Лекція

з дисципліни: «Клієнт-серверне програмування»

на тему: «*Virtualizations. Containers. Docker*.»

ПЛАН ЛЕКЦІЇ

1. Віртуалізація.
2. Принцип контейнеризації.
3. Основи Docker.

ЛІТЕРАТУРА

1. Ian Miell, Aidan Hobson Sayers, Docker in Practice 2nd Edition, 2019 by Manning Publications Co., 384p.
2. Л. О. Левченко, Н. Г. Кучук, Ю. А. Тарнавський, В. П. Колумбет Архітектура системного програмного забезпечення [Електронний ресурс]. Київ : КПІ ім. Ігоря Сікорського, 2022. – 497 с.

1. Віртуалізація.

Віртуалізація – це технологія створення уявлення кількох комп'ютерів чи серверів із урахуванням одного фізичного комп'ютера, сервера чи серверного кластера. Ця фізична машина називається хостом; у неї є певна конфігурація процесора, оперативної та дискової пам'яті тощо.

Фізичні ресурси за допомогою спеціалізованого програмного забезпечення розподіляються таким чином, щоб розгорнути кілька незалежних одна від одної віртуальних машин.

Іншими словами, віртуалізація – ілюзія присутності кількох окремих комп'ютерів, тобто віртуальних машин, на тому самому фізичному обладнанні. А створюється ця ілюзія за допомогою гіпервізора.

Гіпервізор (virtual machine monitor, VMM) — це комп’ютерне програмне забезпечення, вбудоване або апаратне забезпечення, яке створює та запускає віртуальні машини. Комп’ютер, на якому гіпервізор запускає одну або декілька віртуальних машин, називається *хост-машиною*, а кожна віртуальна машина – *гостьовою машиною*.

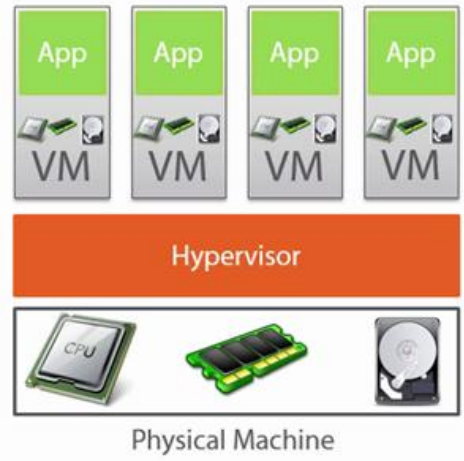


Рисунок 1. Принцип віртуалізації

Програмне забезпечення гіпервізора підключається безпосередньо до фізичного обладнання, дозволяючи розділити одну систему на кілька віртуальних машин (ВМ) і належним чином розподілити ресурси машини.

Віртуалізацію можна пояснити простими словами:

1. Одне обладнання або фізичний ресурс може створити багато віртуальних ресурсів.

Гіпервізор керує фізичними ресурсами обчислювальної машини та розподіляє ці ресурси між декількома різними операційними системами, дозволяючи запускати їх одночасно.

Іншими словами, гіпервізор створює з одного фізичного комп'ютера кілька копій, клонів апаратних ресурсів, і кожен клон видно з боку користувача як окремий пристрій. На кожну віртуальну машину можна встановити гостьову операційну систему користувача, яка не прив'язана до «заліза» хоста.

Гіпервізор ізолює запущені ОС один від одного так, щоб кожна їх монопольно використовувала виділені їй ресурси. Але за необхідності гіпервізор дозволяє операційним системам віртуальних машин і взаємодіяти між собою. Механізмом зв'язку між ОС може бути загальний доступ до певних файлів та обмін даними по локальній мережі.

Гіпервізори прийнято поділяти на два типи. Але є ще й, так званий, гібридний гіпервізор, який поєднує властивості обох типів.

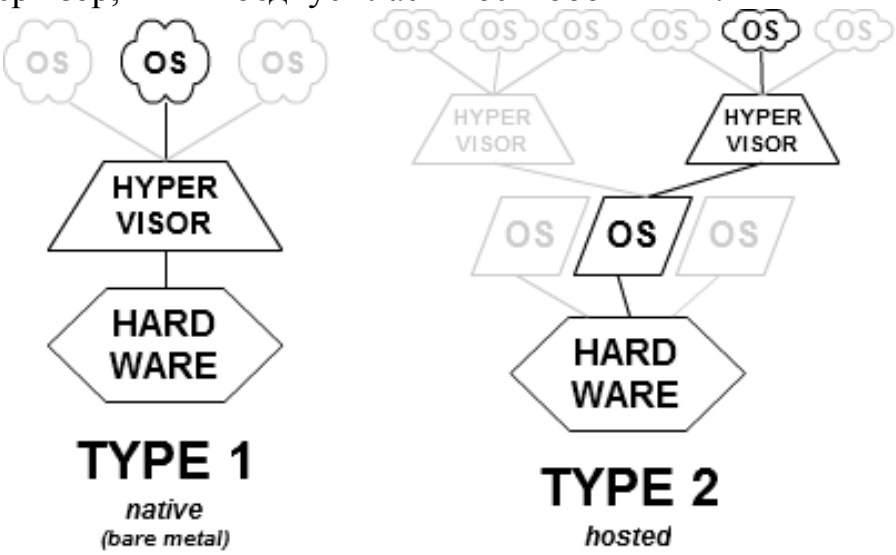


Рисунок 2. Принцип роботи гіпервізора

У чому особливості роботи гіпервізорів 1 типу?

