Титульний лист

# **Зміст**

[**Зміст** 2](#_Toc127366896)

[**Вступ** 4](#_Toc127366897)

[Актуальність теми. 4](#_Toc127366898)

[Мета і завдання дослідження 4](#_Toc127366899)

[Об’єкт дослідження 4](#_Toc127366900)

[Предмет дослідження 4](#_Toc127366901)

[Методи дослідження 4](#_Toc127366902)

[**1. Інструменти які використані в роботі** 5](#_Toc127366903)

[**1.1 Zabbix, інструмент для системного аналізу** 5](#_Toc127366904)

[1.1.1 Історія та походження Zabbix 5](#_Toc127366905)

[1.1.2 Що таке Zabbix? 5](#_Toc127366906)

[1.1.3 Особливості Zabbix: 6](#_Toc127366907)

[1.1.4 Навіщо використовувати Zabbix? 6](#_Toc127366908)

[1.1.5 Переваги Zabbix 7](#_Toc127366909)

[1.1.6 Недоліки Zabbix 7](#_Toc127366910)

[1.1.7 Альтернативи систем аналізу Zabbix 7](#_Toc127366911)

[1.1.8 Підведення підсумків 7](#_Toc127366912)

[**1.2 Аналіз ефективності використання docker для системи Zabbix** 8](#_Toc127366913)

[1.2.1 Введення 8](#_Toc127366914)

[1.2.2 Докер 9](#_Toc127366915)

[1.2.3 Docker Inside 9](#_Toc127366916)

[1.2.4 Аналіз віртуальна машин проти Docker контейнерів 11](#_Toc127366917)

[1.2.5 Переваги Docker Container 14](#_Toc127366918)

[1.2.6 Недоліки Docker Container 16](#_Toc127366919)

[1.2.7 Продуктивність Docker 17](#_Toc127366920)

[1.2.8 Docker проти інших контейнерних технологій 18](#_Toc127366921)

[1.2.9 Вибір між ВМ та контейнерів 24](#_Toc127366922)

[1.2.10 Підведення підсумків 25](#_Toc127366923)

[**1.3 Віртуалізація серверів** 26](#_Toc127366924)

[1.3.1 Термінологія 26](#_Toc127366925)

[1.3.2 Огляд функцій 27](#_Toc127366926)

[1.3.3 Підведення підсумків 32](#_Toc127366927)

[**2. Підготовка для аналізу ефективності Zabbix** 33](#_Toc127366928)

[**2.1 Загальна архітектура для проведення аналізу** 33](#_Toc127366929)

[2.1.1 Сервер А 33](#_Toc127366930)

[2.1.2 Сервер Б 33](#_Toc127366931)

[2.1.3 Підведення підсумків 34](#_Toc127366932)

[**2.2 Конфігурація серверів** 34](#_Toc127366933)

[2.2.1 Конфігурація серверу А 34](#_Toc127366934)

[2.2.2 Конфігурація серверу Б 35](#_Toc127366935)

[2.2.3 Підведення підсумків 36](#_Toc127366936)

[**3. Аналіз ефективності Zabbix** 37](#_Toc127366937)

[**Джерела** 40](#_Toc127366938)

# **Вступ**

## Актуальність теми.

Дослідження інформаційних систем була не легка задача, але з часом з’являлось все більше і більше інструментів, які облегшують цей процес. Аналіз систем є для кожної компанії невід’ємною складовою, але у кожної компанії свої підходи та реалізація. Великий вибір інструментів аналізу систем на ринку може призвести до невизначеності, оскільки деякі з них можуть бути складні в розумінні та потребують багато часу на налаштування для першого аналізу.

## Мета і завдання дослідження

Дослідити ефективність використання інструменту для аналізу систем «Zabbix», з використанням докеру.

## Об’єкт дослідження

Два сервери, з яких буде збиратись дані моніторингу для аналізу систем за допомогою «Zabbix».

## Предмет дослідження

Аналітичний інструмент для дослідження та моніторингу систем «Zabbix», який повинен вхідні дані (напруги на сервер, напруги на мережу) аналізувати та вистроювати у вигляді графіків звіти.

## Методи дослідження

Після розгортання на серверах інструментів для аналізу систем, ми візуалізуймо звіти у вигляді веб-платформи. Всі звіти будуть відображатись на веб-сторінці. Вибір аналізу – це шаблони у Zabbix. На прикладі декількох шаблонів проаналізуємо ефективність інструменту.

# **1. Інструменти які використані в роботі**

## **1.1 Zabbix, інструмент для системного аналізу**

### 1.1.1 Історія та походження Zabbix

Zabbix був створений Олексієм Владишевим і зараз активно розвивається та підтримується Zabbix SIA. Zabbix — це рішення розподіленого моніторингу корпоративного рівня з відкритим кодом. Zabbix — це програмне забезпечення, яке контролює працездатність і цілісність багатьох параметрів мережі та серверів. Zabbix використовує гнучкий механізм сповіщень, який дозволяє користувачам налаштовувати сповіщення електронною поштою практично для будь-якої події. Це дозволяє швидко реагувати на проблеми сервера. Zabbix надає чудові функції звітності та візуалізації даних на основі архівних даних. Це робить Zabbix ідеальним для планування потужності.

### 1.1.2 Що таке Zabbix?

Zabbix визначається як інструмент моніторингу з відкритим кодом, який використовується для моніторингу серверів, мереж, ІТ-компонентів, хмарних служб і віртуальних машин. Інструмент моніторингу Zabbix використовується для забезпечення метрик моніторингу та відстеження використання мережі, споживання дискового простору та навантаження ЦП. Інструмент підтримує різні операційні системи, такі як Mac OS, Solaris, Linux та багато інших. Інструмент використовує окрему базу даних для зберігання даних і моніторингу програм. Інструмент моніторингу Zabbix розроблено мовою програмування C, а для веб-інтерфейсу використовується мова PHP.

### 1.1.3 Особливості Zabbix:

1. Візуалізація
2. Zabbix API
3. Повідомлення та виправлення
4. Легке розгортання
5. Безпека та аутентифікація
6. Метричний збірник
7. Виявлення проблеми
8. Розподілений моніторинг

### 1.1.4 Навіщо використовувати Zabbix?

1. Рішення з відкритим кодом
2. Дуже легке налаштування
3. Дуже проста конфігурація
4. Сервісне дерево високого рівня
5. Вбудована процедура прибирання
6. Високоефективні агенти для платформ на основі UNIX і WIN32
7. Низька крива навчання
8. Високий ROI. Простої дуже дорогі.
9. Низька вартість володіння
10. Централізована система моніторингу. Уся інформація (конфігурація, дані про продуктивність) зберігається в реляційній базі даних
11. Підтримка SNMP (v1,v2). І відлов, і опитування.
12. Можливості візуалізації

### 1.1.5 Переваги Zabbix

1. Гарний веб-інтерфейс.
2. Моніторинг як серверних, так і мережевих пристроїв.
3. Гнучкість у децентралізації користувачів.
4. Легко керувати та налаштовувати: уся конфігурація Zabbix здійснюється через веб-інтерфейс.
5. Підтримує Linux, Solaris, FreeBSD…
6. Довірена автентифікація користувача.
7. Повідомте про проблеми електронною поштою та SMS.

