py-exporter.yml"
scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    scrape_timeout: 5s
    static_configs:
      - targets: ['localhost:9090']
```

8

АРХІТЕКТУРА PROMETHEUS



9

ЕКСПОРТЕРИ PROMETHEUS

```
- job_name: 'node-exporters'
  scrape_timeout: 60s
  scrape_interval: 80s
  scheme: https
  tls_config:
    insecure_skip_verify: true
  basic_auth:
    username: prometheus
    password: 0O2g6YUktgSBE5Uo5wg80mZ4eBs=
  static_configs:
    - targets:
      - "5.61.32.169:9100@prometheus-server"
      - "185.25.116.46:9100@openvpn-server"
  relabel_configs:
    - source_labels: [ __address__ ]
      regex: '.*@(.*)'
      replacement: $1
      target_label: instance
    - source_labels: [ __address__ ]
      regex: '(.*):.*'
      replacement: $1
      target_label: IP
    - source_labels: [ __address__ ]
      regex: '(.*)@.*'
      replacement: $1
      target_label: __address__
```

Експортер - дозволяє збирати метрики з операційної системи та апаратного забезпечення на рівні вузла у мережі, основна функція - використовувати HTTP/HTTPS для надання своїх метрик серверу Prometheus.

```
- job_name: 'py-exporter'
  scrape_timeout: 30s
  scrape_interval: 40s
  scheme: https
  metrics_path: /metrics
  tls_config:
    insecure_skip_verify: true
  basic_auth:
    username: py-exporter
    password: DC6xFm3bat926mnp9J82KnMHA6j6MCa9
  static_configs:
    - targets:
      - "5.61.32.169:9101@prometheus-server"
      - "185.25.116.46:9101@openvpn-server"
  relabel_configs:
    - source_labels: [ __address__ ]
      regex: '.*@(.*)'
      replacement: $1
      target_label: instance
    - source_labels: [ __address__ ]
      regex: '(.*):.*'
      replacement: $1
      target_label: IP
    - source_labels: [ __address__ ]
      regex: '(.*)@.*'
      replacement: $1
      target_label: __address__
```

В даній кваліфікаційній роботі використано node-exporter для моніторингу CPU, RAM, FS (File System) сервера.
А також py-exporter , котрий написаний на python для реалізації моніторингу в даній роботі строку дії сертифікатів OpenVPN та можливості підключення до сервера OpenVPN.

10

ОСНОВНІ ФУНКЦІЇ СКРИПТА PYTHON

```python
def check_openvpn_connection(config, credential_file):
    command = f"openvpn --config {config} --auth-user-pass {credential_file} --connect-retry 1 --connect-retry-max 1 --route-nopull"
    try:
        if check_path(config) and check_path(credential_file):
            process = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output, _ = process.communicate(timeout=30)
            if "auth_failed" in output.decode().lower():
                syslog.syslog(syslog.LOG_CRIT,
                              f"py-exporter has error: connection to openvpn server return - auth failed")
                raise subprocess.CalledProcessError(returncode=1, cmd=command)
            elif "initialization sequence completed" in output.decode().lower():
                raise subprocess.TimeoutExpired(cmd=command, timeout=30)
            else:
                raise subprocess.CalledProcessError(returncode=1, cmd=command)
    except subprocess.CalledProcessError as error:
        syslog.syslog(syslog.LOG_CRIT, f"py-exporter has error with openvpn proccess: {error}")
        subprocess.run(f'pkill -f "{command}"', shell=True)
        return 0
    except subprocess.TimeoutExpired:
        subprocess.run(f'pkill -f "{command}"', shell=True)
        return 1
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT, f"py-exporter has error with openvpn connection check: {error}")
        subprocess.run(f'pkill -f "{command}"', shell=True)
        return 0
```