Гіпервізор першого типу називають ще мікроядром, тонким гіпервізором, автономним гіпервізором, що виконується на «голому залізі». Гіпервізор першого типу найпростіше сприймати як специфічну компактну операційну систему, яка встановлюється прямо на залізо (bare-metal server) і має основні ознаки ОС:

• замість невпорядкованого набору апаратного забезпечення надає абстрактний набір ресурсів для прикладних програм (так званий, «погляд зверху»);

• управляє набором ресурсів – розподіляє процесорний час, пам'ять, пристрої вводу-виводу між програмами, які претендують на використання ресурсів комп'ютера (так званий, «погляд знизу»).

Він надає гостьовим ОС, запущеним під його керуванням на верхньому рівні, абстракцію, тобто службу віртуальної машини. У результаті кожна гостьова операційна система отримує собі від гіпервізора ілюзію повноправного розпорядження всіма «нижчими» ресурсами комп'ютера – аналогічно тому, якби ОС працювала на реальному устаткуванні в привілейованому режимі ядра, режимі супервізора.

• Гіпервізор типу 1: Цей Гіпервізор не потребує додаткового програмного забезпечення для процесу інсталяції і безпосередньо працює на апаратних ресурсах хост-машини.

Більшість сучасних процесорів Intel і AMD для десктопів і серверів на апаратному рівні підтримує технологію віртуалізації та поділ роботи ОС на два рівні привілеїв: режим ядра (привілейований) та режим користувача.

У ПЗ гіпервізора першого типу є дуже важлива особливість – розмір коду на два порядки (тобто в сотні разів) менше, ніж у більшості сучасних операційних систем. Це забезпечує так само меншу кількість можливих помилок, що призводять до зависання всієї системи. Збій у роботі ОС на одній із віртуальних машин користувача не повинен вплинути на роботу всіх сусідніх машин на тому самому фізичному обладнанні.

Однією з найважливіших вимог до гіпервізору є безпека, оскільки гіпервізор отримує повне управління апаратними ресурсами комп'ютера, на яких виконується віртуалізація. Отже, завдання гіпервізора – виконувати машинні інструкції безпечним чином, не дозволяючи гостьовій ОС:

• блокувати переривання;

• модифікувати таблиці відображення сторінок віртуальної пам'яті на фізичну для комп'ютера;

• змінювати дані в осередках пам'яті, виділених інших запущених процесів (крім випадків, коли це заздалегідь передбачено логікою роботи, обміну даними між ними).

Системні виклики також перехоплюються та виконуються всередині гіпервізора, але з боку кожної гостьової ОС все виглядає так, як має бути при звичайному виконанні інструкцій у її режимі ядра. Іншими словами, гіпервізор створює «ілюзію» для гостьової операційної системи, що її код виконується хостом на рівні заліза, у привілейованому режимі, хоча де-факто вона функціонує в режимі користувача. Якщо відбудеться крах однієї з гостьових систем, робота решти триватиме.

Гіпервізор виявляється єдиним програмним забезпеченням, яке запущене в режимі максимальних привілеїв. Така властивість гіпервізора називається еквівалентністю – поведінка програм користувача не відрізняється під час роботи на віртуальній машині та на фізичному обладнанні за винятком тимчасових характеристик.

До першого типу гіпервізорів можна віднести Xen, VMware ESXi, Hyper-V та низку інших.

Гіпервізор другого типу працює як один із процесів, що виконуються основною ОС

Гіпервізор другого типу називається також хостовим (hosted). Він є додатковим програмним шаром, розташованим поверх основної операційної системи.

Фактично гіпервізор другого типу працює як один із процесів, що виконуються основною ОС, найчастіше – Linux. І тут повноваження гіпервізора значно скромніше: він управляє гостьовими операційними системами, а емуляцію і управління фізичними ресурсами перебирає хостова ОС.

Найбільш популярні гіпервізори другого типу - Oracle VirtualBox, VMware Workstation, KVM.

Гібридний гіпервізор управляє процесором та пам'яттю, а пристроями введення-виводу – через гостьові ОС

Гібридні гіпервізори мають частину ознак як першого, так і другого типів (поєднання "тонкого" гіпервізора і спеціальної, що працює "на залозі" службової ОС під його керуванням). Гіпервізор управляє безпосередньо процесором і пам'яттю, а через службову ОС гостьові отримують доступ до пристроїв вводу-виводу.

Технології постійно розвиваються, і виробники гіпервізорів шукають шляхи вдосконалення своїх продуктів, створюють нові версії, більш гнучкі, більш інтегровані до різних систем та умов. В останні роки гіпервізори Xen і Hyper-V все частіше відносять вже не до першого типу, а до гібридного, і це частково вірно. Сучасні версії цих гіпервізорів значною мірою поєднують у собі властивості обох типів.

Що таке віртуальні машини?