### 1.1.6 Недоліки Zabbix

1. Zabbix повинен містити Amazon RDS.
2. UI/UX має бути більш помітним і детальним.
3. Zabbix має надати більше шаблонів для повного моніторингу.

### 1.1.7 Альтернативи систем аналізу Zabbix

1. Datadog.
2. AppDynamics.
3. Dynatrace
4. LogicMonitor.
5. Монітор мережі PRTG.
6. checkmk.
7. Microsoft System Center.

### 1.1.8 Підведення підсумків

В цьому розділі ми ознайомились з Zabbix, дізналися загальні відомості. Мета дипломної роботи, проаналізувати ефективність використання цього інструменту для системного аналізу.

## **1.2 Аналіз ефективності використання docker для системи Zabbix**

Docker надає деякі можливості, корисні для розробників і адміністраторів. Це відкрита платформа, яку можна використовувати для створення, розповсюдження та запуску додатків у портативному, легкому середовищі виконання та інструментів пакування, відомому як Docker Engine. Він також надає Docker Hub, який є хмарним сервісом для обміну програмами. Витрати можна зменшити, замінивши традиційну віртуальну машину докер-контейнером. Це чудово знижує витрати на перебудову платформи хмарної розробки.

### 1.2.1 Введення

Docker — це платформа з відкритим вихідним кодом, яка запускає програми та полегшує процес розробки та розповсюдження. Програми, вбудовані в докер, упаковані з усіма підтримуючими залежностями в стандартну форму,

яка називається контейнером. Ці контейнери продовжують працювати ізольовано поверх ядра операційної системи. Додатковий рівень абстракції може вплинути на продуктивність. Незважаючи на те, контейнерні технології існують уже понад 10 років, але докер, загалом нова надія, зараз є видатним серед найкращих інновацій, оскільки він супроводжує нові можливості, яких не мали попередні технології. Спочатку він дає можливість створювати та контролювати контейнери. Крім того, програми можуть бути легко упаковані розробником у легкі докер-контейнери. З цими віртуалізованими програмами можна легко працювати будь-де без будь-яких змін. Крім того, докер може передавати більше віртуальних ситуацій, ніж різні інновації, на одному обладнанні. Підсумовуючи, докер може легко координувати роботу зі сторонніми інструментами, які допомагають легко розгортати та керувати контейнерами докерів. Контейнери Docker можна легко розгорнути в хмарному середовищі. Ця дипломна робота є також оглядом технології докеру і аналізує її ефективність за допомогою систематичного огляду літератури.

### 1.2.2 Докер

Docker надає засоби для автоматизації програм, коли вони розгортаються в контейнерах. У середовищі, де програми віртуалізуються та виконуються, докер додає додатковий рівень механізму розгортання. Докер розроблений таким чином, щоб створити швидке та легке середовище, де можна ефективно запускати код, і, крім того, він надає додаткову можливість професійного робочого процесу для отримання коду з комп’ютера для тестування перед виробництвом. Докер дозволяє якомога швидше протестувати ваш код і розгорнути його у виробничому середовищі, а також докер надзвичайно простий. Звичайно, ви можете почати з докера з простою системою конфігурації, бінарного докера з ядром Linux.

### 1.2.3 Docker Inside

Існує чотири основні внутрішні компоненти Docker, включаючи Docker-клієнт і сервер, Docker-образи, Docker образи та Docker-контейнери. Ці компоненти будуть детально описані в наступних підрозділах.

Клієнт і сервер Docker

Docker можна пояснити як програму на основі клієнта та сервера, як показано на малюнку 3.2.1. Докер-сервер отримує запит від докер-клієнта, а потім обробляє його відповідним чином. Повний API RESTful (Representational

state transfer) і двійковий файл клієнта командного рядка постачається докером. Докер-демон/сервер і докер-клієнт можуть бути запущені на одній машині або локальний докерклієнт може бути підключений до віддаленого сервера або демона, який працює на іншій машині.

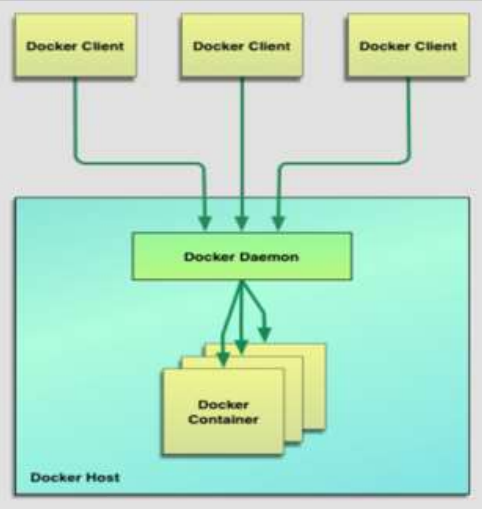


Рисунок 1.2.1 - Архітектура Docker

Docker образи

Існує два способи створення образу. Перший — створити зображення за допомогою шаблону, доступного лише для читання. Основою кожного образу є базове зображення. Образи операційної системи – це в основному базові

образи, наприклад Ubuntu 14.04 LTS або Fedora 20. Образи операційної системи створюють контейнер із можливістю повної запущеної ОС. Базове зображення також можна створити з нуля. Потрібні програми можна додати до базового образу, змінивши його, але необхідно створити новий образ. Процес створення нового створення образу називається «внесенням змін». Другий спосіб — створити докер-файл. Файл докерів містить список інструкцій, коли команда «Docker build» виконується з терміналу bash, вона

дотримується всіх інструкцій, наданих у файлі докерів, і створює образ. Це автоматизований спосіб побудови зображення.

Docker Registries

Образи докерів розміщуються в реєстрах докерів. Він працює відповідно до репозиторіїв вихідного коду, де зображення можна надсилати або витягувати з одного джерела. Існує два типи реєстрів: публічні та приватні. Docker Hub називається загальнодоступним реєстром, де кожен може завантажувати доступні зображення та надсилати власні, не створюючи зображення з нуля. Зображення можна поширювати в певну область (загальнодоступну чи приватну) за допомогою функції докер-хаба.

Контейнери Docker

Зображення докера створює контейнер докера. Контейнери містять весь набір, необхідний для програми, тому програму можна запускати ізольовано. Наприклад, припустимо, що є образ ОС Ubuntu із SQL SERVER, коли цей образ запускається за допомогою команди docker run, тоді буде створено контейнер з SQL сервером, який працюватиме в ОС Ubuntu.

### 1.2.4 Аналіз віртуальна машин проти Docker контейнерів

Віртуалізація — це стара концепція, яка використовувалася в хмарних обчисленнях після того, як IaaS було прийнято як вирішальний метод для створення системи, забезпечення ресурсами та багатокористування. Віртуалізовані ресурси відіграють основну роль у вирішенні проблем за допомогою основної техніки хмарних обчислень. На рисунку 1.2.2 показана архітектура віртуальної машини.