```python
def openvpn_collect_metrics(vpn_server_name, vpn_server_ip, vpn_server_port, vpn_server_config):
    try:
        py_metrics_openvpn_connection.labels(ip=exporter_ip, instance=socket.gethostname(),
                                             server_name=vpn_server_name, server_ip=vpn_server_ip,
                                             server_port=vpn_server_port).set(
            int(check_openvpn_connection(vpn_server_config, openvpn_credential_file)))
        time.sleep(60)
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT, f"py-exporter has error with openvpn metrics: {error}.")


def cert_collect_metrics(path):
    try:
        py_metrics.labels(ip=exporter_ip, instance=socket.gethostname(), ca=path).set(int(check_ca_days(path)))
        time.sleep(120)
    except Exception as error:
        syslog.syslog(syslog.LOG_CRIT, f"py-exporter has error with metrics: {error}.")
```

```python
def check_ca_days(ca_path):
    command = "echo $(($($(date -d \"$(openssl x509 -enddate -noout -in path | sed 's/notAfter=//')\" +%s) - $(date +%s)) / 86400))"
    try:
        days = subprocess.run(command.replace("path", ca_path), shell=True, capture_output=True, text=True)
        return days.stdout
    except subprocess.CalledProcessError as error:
        syslog.syslog(syslog.LOG_CRIT, f"py-exporter has error: {error}.")
```


TLS/SSL ТА BASIC AUTH ДЛЯ PY-EXPORTER

```python
from flask import Flask
from flask_httpauth import HTTPBasicAuth

exporter_port = 9101
exporter_ip = "185.25.116.46"
exporter_login = "py-exporter"
exporter_password = "ptC53uYS10cNkvP5f6h5T58YHU8z88vCi"
exporter_certificate_ssl = "/etc/py-exporter/crt/exporter.crt"
exporter_key_ssl = "/etc/py-exporter/crt/exporter.key"

app = Flask(__name__)
auth = HTTPBasicAuth()
```

Використовуючи фреймворк Flask було перенесено py-exporter з http на https та додано базову аутентифікацію.

```python
@auth.verify_password
def verify_password(username, password):
    if username == exporter_login and password == exporter_password:
        return True
    if username == "":
        return False
    syslog.syslog(syslog.LOG_WARNING, f"User {username} auth failed.")
    return False


@app.route('/metrics')
@auth.login_required
def metrics():
    return make_wsgi_app()


def run_server(ip, port, ssl_cert, ssl_key):
    context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    context.load_cert_chain(ssl_cert, ssl_key)
    app.run(host=ip, port=port, ssl_context=context)
```

ALERTMANAGER TA GRAFANA

Alertmanager використовується для обробки та управління алертами, які генеруються системою моніторингу. Він дозволяє налаштовувати правила для спрацьовування алертів, групувати їх та відправляти на різні канали оповіщення, такі як електронна пошта, Slack або інші інтеграції. В даній роботі було реалізовано з Telegram.

```
global:
  resolve_timeout: 5m

route:
  group_by: ['alertname', 'job', 'service']
  group_wait: 10s
  group_interval: 5m
  repeat_interval: 3h
  receiver: 'telegram'
  routes:
  - match:
      severity: critical
    receiver: telegram