Сучасна віртуальна машина дозволяє приховати від встановленої на ній ОС деякі параметри фізичних пристроїв комп’ютера і тим самим забезпечити взаємну незалежність ОС і встановленого устаткування. Такий підхід надає користувачам(і/або адміністраторам обчислювальних машин) цілий ряд переваг, до котрих зокрема належать:

- можливість установки на одному комп’ютері декількох ОС без необхідності відповідної конфігурації фізичних жорстких дисків;
- робота з декількома ОС одночасно з можливістю динамічного перемикання між ними без перенавантаження системи;
- скорочення часу зміни складу встановлених ОС;
- ізоляція реального устаткування від небажаного впливу програмного забезпечення, що працює в середині віртуальної машини;
- можливість моделювання обчислювальної мережі на єдиному автономному комп’ютері.

Не дивлячись на всі переваги, віртуальні машини також мають і свої недоліки:

- Потреба в наявності достатніх апаратних ресурсів для функціонування декількох ОС одночасно

- ОС працює дещо повільніше
- Є методи визначення того що програма працює у віртуальній машині, що використовується вірусними програмами
- Різні платформи віртуалізації поки що не підтримують повну віртуалізацію всього апаратного забезпечення і інтерфейсів

З погляду користувача, віртуальна машина – це конкретний екземпляр якогось віртуального обчислювального середовища(віртуального компютера), створений за допомогою спеціального програмного інструменту. Зазвичай такі інструменти дозволяють запускати довільне число віртуальних машин, що обмежується лише фізичними ресурсами реального комп'ютера.

Власне інструмент для встановлення віртуальної машини (його іноді називають додатком віртуальних машин) – це звичайне застосування, що встановлюється, як і будь-яке інше, на конкретну реальну операційну систему. Ця реальна ОС іменується «господарською» або ж хостовою ОС (від англ. терміну host– головний, базовий).

Всі завдання по управлівлінню віртуальними машинами вирішує спеціальний модуль у складі додатку віртуальної машини – монітор віртуальної машини(МВМ). Монітор грає роль посередника в усіх взаємодіях між віртуальними машинами і базовими устаткуваннями , підтримуючи виконання всіх створених віртуальних машин на єдиній апаратній платформі і забезпечуючи їх надійну ізоляцію.

Користувач не має безпосереднього доступу до монітору віртуальної машини, адже у більшості програмних продуктів йому надається лише графічний інтерфейс для створення і настройки віртуальних машин, який зазвичай називають консоллю віртуальних машин.

«Усередині» віртуальної машини користувач встановлює, як і на реальному комп'ютері необхідну йому операційну систему. Така ОС, що належить конкретній віртуальній машині, називається гостьовою(guest OS). Перелік підтримуваних гостьових ОС є однією з найбільш важливих характеристик віртуальної машини. Найбільш потужні з сучасних віртуальних машин забезпечують підтримку близько десятка популярних версій ОС з сімейств Windows, Linux, Macos.

2. *Контейнери.*

Віртуальні машини можуть зайняти багато системних ресурсів. Кожна віртуальна машина працює не тільки з повною копією операційної системи, але і з віртуальною копією всього обладнання, яке операційна система потребує для запуску. Це значно збільшує об'єм необхідної оперативної пам'яті і необхідний процесорний час. Це все ще економічно вигідно, порівняно з запуском окремих реальних комп'ютерів, але для деяких застосувань це може бути надмірним, що призвело до розробки контейнерів.

Контейнер – це стандартна одиниця ПЗ, в яку упаковано додаток з усіма необхідними для його роботи залежностями – кодом програми,

середовищем запуску, системними інструментами, бібліотеками і налаштуваннями.

Контейнери можуть розгорнути більше програм на одному фізичному сервері, ніж гіпервізори

Що таке контейнерні рішення?

Останні кілька років гіпервізори стала відтісняти на другий план порівняно нова технологія контейнерів. Причина цього в тому, що контейнери можуть на одному фізичному сервері розгорнути більшу кількість програм порівняно з гіпервізорами. Контейнерні рішення віртуалізації ґрунтуються переважно на доопрацьованому ядрі Linux. У цьому випадку, коли на хост-машині використовується ядро Linux, гостьовими ОС можуть бути тільки представники сімейства Linux.

За допомогою контейнерів, замість віртуалізації базового комп'ютера, як віртуальної машини (VM), віртуалізована тільки сама ОС.

Контейнери працюють поверх фізичного сервера, в якого його хостова ОС типово є Linux або Windows. Контейнери розділяють між собою єдине ядро хостової ОС і, як правило, також бінарні файли і бібліотеки. Ці спільні компоненти доступні лише для читання. Спільне використання ресурсів ОС, таких як бібліотеки, значно зменшує необхідність відтворення коду операційної системи і означає, що сервер може запускати кілька робочих навантажень за допомогою однієї установленої операційної системи. Таким чином, контейнери є надзвичайно легкими - вони мають розмір порядку кількох десятків мегабайт і запускаються протягом кількох секунд. У порівнянні з контейнерами запуск віртуальних машин проходить протягом кількох хвилин, і вони займають на порядок більше пам’яті, аніж еквівалентний до них за функціями контейнер.

На відміну від віртуальних машин, все, чого вимагає контейнер - це наявна операційна система, підтримуючі застосунки і бібліотеки, а також системні ресурси для запуску певної програми. На практиці це значить, що на одному сервері з контейнерами ви можете помістити в два-три рази більше застосунків, ніж з використанням ВМ. Крім того, за допомогою контейнерів ви можете створити портативне, послідовне робоче середовище для розробки, тестування та розгортання.