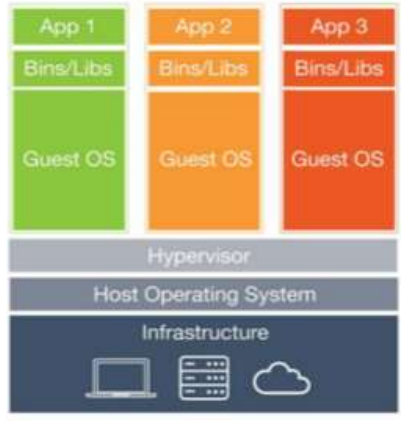


Рисунок 1.2.2 - Архітектура віртуальної машини

Гіпервізор лежить між хостовою та гостьовою операційними системами. Це віртуальна платформа, яка підтримує більше ніж одну операційну систему на сервері. Він працює між операційною системою та процесором. Віртуалізація ділить його на два сегменти: перший – паравіртуалізація, а другий – повна віртуалізація. На малюнку 1.2.3 зображено архітектуру Docker Container. Контейнерами Linux керує докер-інструмент, який використовується як метод віртуалізації на рівні операційної системи. На малюнку 1.2.3 показано, що в одному контрольному хості є багато контейнерів Linux, які є ізольованими. Такі ресурси, як мережа, пам’ять, процесор і блоковий ввід/вивід, виділяються ядром Linux, а також працює з контрольними групами без запуску машини віртуалізації.

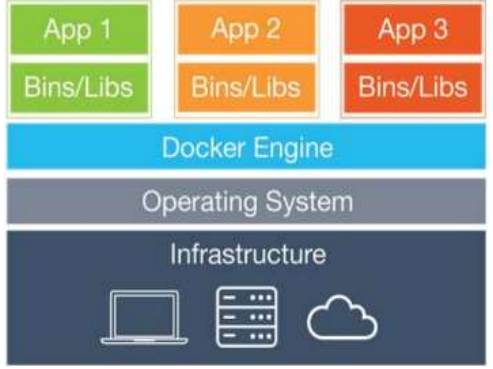


Рисунок 1.2.3 - Архітектура Docker Container

Відповідно до Waldspurger (2002), в контейнерах Linux архітектура має керувати центральним процесором і більш вміло розподіляти його ресурси. У будь-якому прикладі Hyper-V або VMWare через накладні витрати нелегко запустити більше десяти віртуальних машин. Значною мірою цю проблему вирішили контейнери. Контейнери використовують лише ті ресурси, які необхідні для служб або програм. Тому на слабко налаштованій машині можна виконати понад 50 запитів контейнерів. Наприклад, припустимо, що організація надає послуги безпеки електронної пошти. Основними функціями цих служб є перевірка електронної пошти на наявність вірусів, спаму та зловмисного програмного забезпечення. Крім того, вони можуть керувати пересиланням повідомлень до агента, журналів і звітів про збій доставки, якщо продукт встановлено в хмарі. Здебільшого в цих випадках не використовуються пов’язані залежності чи бібліотеки рівня ОС чи будь-які структури даних ядра. Тому варто контейнеризувати кожен компонент, помістивши їх у пісочницю за допомогою OpenVZ або Docker замість віртуальних машин. На багатьох підприємствах для тестування елементів використовуються віртуальні машини. У цьому процесі споживається багато ресурсів процесора та пам’яті. Тоді як контейнерна технологія дає гарантію своїм користувачам, що перевищення робочого навантаження не вплине на ефективність використання ресурсів. Контейнер займає менше часу для встановлення порівняно з віртуальними машинами, тому адаптивність контейнерів набагато вища, ніж віртуальних машин. Крім того, і Docker, і OpenVZ ретельно перевірялися з точки зору їхніх аспектів безпеки. Коли ізоляція знижується, це безпосередньо впливає на безпеку, яка також швидко знижується. Root користувачі Linux можуть легко отримати доступ до контейнерів, оскільки контейнери також використовують те саме ядро та операційну систему. Ізоляція докера не така сильна, як віртуальна машина, навіть якщо докер ізолює програму, яка працює в контейнері докерів, від основного хоста. Крім того, можливо, що деякі програми не зможуть працювати в контейнерній технології, і вони повинні працювати в іншій операційній системі.

### 1.2.5 Переваги Docker Container

За останні кілька років можна побачити попит і розвиток контейнерів Linux. Docker дуже швидко став популярним завдяки перевагам, які надає докер контейнер. Основними перевагами докера є швидкість, мобільність, масштабованість, швидка доставка та щільність.

Швидкість

Швидкість є однією з найбільш рекламованих переваг контейнерів. Коли висвітлюються переваги використання докерів, було б неймовірно не згадати про швидкість докерів у розмові (Chavis & Architect, 2015). Час, необхідний для створення контейнера, дуже швидкий, оскільки вони дійсно малі. Розробку, тестування та розгортання можна виконувати швидше, оскільки контейнери малі. Контейнери можуть бути відправлені на тестування після їх створення, а потім у виробниче середовище.

Портативність

Ті програми, які вбудовані в докер-контейнери, надзвичайно портативні. Ці портативні програми можна легко переміщати як єдиний елемент, а продуктивність залишається незмінною.

Масштабованість

Docker має можливість розгорнути його на кількох фізичних серверах, серверах даних і хмарних платформах. Його також можна запускати на будь-якій машині Linux. Контейнери можна легко перемістити з хмарного середовища на локальний хост, а звідти знову в хмару швидко. Регулювання можна легко зробити; шкала може бути просто налаштована користувачем відповідно до потреб.

Швидка доставка

Формат контейнерів Docker стандартизовано, тому програмістам не доводиться напружуватися над завданнями один одного. Відповідальністю адміністратора є розгортання та обслуговування сервера з контейнерами, тоді як відповідальністю програміста є догляд за програмами в контейнері докерів. Контейнери можуть працювати в будь якому середовищі, оскільки вони мають усі необхідні залежності, вбудовані в програми, і всі вони перевірені. Docker забезпечує надійне, послідовне та покращене середовище, тому результати можуть бути передбачуваними - досягається, коли коди переміщуються між системами розробки, тестування та виробництва (Chavis & Architect, 2015).

Щільність

Docker ефективніше використовує доступні ресурси, оскільки не використовує гіпервізор. Це є причиною того, що на одному хості можна запускати більше контейнерів порівняно з віртуальними машинами. Продуктивність Docker Containers вища через більшу щільність і відсутність накладних витрат ресурсів.