receivers:
- name: 'telegram'
  telegram_configs:
  - bot_token: '6276741927:AAEK6P5g-sk774BnzrSAphiHCtAVR35[redacted]'
    chat_id: 53560[redacted]
```

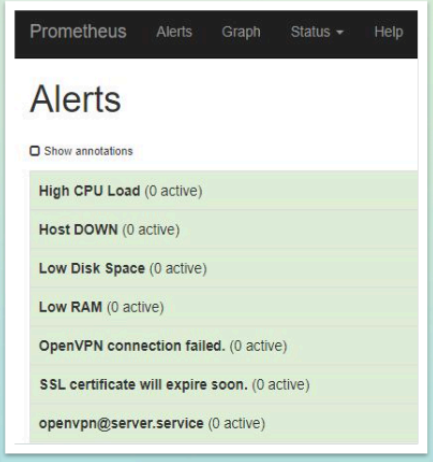
```
groups:
- name: py-exporter
  rules:
    - alert: OpenVPN connection failed.
      expr: py_check_ovpn_connection == 0
      for: 1s
      labels:
        severity: critical
      annotations:
        summary: "On instance {{ $labels.exported_instance }} has problem"
        description: "From instance {{ $labels.exported_instance }} with IP:({{
    - alert: SSL certificate will expire soon.
      expr: py_check_ssl_ca_ovpn_expiry < 30
      for: 1m
      labels:
        severity: "critical"
      annotations:
        summary: "On instance {{ $labels.exported_instance }} has problem."
        description: "Certificate: {{ $labels.ca }} on instance {{ $labels.exp
```

Grafana для візуалізації та аналізу даних з різних джерел, включаючи дані, які збираються системою моніторингу. Він надає гнучкі можливості для створення графіків, панелей та інших візуальних елементів, що допомагають адміністраторам та розробникам відстежувати та аналізувати параметри систем та додатків.

13

ПРАВИЛА ALERTMANAGER

```
root@vpntermde1:/etc/prometheus# cat metrics-from-node-exporter.yml
groups:
- name: node-exporter
  rules:
  - alert: 'Host DOWN'
    expr: up == 0
    for: 90s
    labels:
      severity: critical
    annotations:
      description: "Instance {{ $labels.instance }} with IP:({{{ $labels.IP }}}) has been down
        for more than 90 seconds."
      summary: Instance {{ $labels.instance }} is DOWN
  - alert: Low RAM
    expr: node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 < 10
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: "On instance {{ $labels.instance }} out of RAM memory"
      description: "RAM memory on instance {{ $labels.instance }} with IP:({{{ $labels.IP }}}) is filling up (< 10% left). Free RAM level in right now
  - alert: openvpn@server.service
    expr: node_systemd_unit_state{name="openvpn@server.service",state="active"} == 0
    for: 1s
    labels:
      severity: critical
    annotations:
      summary: "Instance {{ $labels.instance }} is down"
      description: "{{ $labels.name }} on {{ $labels.instance }} of job {{ $labels.job }} is down."
  - alert: Low Disk Space
    expr: (node_filesystem_avail_bytes * 100) / node_filesystem_size_bytes < 10 and ON (instance, device, mountpoint) node_filesystem_readonly == 0
    for: 2m
    labels:
      severity: critical
    annotations:
      summary: "On instance {{ $labels.instance }} out of disk space"
      description: "Disk space device ({{ $labels.device }}) on instace {{ $labels.instance }} with IP:({{{ $labels.IP }}}) is almost full (< 10% left)
  - alert: High CPU Load
    expr: 100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[2m])) * 100) > 80
    for: 0m
    labels:
      severity: warning
    annotations:
      summary: "On instance {{ $labels.instance }} high CPU load"
      description: "Load CPU on instance {{ $labels.instance }} is (> 80%). CPU load in right now is: {{ $value }} %."
```



14

РОБОТА СИСТЕМИ PROMETHEUS