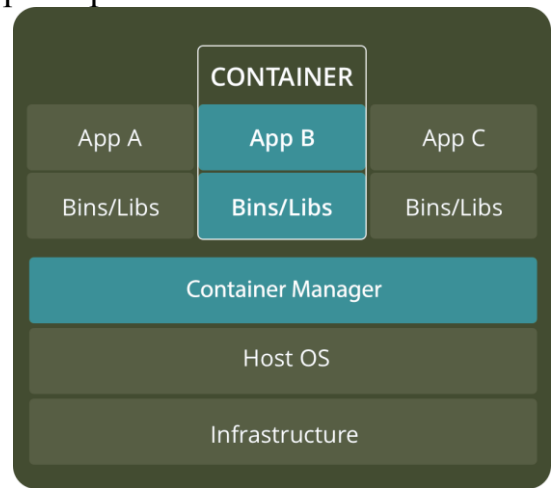


Рисунок 3. Структура контейнера

Типи контейнерів

Linux Containers (LXC). Оригінальна технологія контейнера Linux, відома як LXC. LXC - це метод віртуалізації на рівні операційної системи Linux для запуску декількох ізольованих систем Linux на одному хості.

Docker . Docker був запущений як проект з побудови контейнерів LXC з одним застосунком, вводячи кілька змін до LXC, які роблять контейнери більш портативними та гнучкими у використанні. Пізніше він перетворився на своє власне середовище виконання. На вищому рівні Docker - це утиліта Linux, яка може ефективно створювати, відправляти і запускати контейнери.

Переваги контейнерів

– гнучке середовище, контейнери можна створювати значно швидше, ніж екземпляри VM, їх розмір невеликий – вимірюється в мегабайтах, на відміну від розмірів VM, яка містить ОС, розмір якої може бути декілька гігабайтів; крім того, контейнери швидко запускаються;

– підвищена продуктивність, кожний контейнер можна розглядати як окремий мікросервіс, його оновлення не потребує синхронізації;

– керування версіями, дозволяє відстежувати версії контейнера та їх відмінності;

– переносимість середовища обчислення, контейнери інкапсулюють ОС та залежності додатків, тобто легко переносити образ контейнера з одного середовища в інше, наприклад, Windows/Linux;

– стандартизація, більшість контейнерів основані на відкритих стандартах і можуть працювати в усіх основних дистрибутивах Windows/Linux;

– безпека, контейнери ізолюють процеси одного контейнера від іншого, тобто будь-яке оновлення в одному контейнері не впливає на інший.

І контейнери, і віртуальні машини мають переваги та недоліки, і остаточне рішення буде залежати від ваших конкретних потреб, але є деякі загальні правила.

- Віртуальні машини є кращим вибором для запуску застосунків, які потребують усіх ресурсів та функціональності операційної системи, коли потрібно запускати декілька застосунків на серверах або мати широкий спектр операційних систем для керування.
- Контейнери є кращим вибором, коли найбільшим пріоритетом є максимізація кількості застосунків, що працюють на мінімальній кількості серверів.

Контейнери мають деякі переваги, що забезпечують такі варіанти використання, які важко або неможливо реалізувати у звичайних віртуальних машинах:

– контейнери спільно використовують ресурси основної ОС, що робить їх на порядок більш ефективними. Контейнери можна запускати і зупиняти за частки секунди. Для додатків, що запускаються в контейнерах, накладні витрати мінімальні або їх немає взагалі порівняно з додатками, що запускаються безпосередньо під керуванням основної ОС; переносимість контейнерів забезпечує потенційну можливість усунення цілого класу

програмних помилок, що викликаються незначними змінами робочого середовища, – позбавляється обґрунтування давній аргумент розробника «але це працює на моєму комп’ютері»;

– розробники можуть одночасно запускати десятки контейнерів, що дає можливість імітації роботи промислової розподіленої системи. Інженери з експлуатації можуть запустити на одному хості набагато більше контейнерів, ніж при використанні окремих віртуальних машин; крім того, контейнери надають переваги кінцевим користувачам і розробникам без необхідності розгортання програми у хмарі. Користувачі можуть завантажувати і запускати складні додатки без багатогодинної метушні з конфігурацією і проблемами під час установки й при цьому не турбуватися про будь-які зміни в їх локальних системах. У свою чергу, розробники подібних додатків можуть уникнути проблем, пов’язаних з відмінностями в конфігураціях призначених для користувача середовищ і з доступністю залежностей для цих додатків.

3. *Docker.*

Docker – це ПЗ з відкритим вихідним кодом, яке застосовується для розробки, тестування, доставки і запуску веб-додатків в середовищах з підтримкою контейнеризації. Він потрібен для більш ефективного використання системи і ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування і переносу в інші середовища з гарантійним збереженням стабільної роботи.

Основний принцип роботи Docker – контейнеризація. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення по ізольованим середовищам - контейнерам. Кожен з цих віртуальних блоків отримає всі необхідні елементи для роботи програми. Це дає можливість одночасного запуску великої кількості контейнерів на одному хості.