### 1.2.6 Недоліки Docker Container

* + Повна віртуалізація не забезпечується докером, оскільки вона залежить від ядра Linux, яке надається локальним хостом.
  + Наразі docker не працює на старих машинах. Він підтримує лише 64-розрядні локальні машини.
  + Повне віртуалізоване середовище має забезпечуватися контейнером докерів для машин Windows і Mac. Незважаючи на те, що boot2docker інструмент заповнює цю прогалину, але все ж слід перевірити, чи створює він перешкоди для прийняття користувачами цих систем, чи інтеграція та продуктивність з операційною системою хост-машини є адекватними.
  + Необхідно оцінити можливість проблем з безпекою. Створення довірчих двійкових файлів може бути простіше за допомогою цифрового підпису зображень докерів для майбутньої підтримки.
  + Важливо перевірити, чи викладацьке співтовариство чи науковий дослідник серйозно задумаються про прийняття докерів.

### 1.2.7 Продуктивність Docker

Microsoft використовували два сервери з однаковою конфігурацією в хмарному середовищі. Один сервер використовувався для докерів, а інший – для платформи Open Stack для KVM за допомогою інструменту віртуалізації. За словами Microsoft, ВМ працює самостійно. Цей фактор полегшує застосування та керування політикою мережі, безпеки, користувача та системи. Однак докер не містить гостьової операційної системи. Тому розповсюдження та збір зображень займає дуже мало часу. Час завантаження також дуже короткий. Це основні переваги використання Docker Cloud порівняно з VM Cloud. Scheepers (2014) порівнює технології віртуалізації LXC і Xen для порівняння деяких програм. Він пояснює, що Xen був би кращим вибором у сенсі рівномірного розподілу ресурсів, а продуктивність – ні. Однак LXC набагато кращий у сенсі отримання більшості апаратних ресурсів або для виконання менших ізольованих процесів. У приватних і точкових хмарах LXC є кращим варіантом. Microsoft оцінюють продуктивність трьох різних

середовищ: Native, Docker і KVM. Він уточнює, що контейнери та віртуальні машини є зрілими інноваціями, які отримали вигоду від останніх 10 років інкрементного обладнання та вдосконалення програм. Згідно з цим дослідженням, докер еквівалентний або перевершує виконання KVM для кожної ситуації, яку вони пробували. Їхні результати демонструють, що і KVM, і докер представляють несуттєві накладні витрати на процесор і пам’ять. Також було показано, що загальна продуктивність докера краща, ніж у локального хосту, оскільки програми виконувалися та відповідали швидше, ніж у локальному хості. Крім того, для виконання завдань у контейнері докерів використовувалося менше апаратних ресурсів. Docker — це дійсно вимоглива технологія майбутнього. Коли користувачі та розробники дізнаються більше про докер та його можливості, вони подумають про заміну традиційної віртуалізації на технологію докера. Docker надає багато простих і корисних функцій. Щоб отримати найкращу продуктивність і результати, настійно рекомендується змінити конфігурацію за замовчуванням. Контейнери забезпечують підвищену щільність, кращу продуктивність, масштабованість і зручність використання порівняно з традиційною віртуалізацією, оскільки контейнери розумно використовують свої ресурси, що зменшує ймовірність непотрібних накладних витрат. Контейнери кращі за продуктивністю, ніж віртуальні машини, оскільки контейнери потребують менше часу для запуску. Docker усунув найбільшу проблему «залежності». Тепер контейнери мають усі свої необхідні залежності, які допомагають належним чином створювати контейнери, і виконувати їх у будь-якому середовищі докерів. Додатковий шар ізоляції забезпечує контейнер, що підвищує безпеку контейнерів. Docker не такий небезпечний, як люди зазвичай думають, але він забезпечує повний захист.

### 1.2.8 Docker проти інших контейнерних технологій

У цьому розділі буде обговорено продуктивність віртуалізації програми та продуктивність контейнера докерів, а також буде порівняно та розглянуто оцінку інших технологій контейнеризації. В хмарі немає гостьової ОС докера, тому сховище та втрата ресурсів ЦП менші. Зображення не порушуються; час завантаження швидший, а час створення зображень короткий. Це переваги Docker Cloud порівняно з VM Cloud. Вони використовували два подібних сервера з однаковою конфігурацією в хмарному середовищі. Один сервер використовувався для докерів, а інший – для платформи Open Stack для KVM за допомогою засобу віртуалізації. В якості базової платформи використовувався Ubuntu Server. Щоб обчислити приблизний час завантаження, на кожному сервері було створено 20 зображень і перевірено час завантаження. На рисунку 1.2.4 показано, що час завантаження докера менший, ніж час завантаження KVM. Docker використовує хост-ОС, тоді як KVM використовує гостьову ОС. Таким чином, час завантаження докера коротший, ніж час завантаження KVM.

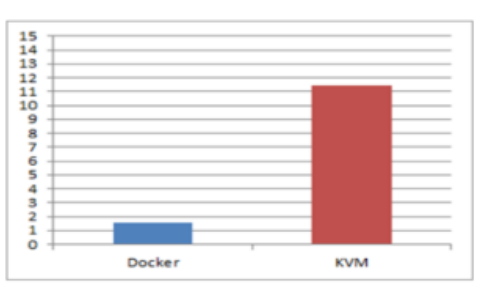


Рисунок 1.2.4 - Середній час завантаження Docker проти KVM

Для розрахунку оперативної швидкості використовувалася мова python. На малюнку 1.2.5 показано, що швидкість роботи 100 000 в середньому становить близько 4,5 с. Щоб виміряти швидкість роботи, вони отримують середній час процесу та стандартне відхилення, повторюючи той самий процес 100 разів на докері та віртуальній машині.

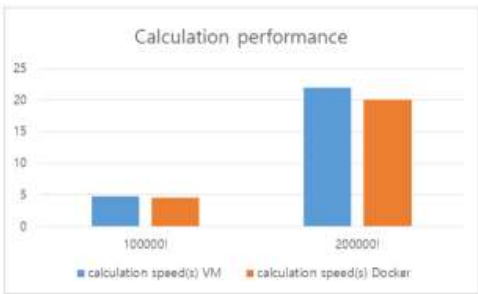


Рисунок 1.2.5 - Продуктивність обчислення ЦП

На рисунку 5 показано, що швидкість обчислення докера трохи вища, ніж швидкість обчислення ВМ. Висновок, що ВМ працює незалежно. Це одна з причин легкості застосування та керування політикою мережі, безпеки, користувача та системи. Однак докер не містить гостьової операційної системи. Таким чином, розповсюдження та збір зображень займає набагато менше часу. Його час завантаження також дуже короткий. Це основні переваги використання докер-хмари порівняно з VM Cloud. Технології віртуалізації LXC і Xen для порівняння деяких програм . Для цього використовує два сервери Core OS 324.3.0 і XenServer 6.2 з докером версії 0.11.1. Конфігурація цих систем — 4 ГБ оперативної пам’яті, чотириядерний процесор Intel Xeon і підтримка віртуалізації Intel VT-X. Базовою операційною системою є Ubuntu 12.04, і контейнери працюватимуть на обох машинах. 2 ГБ пам’яті виділено для першої віртуальної машини та Apache 2.2, WordPress 3.9 і PHP 5.3. Це було використано як сервер додатків. 1 ГБ пам’яті використовується другою віртуальною машиною з базою даних MYSQL 5.5. Ця база даних була заповнена зразком вмісту WordPress. Ця машина використовувалася як сервер бази даних. JMeter використовувався як інструмент тестування. На малюнку 1.2.6 показано, що під час виконання запиту SELECT у LXC спостерігалося менше накладних витрат порівняно з Xen. Основна увага під час виконання цього процесу порівняння полягає в тому, щоб побачити використання процесора та продуктивність швидкості мережі, оскільки це основні ресурси, що споживаються в цьому тесті.