```
root@vps-43966:/etc/openvpn/others# bash openssl-ca.sh
###############################################################################################################################################
What need do?
1.) Create new client certificate and key.
2.) Revoke certificate.
3.) Verify certificate.
4.) Create server, client conf file for OpenVPN.
5.) Exit.
###############################################################################################################################################
Enter number 1 - 5: 1
Create new CA certificates or use already created? [new/old] (auto: old)
###############################################################################################################################################
Name for client: client-5-days
Which size certificate need in bits? |1024|2048|4096|8192| 1024
How many days the certificate must be valid? 5
Generating a RSA private key
.....................+++++
.....................+++++
writing new private key to 'client-5-days.key'
-----
Signature ok
subject=C = IT, ST = IT, L = IT, O = IT, OU = IT, CN = client-5-days
Getting CA Private Key
###############################################################################################################################################
Verify:
client-5-days.crt: OK
Enddate:
notAfter=Nov 29 05:18:22 2023 GMT
###############################################################################################################################################
root@vps-43966:/etc/openvpn/others# ls
ca.crt  ca.key  cert        client-5-days.key      client-prometheus.key  crl     dh.pem        server-bryksina.crt  tls-crypt.key
ca.crt  ca.srl  client-5-days.crt  client-prometheus.crt  client-prometheus.ovpn  db.txt  openssl-ca.sh  server-bryksina.key
root@vps-43966:/etc/openvpn/others# cat client-5-days.crt > client-prometheus.crt
root@vps-43966:/etc/openvpn/others# cat ../usrps
bryksina:rGixnJe4eQlySCfnAAyw02dlR/bLvm20+eHVdCX5F+U
```

Для тесту системи було змінено пароль для користувача bryksina на OpenVPN сервері та замінено сертифікат зі строком дії на 5 днів.

Alerts Firing:
Labels:
- alertname = OpenVPN connection failed.
- IP = 5.61.32.169
- exported_instance = vpntermde1
- instance = prometheus-server
- ip = 5.61.32.169
- job = py-exporter
- server_ip = 185.25.116.46
- server_name = bryksina
- server_port = 2294
- severity = critical
Annotations:
- description = From instance vpntermde1 with IP:(5.61.32.169) connection to bryksina with IP:(185.25.116.46:2294) is failed.
- summary = On instance vpntermde1 has problem
Source: http://vpntermde1:9090/graph?g0.expr=py_check_ovpn_connection+%3D%3D+0&g0.tab=1

Alerts Firing:
Labels:
- alertname = SSL certificate will expire soon.
- IP = 185.25.116.46
- ca = /etc/openvpn/others/client-prometheus.crt
- exported_instance = vps-43966
- instance = openvpn-server
- ip = 185.25.116.46
- job = py-exporter
- severity = critical
Annotations:
- description = Certificate: /etc/openvpn/others/client-prometheus.crt on instance vps-43966 with IP:(185.25.116.46) will expire in 4 days.
- summary = On instance vps-43966 has problem.
Source: http://vpntermde1:9090/graph?g0.expr=py_check_ssl_ca_ovpn_expiry+%3C+30&g0.tab=1

Home › Dashboards › Prometheus AlertManager

All Alerts Info

Time	SeverityValue	description	summary	IP	alertname	exported_instance	instance	ip	job	server_ip	server_name	server_port	severity	ca
2023-11-24 07:22:41	1.00	Certificate: /etc/openvpn/others/client-prometheus.crt on instance vps-43966 with IP:(185.25.116.46) will expire in 4 days.	On instance vps-43966 has problem.	185.25.116.46	SSL certificate will expire soon.	vps-43966	openvpn-server	185.25.116.46	py-exporter	0.00	0.00	0.00	critical	/etc/openvpn/others/client-prometheus.crt
2023-11-24 07:15:41	1.00	From instance vpntermde1 with IP:(5.61.32.169) connection to bryksina with IP:(185.25.116.46:2294) is failed.	On instance vpntermde1 has problem	5.61.32.169	OpenVPN connection failed.	vpntermde1	prometheus-server	5.61.32.169	py-exporter	185.25.116.46	bryksina	2.29 K	critical	0.00

15

ВИСНОВОК

В ході роботи були проаналізовані популярні протоколи віртуальних приватних мереж, систем моніторингу, unix-based ОС, основи роботи системи моніторингу Prometheus та її компонентів. Встановлено, що протокол OpenVPN являється зручним рішенням при реалізації VPN-шлюзу для доступа до приватної мережі чи інтернету. Сервер Prometheus доволі легко інтегрується в мережу. Використання систем моніторингу дозволяє скоротити час необхідний для реагування на виникаючі інциденти. Мову програмування Python є можливість використовувати для реалізації моніторингових рішень будь якого маштабу.

16