Docker - інструмент який дозволяє повністю ізолювати додаток. Забезпечує автоматизацію розгортання і управління додатками.

Додатки працюють в ізольованому середовищі (побудованої за допомогою просторів імен, namespaces, і груп процесів, cgroups), з одного боку ізолюючі процеси один від одного, з іншого боку не вдаючись при цьому до таких надлишкових засобів як віртуалізація або емуляція.

При використанні Docker контейнери поділяють одне і те ж ядро свого хостового комп'ютера. Наприклад, на хостовому комп'ютері Ubuntu Linux у вас можуть бути контейнер Ubuntu Linux і контейнер CentOS Linux. Обидва ці контейнери поділяють одне ядро Linux. Однак ви не можете мати контейнер Windows 10 на цьому ж хостовому комп'ютері Ubuntu Linux, тому що Windows використовує інше ядро. Спільне використання одного ядра вимагає набагато менше ресурсів, ніж використання окремих віртуальних машин, кожна з яких має власне ядро.

Контейнери також вирішують проблему, яка виникає, коли кілька застосунків потребують для роботи різних версій однієї бібліотеки. Оскільки

кожна програма знаходиться в своєму власному контейнері, вона ізольована від будь-яких конфліктних бібліотек і двійкових файлів.

Набір утиліт, які забезпечують роботу з контейнерами, називають «двигуном» (container engine), а його частина, яка відповідає за запуск та моніторинг контейнерів, – середовищем виконання (container runtime). Середовище виконання присвоює контейнерам власні ідентифікатори (ID).

Платформа Docker складається із двох окремих компонентів:

• Docker Engine механізму, що відповідає за створення і функціонування контейнерів, це клієнт-серверний додаток;

• Docker Hub хмарного сервісу для поширення контейнерів.

Механізм Docker Engine надає ефективний і зручний інтерфейс для запуску контейнерів. До цього для запуску контейнерів, що використовують таку технологію, як, наприклад, LXC (Linux Container), були потрібні неабиякий запас спеціальних знань у цій сфері та великий обсяг ручної роботи.

Docker Hub – публічний репозиторій з інтерфейсом, який надається Docker Inc (назва компанії розробника Docker Hub). Цей ресурс є джерелом «офіційних» образів, розроблених командою Docker або створених у співпраці з розробником ПЗ. Для офіційних образів перераховані їх потенційні уразливості. Ця інформація відкрита для будь-якого зареєстрованого користувача. Доступні як безкоштовні, так і платні акаунти.

Архітектура Docker

Docker використовує архітектуру клієнт-сервер. Docker-клієнт спілкується з демоном Docker, який бере на себе створення, запуск, розподіл контейнерів.

Клієнт та сервер можуть працювати на одній системі, можна підключити клієнт до віддаленого демона docker. Клієнт та сервер спілкуються через сокет або через RESTful API.

Docker складається з трьох компонентів:

• Образи (images)

• Реєстри (registries)

• Контейнери (containers)

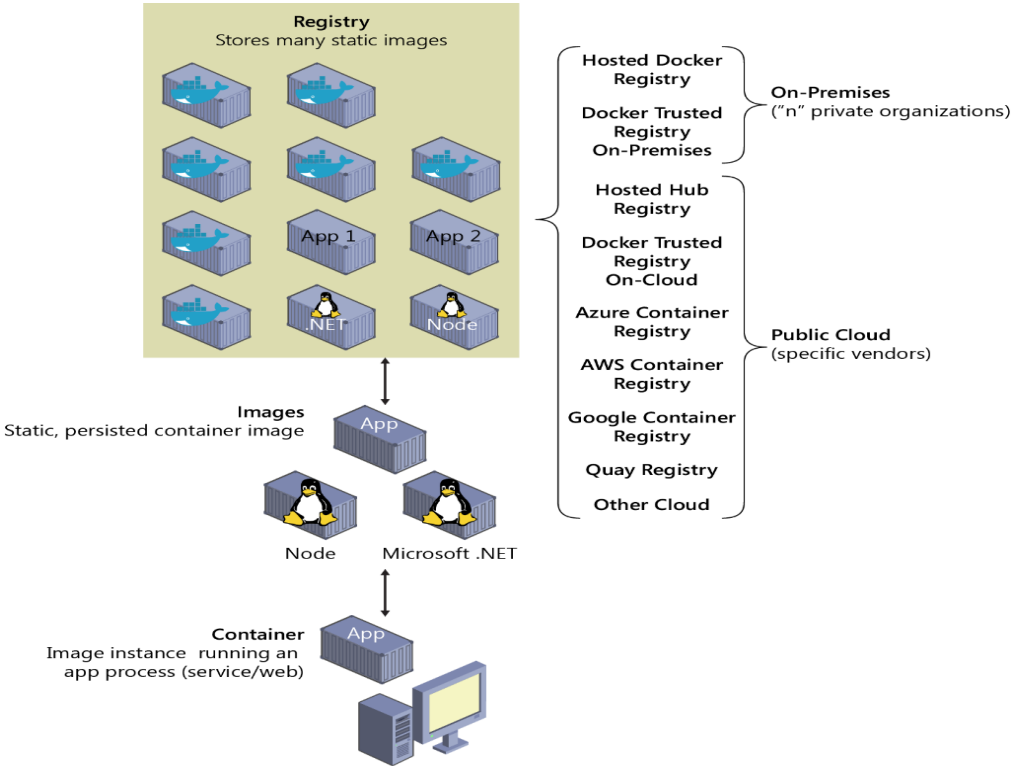


Рисунок 4. Компоненти Docker

Образи:

Docker-образ – це read-only шаблон. Наприклад, образ може містити ОС Ubuntu з Apache і додатком на ній. Образи використовуються для створення контейнерів. Docker дозволяє легко створює нові образи, обновляти вже створені, або завантажувати образи, які були створені іншими людьми. Образи – це компонента збірки docker-а.