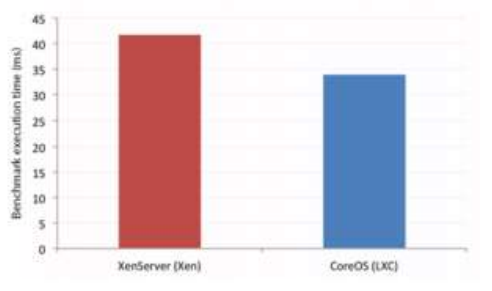


Рисунок 1.2.6 - Час у мілісекундах для виконання одного запиту на вибірку SQL

На малюнку 1.2.7 показано, що в налаштуваннях Xen виконання запиту INSERT у базі даних займало 16 секунд, тоді як у налаштуваннях LXC це займало більше часу — близько 335 секунд. Це показує нездатність контейнера LXC ефективно ізолювати ресурси.

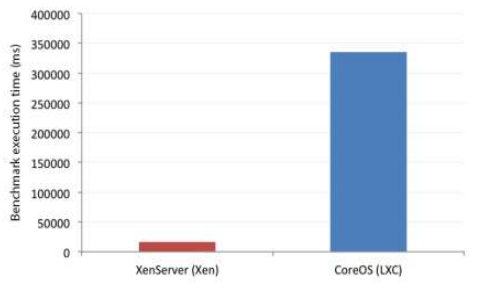


Рисунок 1.2.7 - Час у мілісекундах для виконання 10 000 запитів SQL INSERT

Висновок, що Xen був би кращим вибором у сенсі рівномірного розподілу ресурсів, продуктивність не залежить від інших завдань і виконується на одній машині. Однак LXC набагато кращий у сенсі використання більшості апаратних ресурсів або виконання менших ізольованих процесів. У приватних і точкових хмарах LXC є кращим варіантом. Оцінка продуктивність трьох різних середовищ: Native, Docker і KVM. Проблеми накладних витрат також висвітлюються в дослідженні. Були досліджені сценарії, коли більше одного апаратного ресурсу було використано повністю. Для проведення тестів використовувався сервер IBM x3650 M4, 16-ядерний процесор Xeon E5-2665, два Intel Sandy Bridge-EP 2,4 - 3,0 ГГц і 256 ГБ оперативної пам'яті. Щоб зробити нерівномірний доступ до пам'яті, два процесори були з'єднані між собою за допомогою зв'язку QPI. Хмарні провайдери також використовують подібний сервер. Базовою операційною системою були Ubuntu 13.10, докер версії 1.0, ядро Linux 3.11.0, libvirt версії 1.1.1 і QEMU 1.5.0. Цей малюнок 1.2.8 показує, що середній розмір 1 Мбайт використовувався для введення-виведення, трохи більше 60 секунд, вимірюючи продуктивність послідовного читання та запису. В цьому випадку, Докер і KVM можуть помітити невеликі витрати. В інших випадках KVM має майже чотириразову різницю в продуктивності.

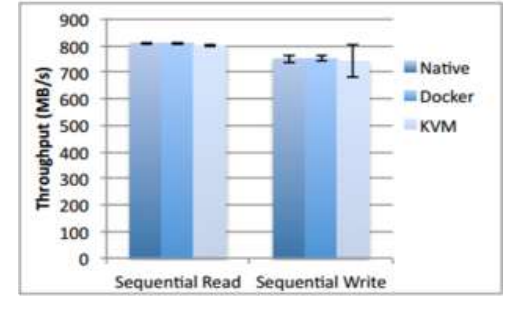


Рисунок 1.2.8 - Пропускна здатність послідовного вводу-виводу (МБ/с)

На малюнку 1.2.9 показано виконання нерегулярних робочих навантажень читання, запису та змішаних навантажень із використанням розміру квадрата 4 Кбайт і одночасності 128, що, як ми попередньо вирішили, забезпечує найкраще виконання для цього конкретного SSD. Як і слід було очікувати, докер не використовує накладних витрат, на відміну від Linux, однак KVM передає однакову кількість IOPS, оскільки кожна операція вводу-виводу повинна мати QEMU. Незважаючи на те, що максимальна продуктивність віртуальної машини залишається дуже високою, вона використовує більше циклів ЦП на операцію вводу-виводу, залишаючи менше ЦП доступним для роботи програми.

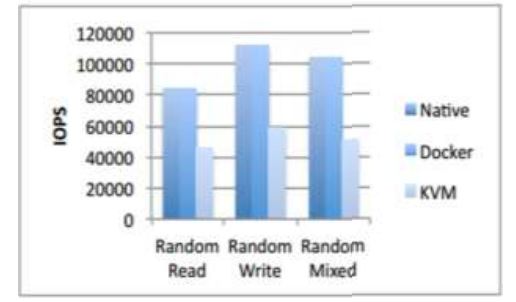


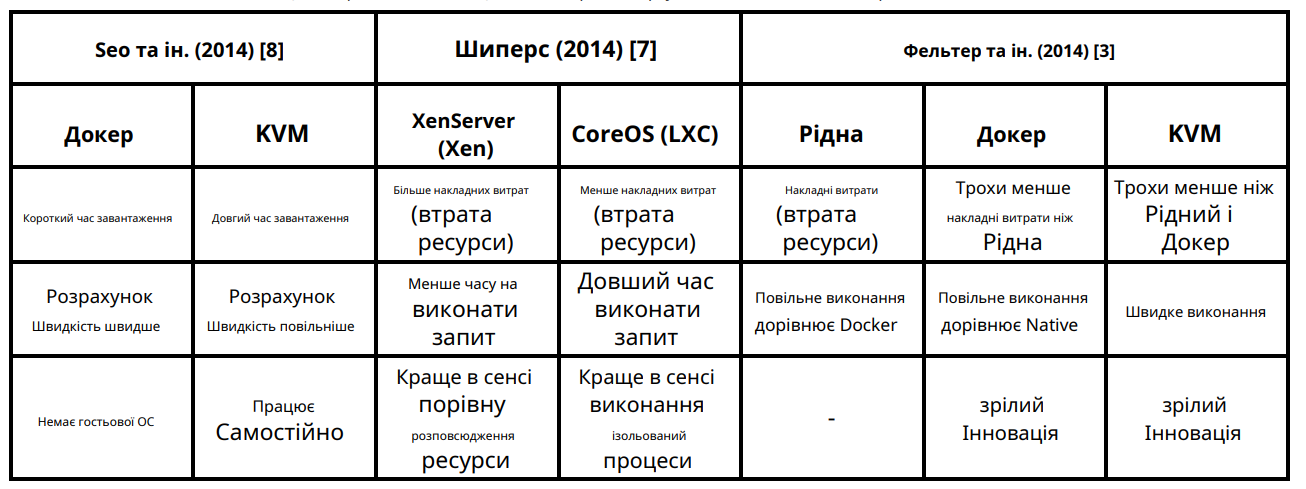
Рисунок 1.2.9 - Пропускна здатність послідовного вводу-виводу

Висновок, що контейнери та віртуальні машини є зрілими інноваціями, які отримали вигоду від інкрементального обладнання та вдосконалення програмування за останні 10 років. Коли все сказано та зроблено, докер еквівалентний або перевершує виконання KVM для кожної ситуації, яку ми пробували. Наші результати демонструють, що як KVM, так і докер представляють нерелевантні накладні витрати на виконання процесора та пам’яті. Підсумовуючи ці минулі роботи, незалежно від використання відмінних методів і наявності різноманітних центрів, одна загальна річ – вимірювання та порівняння продуктивності додатків і різних типів контейнерних і віртуалізованих технологій.

### 1.2.9 Вибір між ВМ та контейнерів

Таблиця 1.2.1 порівнює особливості різних контейнерних технологій і технологій віртуальних машин. Віртуальна машина використовує додатковий рівень між головною операційною системою та гостьовою операційною системою. Цей рівень відомий як гіпервізор. Тоді як Docker додає додатковий рівень між головними операційними системами та місцем віртуалізації та виконання додатків, який відомий як Docker Engine. Оскільки докер не використовує жодної гостьової операційної системи, це робить велику різницю в продуктивності між контейнером докерів і технологією віртуальної машини. У таблиці 1.2.1 також коротко порівнюється продуктивність програм, що працюють у різних контейнерах і віртуальних машинах. Як наведено в таблиці вище, згідно з цим продуктивність докера краща, ніж KVM, з точки зору часу завантаження та швидкості обчислень.

Таблиця 1.2.1 - Порівняльна таблиця на основі різних віртуальних машин і контейнерної технології



### 1.2.10 Підведення підсумків

Docker автоматизує програми, коли вони контейнеризовані. До основної операційної системи додається додатковий рівень механізму докерів. Докер працює швидше, ніж віртуальні машини, оскільки він не має гостьової операційної системи та менше накладних ресурсів. Тому для використання сервісу Zabbix ми використаємо контейнерну технологію.

## **1.3 Віртуалізація серверів**

Оскільки для дослідження ефективності використання аналізу систем Zabbix нам потрібні два сервери, скористуємось інструментами віртуалізації. Для цього був обраний інструмент – oracle virtual box. Розглянемо детально, що нам може запропонувати цей інструмент.

### 1.3.1 Термінологія

Коли ви маєте справу з віртуалізацією, а також для розуміння наступних розділів дослідження, це допоможе ознайомитися з деякою важливою термінологією, особливо з такими термінами:

* Головна операційна система (хост-ОС). Це ОС фізичного комп’ютера, на якому встановлено Oracle VM VirtualBox. Існують версії Oracle VM VirtualBox для хостів Windows, macOS, Linux і Oracle Solaris.
* Гостьова операційна система (гостьова ОС). Це ОС, яка працює всередині віртуальної машини. Теоретично Oracle VM VirtualBox може запускати будь-яку ОС x86, наприклад DOS, Windows, OS/2, FreeBSD і OpenBSD. Але щоб досягти майже рідної продуктивності гостьового коду на вашій машині, нам довелося пройти через багато оптимізацій, які характерні для певних ОС. Таким чином, хоча ваша улюблена ОС може працювати як гостьовий режим, ми офіційно підтримуємо та оптимізуємо для кількох вибраних, які включають найпоширеніші ОС.
* Віртуальна машина (ВМ). Це спеціальне середовище, яке Oracle VM VirtualBox створює для вашої гостьової ОС під час її роботи. Іншими словами, ви запускаєте гостьову ОС у віртуальній машині. Зазвичай віртуальна машина відображається у вигляді вікна на робочому столі комп’ютера. Залежно від того, який із різних інтерфейсів Oracle VM VirtualBox ви використовуєте, віртуальна машина може відображатися в повноекранному режимі або віддалено на іншому комп’ютері.

Всередині Oracle VM VirtualBox розглядає віртуальну машину як набір параметрів, які визначають її поведінку. Деякі параметри описують налаштування апаратного забезпечення, наприклад обсяг пам’яті та кількість призначених ЦП. Інші параметри описують інформацію про стан, наприклад, чи працює віртуальна машина чи збережена.

Ви можете переглянути ці параметри віртуальної машини в VirtualBox Manager у вікні налаштувань і за допомогою команди VBoxManage

* Гостьові доповнення. Це стосується спеціальних пакетів програмного забезпечення, які постачаються з Oracle VM VirtualBox, але призначені для встановлення всередині віртуальної машини для покращення продуктивності гостьової ОС і додавання додаткових функцій.

### 1.3.2 Огляд функцій

Нижче наведено короткий опис основних функцій Oracle VM VirtualBox:

1. Портативність. Oracle VM VirtualBox працює на великій кількості 64-розрядних операційних систем.

Oracle VM VirtualBox — це так званий розміщений гіпервізор, який іноді називають гіпервізором типу 2. У той час як залізо комп’ютеру або гіпервізор типу 1 працює безпосередньо на апаратному забезпеченні, Oracle VM VirtualBox вимагає встановлення наявної ОС. Таким чином, він може працювати разом із існуючими програмами на цьому хості.

Oracle VM VirtualBox значною мірою функціонально ідентичний на всіх хост-платформах, і використовуються однакові формати файлів і зображень. Це дає змогу запускати віртуальні машини, створені на одному хості, на іншому хості з іншою хост-ОС. Наприклад, ви можете створити віртуальну машину в Windows, а потім запустити її в Linux.

Крім того, віртуальні машини можна легко імпортувати та експортувати за допомогою відкритого формату віртуалізації (OVF), галузевого стандарту, створеного для цієї мети. Ви навіть можете імпортувати файли OVF, створені за допомогою іншого програмного забезпечення віртуалізації.

Для користувачів Oracle Cloud Infrastructure функціональні можливості поширюються на експорт та імпорт віртуальних машин до та з хмари. Це спрощує розробку додатків і розгортання у виробничому середовищі.

1. Гостьові доповнення: спільні папки, 3D-віртуалізація. Гостьові додатки Oracle VM VirtualBox — це пакети програмного забезпечення, які можна інсталювати всередині підтримуваних гостьових систем, щоб покращити їхню продуктивність і забезпечити додаткову інтеграцію та зв’язок із хост-системою. Після встановлення гостьових доповнень віртуальна машина підтримуватиме автоматичне налаштування роздільної здатності відео, безшовні вікна, прискорену 3D-графіку тощо.