Реєстри:

Docker-реєстри зберігають образи. Існують публічні(public) та приватні(private) регістри. У цих реєстрах можна знайти та завантажити корисні образи, щоб не створювати дублікати. Public Docker-регістр називається Docker Hub. Там зберігається неймовірна кількість образів. Образи можна створювати, або використовувати вже існуючі, які були створені іншими людьми. Реєстри називають компонентом поширення образів.

Контейнери:

Контейнери схожі на директорії. В контейнерах міститься все, що потрібно для роботи програми. Вони нагадують нам звичайні директорії операційної системи. Кожен контейнер створюється з образу. Контейнери можуть бути створені, запущені, зупинені, перенесені або видалені. Кожен контейнер ізольований і є безпечною платформою для програми. У контейнерах міститься усе необхідне, що потім буде використано для роботи програми. Контейнери – це компонента роботи.

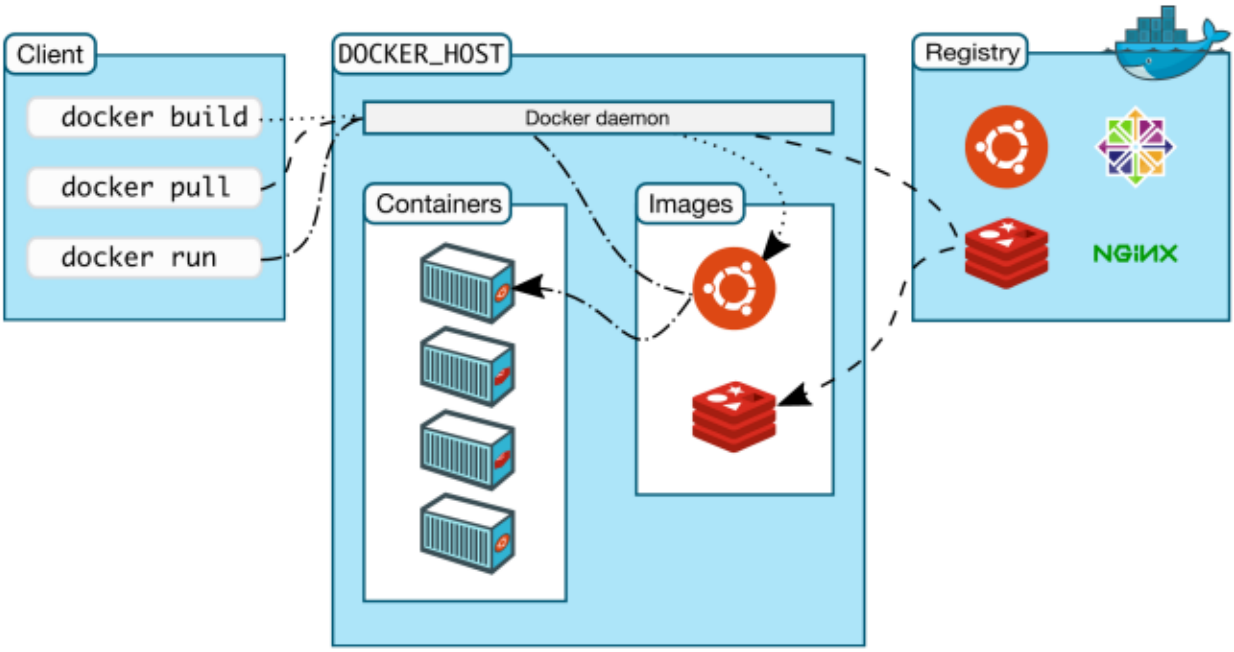


Рисунок 5. Архітектура Docker

Основними складовими частинами архітектури Docker є:

– сервер, який містить сервіс Docker, образи та контейнери; сервіс зв'язується з Registry, образи – метадані додатків, які запускаються в контейнерах Docker;

– контейнери – ізольовані за допомогою технологій ОС користувацькі оточення, в яких виконуються додатки. Контейнер Docker запускається з образу додатка. Розробники Docker дотримуються єдиного принципу: один контейнер – це один додаток;

– образи (Images) – шаблони додатків, які доступні тільки для читання. Поверх існуючих образів можуть додаватися нові рівні образів, які спільно представляють файлову систему, змінюючи або доповнюючи попередній рівень. Зазвичай новий образ створюється або за допомогою зберігання вже запущеного контейнера в новий образ, або за допомогою спеціальних інструкцій для утиліти Dockerfile. Для розділення різних рівнів контейнера на рівні файлової системи можуть використовуватися UnionFS, aufs, btrfs, vfs, OverlayFS та Device Mapper;

– REST API, що визначає інтерфейси, які програми можуть використовувати для спілкування з демоном і вказувати йому що робити;

– клієнт інтерфейсу командного рядка (CLI) (команда docker). Застосовується для запуску різних дій на сервері Docker;

– інтерфейс командного рядка використовує API-інтерфейс Docker REST для керування або взаємодії з додатками-демонами Docker з використанням базових API і CLI;

– реєстри (registry), використовуються для зберігання образів й містять репозиторії (repository) образів, тобто це мережеві сховища образів. Вони можуть бути як приватними, так і загальнодоступними.