Зокрема, гостьові додатки забезпечують спільні папки , які дають вам доступ до файлів у системі хоста з гостьової машини.

1. Комплексна апаратна підтримка. Серед інших функцій Oracle VM VirtualBox підтримує наступне:
   * Гостьова багатопроцесорна обробка (SMP). Oracle VM VirtualBox може представити до 32 віртуальних процесорів для кожної віртуальної машини, незалежно від того, скільки ядер ЦП фізично присутні на вашому хості.
   * Підтримка USB-пристроїв. Oracle VM VirtualBox реалізує віртуальний USB-контролер і дає змогу підключати довільні USB-пристрої до віртуальних машин без необхідності встановлення драйверів для окремих пристроїв на хості. Підтримка USB не обмежується певними категоріями пристроїв.
   * Апаратна сумісність. Oracle VM VirtualBox віртуалізує величезну кількість віртуальних пристроїв, серед яких багато пристроїв, які зазвичай надаються іншими платформами віртуалізації. Це включає в себе контролери жорстких дисків IDE, SCSI та SATA, кілька віртуальних мережевих карт і звукових карт, віртуальні послідовні та паралельні порти та вдосконалений програмований контролер переривань вводу/виводу (I/O APIC), який є в багатьох комп’ютерних системах. Це дозволяє легко клонувати образи дисків із реальних машин та імпортувати віртуальні машини сторонніх розробників у Oracle VM VirtualBox.
   * Повна підтримка ACPI. Інтерфейс розширеної конфігурації та живлення (ACPI) повністю підтримується Oracle VM VirtualBox. Це дозволяє легко клонувати образи дисків із реальних машин або віртуальних машин сторонніх розробників у Oracle VM VirtualBox. Завдяки унікальній підтримці статусу живлення ACPI Oracle VM VirtualBox може навіть повідомляти гостьовим ОС із підтримкою ACPI статус живлення хоста. Для мобільних систем, що працюють від акумулятора, гість може таким чином увімкнути енергозбереження та повідомити користувача про залишок заряду, наприклад, у повноекранних режимах.
   * Багатоекранна роздільна здатність. Віртуальні машини Oracle VM VirtualBox підтримують роздільну здатність екрана, яка у багато разів перевищує роздільну здатність фізичного екрана, що дозволяє розподіляти їх на великій кількості екранів, підключених до хост-системи.
   * Вбудована підтримка iSCSI. Ця унікальна функція дає змогу підключати віртуальну машину безпосередньо до сервера зберігання iSCSI без використання хост-системи. Віртуальна машина отримує прямий доступ до цілі iSCSI без додаткових витрат, необхідних для віртуалізації жорстких дисків у файлах-контейнерах.
   * Завантаження мережі PXE. Вбудовані віртуальні мережеві карти Oracle VM VirtualBox повністю підтримують віддалене завантаження за допомогою Preboot Execution Environment (PXE).
2. Багатогенераційні розгалужені знімки. Oracle VM VirtualBox може зберігати довільні знімки стану віртуальної машини. Ви можете повернутися в минуле та повернути віртуальну машину до будь-якого такого знімка та запустити звідти альтернативну конфігурацію віртуальної машини, фактично створивши ціле дерево знімків. Ви можете створювати та видаляти знімки під час роботи віртуальної машини.
3. Групи ВМ. Oracle VM VirtualBox надає групову функцію, яка дозволяє користувачеві організовувати та контролювати віртуальні машини як колективно, так і окремо. Окрім основних груп, будь-яка віртуальна машина також може входити до кількох груп, а також для груп, вкладених у ієрархію. Це означає, що ви можете мати групи груп. Загалом операції, які можна виконувати над групами, такі ж, як і ті, які можна застосовувати до окремих віртуальних машин: запуск, призупинення, скидання, закриття (збереження стану, надсилання вимкнення, вимкнення), скасування збереженого стану, відображення у файловій системі, Сортувати.
4. Чиста архітектура та безпрецедентна модульність. Oracle VM VirtualBox має надзвичайно модульну конструкцію з чітко визначеними внутрішніми інтерфейсами програмування та чітким розділенням коду клієнта та сервера. Це дозволяє легко керувати нею з кількох інтерфейсів одночасно. Наприклад, ви можете запустити віртуальну машину, просто натиснувши кнопку в графічному інтерфейсі користувача Oracle VM VirtualBox, а потім керувати цією машиною з командного рядка або навіть віддалено. Дивіться Розділ 1.18, «Альтернативні інтерфейси» .

Завдяки своїй модульній архітектурі Oracle VM VirtualBox також може розкрити свою повну функціональність і конфігурацію за допомогою комплексного набору для розробки програмного забезпечення (SDK), який забезпечує інтеграцію Oracle VM VirtualBox з іншими програмними системами.

1. Віддалений дисплей машини. Розширення віддаленого робочого столу VirtualBox (VRDE) забезпечує високопродуктивний віддалений доступ до будь-якої запущеної віртуальної машини. Це розширення підтримує протокол віддаленого робочого стола (RDP), спочатку вбудований у Microsoft Windows, зі спеціальними доповненнями для повної підтримки клієнтського USB.

VRDE не покладається на сервер RDP, вбудований у Microsoft Windows. Натомість VRDE підключається безпосередньо до рівня віртуалізації. Як результат, він працює з гостьовими ОС, відмінними від Windows, навіть у текстовому режимі, і також не потребує підтримки програм у віртуальній машині. VRDE детально описано в розділі 7.1 «Віддалений дисплей (підтримка VRDP)» .

Окрім цієї спеціальної можливості, Oracle VM VirtualBox пропонує вам більше унікальних функцій:

* + Розширювана автентифікація RDP. Oracle VM VirtualBox уже підтримує Winlogon у Windows і PAM у Linux для автентифікації RDP. Крім того, він містить простий у використанні SDK, який дозволяє створювати довільні інтерфейси для інших методів автентифікації.
  + USB через RDP. Використовуючи підтримку віртуального каналу RDP, Oracle VM VirtualBox також дозволяє локально підключати довільні USB-пристрої до віртуальної машини, яка працює віддалено на RDP-сервері Oracle VM VirtualBox.

### 1.3.3 Підведення підсумків

Проаналізовані можливості роботи інструменту віртуалізації VM VirtualBox. Це інструмент який повністю дозволить використовувати віртуальні машини як реальні сервера, які нам знадобляться для аналізу ефективності аналізу систем Zabbix. В них ми будемо створювати докер зображення, які зменшать затрати ресурсів “реального серверу”.

# **2. Підготовка для аналізу ефективності Zabbix**

## **2.1 Загальна архітектура для проведення аналізу**

Для проведення аналізу ефективності Zabbix знадобиться 2 сервер. Для розуміння на якому сервері будуть використані інструменти, дамо їм назву.

### 2.1.1 Сервер А

Це основний сервер, на якому буде встановлений докер контейнер з інструментом для системного аналізу – Zabbix server. Також щоб зберігати дані звітів, Zabbix потребує сконфігуроване зображення з PostgresSQL. Щоб візуалізувати дані знадобиться веб сервер – апачі або nginx, для цілей аналізу обрано Nginx web server, який буде також в докер контейнері. Щоб збирати метрики з системи знадобиться ще Zabbix агент.