Наведемо схему роботи Docker:

1. Користувач віддає команду за допомогою клієнтського інтерфейсу Docker-демона, розгорнутому на Docker-хості. Наприклад, скачати готовий образ з реєстру (сховища Docker-образів) за допомогою команди docker pull. Взаємодія між клієнтом і демоном забезпечує REST API. Демон може використовувати публічний (Docker Hub) або приватний реєстри.

2. Відповідно до команди, заданої клієнтом, демон виконує різні операції з образами на основі інструкцій, прописаних у файлі Dockerfile. Наприклад, виконує їх автоматичне збирання за допомогою команди docker build.

3. Робота образу в контейнері. Наприклад, запуск docker-image за допомогою команди docker run або видалення контейнера через команду docker kill.

Образи, контекст створення образу, реєстр образів

Образи Docker є результатом процесу їх збирання, а контейнери Docker – це виконувані образи. Кожному образу Docker відповідає файл, який називають Dockerfile. Його ім'я записують саме так – без розширення. Нові образи контейнерів створюють за допомогою файлів Dockerfile і команди docker build, тому важливо розуміти внутрішнє функціонування команди створення. Для команди docker build необхідно: Dockerfile і контекст створення образу (build context) (який може бути порожнім).

Контекст створення – це набір локальних файлів і каталогів, до яких можна звертатися з інструкцій ADD і / або COPY в Dockerfile і які зазвичай визначаються як шлях до потрібного каталогу.

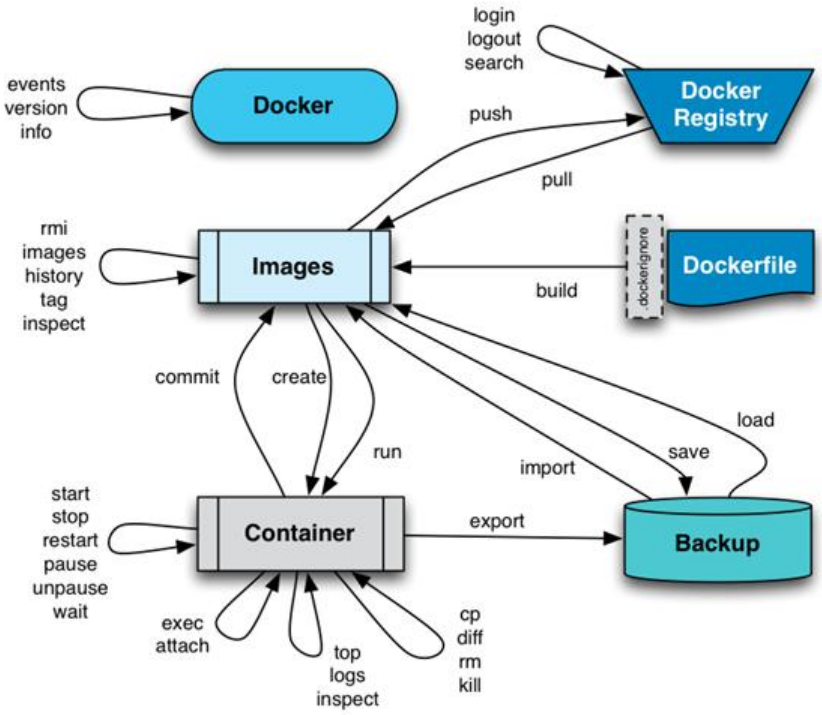
Під час запуску команди docker build для створення нового образу вважають, що Dockerfile міститься у потоковому робочому каталозі – визначено як крапка «.» (це і є контекст створення). Це означає, що усі файли і каталоги, розмішені по зазначеному шляху, формують контекст створення образу і передаються в демон Docker як частина процесу створення. Якщо цей файл перебуває в якомусь іншому місці, його розміщення можна вказати з використанням прапорця «-f».

Команди для роботи з реєстром

Команда docker login виконує процедуру реєстрації або входу на заданий сервер реєстру. Якщо сервер не задано, то за замовчуванням передбачається Docker Hub. За потреби під час цієї процедури в інтерактивному режимі буде запитуватися додаткова уточнювальна інформація у вигляді аргументів.

Команда docker logout виконує процедуру виходу з реєстру Docker. Якщо сервер не заданий, то за замовчуванням передбачається Docker Hub.

Команда docker pull завантажує заданий образ з реєстру. Реєстр визначається за іменем образу, а за замовчуванням – Docker Hub. Якщо не задане ім'я тега, буде завантажений образ з тегом latest (за наявності такого способу в певному реєстрі). Для завантаження всіх образів зі сховищ використовують аргумент «-a».