### 2.1.2 Сервер Б

Це ще один сервер, в тій же локальній мережі, як і сервер А. На ньому в докер контейнері буде піднятий тільки Zabbix агент, який також буде збирати метрики з нього. ОС серверу – це ubuntu 22.04 server.

На рисунку 2.1.1 ілюстрована загальна архітектура, яка потрібна для тестування ефективності zabbix. ОС серверу – це ubuntu 22.04 server.

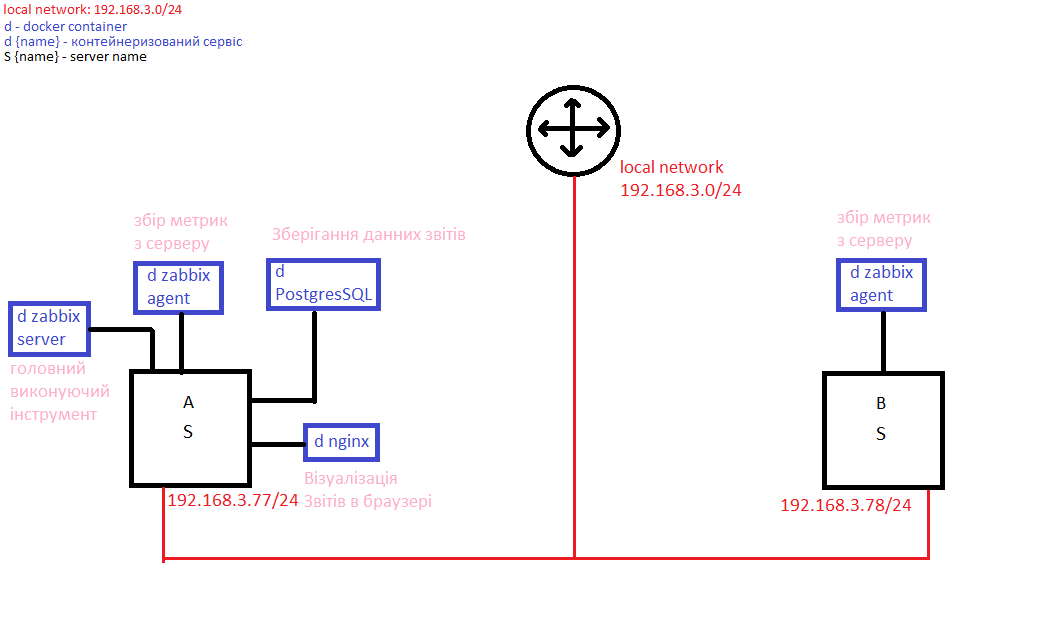


Рисунок 2.1.1 - загальна архітектура

### 2.1.3 Підведення підсумків

В цьому розділі ми проаналізували що нам знадобиться для дослідження

## **2.2 Конфігурація серверів**

### 2.2.1 Конфігурація серверу А

Після інсталяції ОС, необхідно для конфігурації створити скрипт, який необхідний для встановлення контейнерів забігс серверу, забігс агенту, PostgresSQL та Nginx.

Оскільки конфігурація не є частиною дослідження, тому все зроблено автоматизованим скриптом, який при виконані всі необхідні докер-контейнери піднімає і запускає.

Необхідно зайти на сервер, через FileZila або іншими можливостями зайти на сервер та перекинути 2 файли конфігурації: serverA-config.sh та docker-compose-serverA.yaml.

Зайдемо на сервер та виконаємо скрипт через bash, як на малюнку 2.2.1.

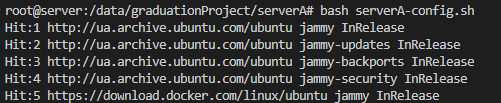


Рисунок 2.2.1 - процес виконання скрипту

Далі перевіримо, що наші контейнери сконфігуровані, командою: docker ps -a, як на малюнку 2.2.2

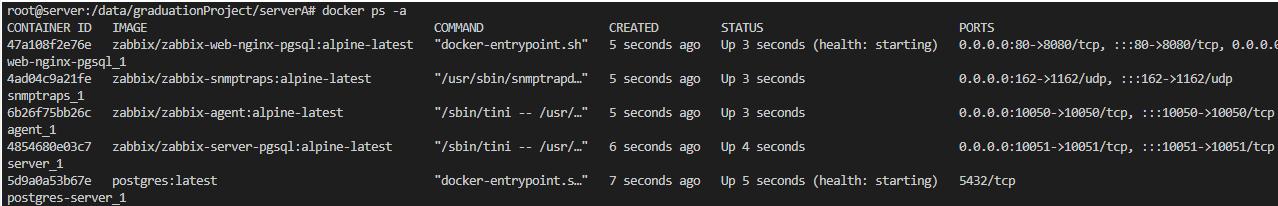


Рисунок 2.2.2 – список контейнерів на сервері А

Бачимо, що в строчці статусу «Up» - це свідчить про те, що контейнери піднялись та готови для роботи. На цьому конфігурація серверу А завершена.

### 2.2.2 Конфігурація серверу Б

На сервері Б для аналізу нам потрібен тільки контейнер з забігс агентом, який буде збирати метрики з серверу.

Необхідно зайти на сервер, через FileZila або іншими можливостями зайти на сервер та перекинути 2 файли конфігурації: serverB-config.sh та docker-compose-serverB.yaml.

Результати виконання скрипту подивимось на малюнку 2.2.3.



Рисунок 2.2.3 - процес виконання скрипту

Перевіримо, що контейнер з агентом піднятий, через команду docker ps, як на малюнку 2.2.4.



Рисунок 2.2.4 - список контейнерів на сервері Б

### 2.2.3 Підведення підсумків

Сервери налаштовані та всі інструменти інстальовані, далі ми можемо досліджувати ефективність використання Zabbix для системного аналізу.

# **3. Аналіз ефективності Zabbix**

# **Джерела**

[13.02.2023] Інтернет ресурс з загальними відомостями Zabbix: <https://www.bookmyessay.com/zabbix-assignment/>

[15.02.2023] Інтернет ресурс з загальними відомостями Virtual box: <https://www.virtualbox.org/manual/ch01.html#hostossupport>

[14.02.2023] pdf, дослідження використання docker: <https://www.scribd.com/document/462316610/028-AnIntroductiontoDockerandAnalysisofitsPerformance-IJCSNS-20170327>

[15.02.23] Інтернет ресурс з контейнерами, інструментом Zabbix та параметрами для конфігурації: <https://hub.docker.com/u/zabbix>

[15.02.23] Інтернет ресурс з повною інструкцію використання docker engine: <https://docs.docker.com/engine/reference/builder/>

[15.02.23] Інтернет ресурс з повною інструкцію використання docker-compose: <https://docs.docker.com/compose/compose-file/>