Команда docker push вивантажує образ або репозиторій у заданий реєстр. Якщо тега немає, вивантажуються всі образи зазначеного сховища в заданий реєстр, а не тільки образи з тегом latest.

Команда docker search виводить список загальнодоступних репозиторіїв з реєстру Docker Hub, які відповідають заданим шаблоном пошуку.

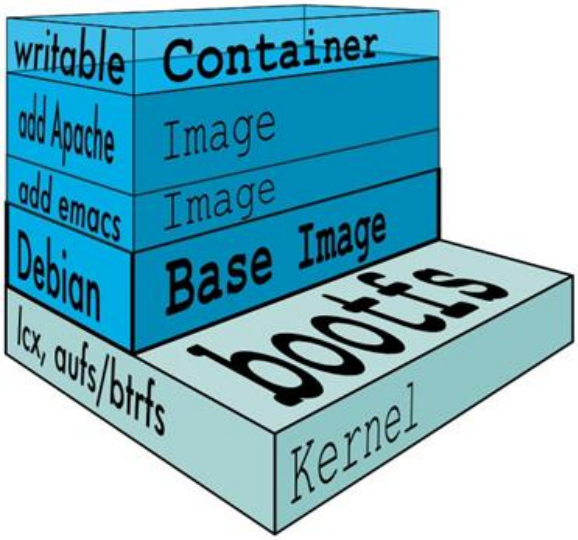
Команда docker create створює контейнер із заданого образу, але не запускає його. Аргументи цієї команди в основному ті самі, що й для команди docker run. Щоб запустити створений контейнер, потрібно виконати команду docker start.

Команда docker cp дозволяє копіювати файли між файловими системами контейнера і хоста.

Команда docker exec запускає задану команду всередині контейнера.

Команда docker commit створює образ із вказаного контейнера.

Рівні образу контейнера



Кожний Docker-образ складається з шарів (layers). Кожна інструкція в Dockerfile призводить до появи нового рівня (layer) образу. Кожний шар, крім останнього, що міститься поверх усіх інших, призначений тільки для читання. Docker відстежує кожний рівень. Образ може складатися з десятків шарів (границя дорівнює 127). Перевага шарів полягає в тому, що образи можуть спільно використовувати шари. Далі Docker об'єднує інформацію з кожного шару і створює шаблон-образ, з якого запускається контейнер, в якому виконуються інструкції з кожного шару, який був включений у цей образ.

Життєвий цикл ПЗ з використанням контейнера Docker полягає у проходженні таких стадій:

docker create – створення контейнера без його запуску, може бути використана для зберігання і виведення ідентифікатора контейнера для майбутнього використання;

docker run – робота контейнера: створити і запустити контейнер;

docker stop – призупинення роботи контейнера;

docker start – відновлення роботи контейнера: запустити наявний зупинений контейнер;

docker restart – перезапуск контейнера;

docker rm – видалення контейнера;

docker kill – відправити сигнал SIGKILL контейнеру про завершення роботи;

docker attach – підключитися до працюючого контейнера;

docker wait – блокувати команду і чекати, поки контейнер не зупиниться.

Переваги використання Docker.

1. Мінімальне використання ресурсів – контейнери не віртуалізують всю операційну систему, а використовують ядро хоста і ізолюють програму на рівні процесу. Останній використовує набагато менше ресурсів, ніж віртуальна машина.

2. Швидкісне розгортання – допоміжні компоненти можна не встановлювати, а використовувати вже готові docker-образи. Наприклад, немає сенсу постійно встановлювати і налаштовувати Linux Ubuntu. Достатньо один раз її інсталювати, створити образ і постійно його використовувати, обновляючи версію при необхідності.

3. Зручне приховування процесів – для кожного контейнера можна використовувати різні методи обробки даних, приховуючи фонові процеси.

4. Робота з небезпечним кодом – технологія ізоляції контейнерів дозволяє запускати будь-який код без шкоди для ОС.

5. Просте масштабування – будь-який проект можна розширити, впровадивши нові контейнери.

6. Зручний запуск – програму, яка знаходиться всередині контейнера можна запустити на будь-якому docker-хості.

7. Оптимізація – файлової системи – образ складається з шарів, які дозволяють дуже ефективно використовувати файлову систему.

Питання для самоконтролю

1. Що таке контейнер, яке його призначення?
2. У чому полягають відмінності між віртуальними машинами та контейнерами?
3. У чому полягають переваги та недоліки контейнерів?
4. Які технології використовують під час створення контейнерів?
5. Що таке образ контейнера та з чого він складається?
6. Що таке репозиторій?
7. Які основні компоненти Docker?
8. Що таке реєстри? Яке їх призначеня?
9. Які вам відомі інструментальні засоби для Docker?
10. Що таке Dockerfile?
11. Які команди використовують для роботи з реєстрами?
12. Що таке рівні образу контейнера?
13. Які вам відомі команди щодо керування контейнерами?
14. Як створити образ контейнера?
15. Як переглянути активні та неактивні контейнери?
16. Які команди використовують під час роботи з Dockerfile?
19. Як забезпечити обмін інформацією між контейнерами на одному хості?
20. У чому полягає життєвий цикл програмного забезпеченні з використанням контейнера Docker?
23. Як запустити конейнер в автономному режимі?
24. Як запустити контейнер Docker за допомогою створеного образу?