*Лекція 11.*

*Контейнеризація, також відома як наповнення або завантаження контейнерів, є відносно новим поняттям у сфері розробки програмного забезпечення. З появою різноманітних середовищ розгортання, особливо пов'язаних з хмарними обчисленнями, вона набула значної популярності.*

*У цій статті ми розповіли, чому виникла віртуалізація на рівні операційної системи, як працює та які переваги надає розробникам та бізнесу.*

**Що таке контейнеризація**

**Контейнеризація** ― це розгортання програмного забезпечення разом з усіма необхідними компонентами: код, бібліотеки, фреймворки тощо, таким чином, щоб вони були ізольовані у власному контейнері.

**Контейнер** — це ізольоване місце, де програма працює, не впливаючи на решту системи та без впливу системи на програму. Оскільки вони ізольовані, контейнери добре підходять для безпечного запуску програмного забезпечення, наприклад баз даних або веб-додатків, яким потрібен доступ до конфіденційних ресурсів без надання доступу кожному користувачеві системи. 16 Контейнерне програмне забезпечення, доступне як для додатків Linux, так і для Windows, завжди працюватиме однаково, незалежно від інфраструктури. Додатки також можуть працювати в будь-якій інфраструктурі та в будь-якій хмарі. Ви можете ізолювати програми та їх базову інфраструктуру від інших програм.

Якщо взяти традиційні методи розробки ПЗ, то цей процес відбувається в конкретному обчислювальному середовищі. Якщо його перенести в інше місце, наприклад, з ПК у хмару чи з Linux на Windows, може виникнути низка помилок та багів. За допомогою контейнеризації можна об’єднати всі складові програмного забезпечення у так званий контейнер. Він з усім вмістом абстрагується від хостової операційної системи, є автономним і може без проблем працювати на будь-якій платформі.

Щось подібне відбувається під час переїзду на іншу квартиру. Ви складаєте речі у коробки, при чому посуд в одну, одяг ― в іншу, книги ― в третю і так далі. Тобто всі ці речі ізольовані одна від одної. Під час переїзду коробки легко переміщати з місця на місце, при чому без необхідності розпакування та перезапакування кожного предмета окремо. Так і контейнери у технології контейнеризації дозволяють зручно розгортати та переміщати програми з одного середовища в інше без необхідності перекомпілювання або налаштування.

**Чому виникла ідея контейнеризації**

Головна причина розробки технології контейнеризації — поява складних і високонавантажених IT-інфраструктур. Для їхньої повноцінної роботи необхідно забезпечити надійність кожного окремого елемента та мінімізувати їхній вплив один на одного. Потрібна операційна система та ПЗ, які зможуть «завертати» ці елементи в контейнери, дробити великі та монолітні ІТ-інфраструктури на безліч дрібних, незалежних елементів, взаємодіючих між собою як одне ціле.

Ідея ізоляції процесів існує багато років. У 2013-му Docker представив Docker Engine та тим самим встановив стандарт використання контейнерів з простими у використанні інструментами, а також універсальний підхід до пакування, що прискорило впровадження контейнерної технології. Влітку 2014 року запустили інструмент для автоматизації управління контейнерами (оркестрації) — [Kubernetes](https://gigacloud.ua/blog/navchannja/scho-take-kubernetes-i-dlja-chogo-vin-potriben).

**Контейнерна оркестровка** — це автоматизація більшої частини операційних зусиль, необхідних для виконання контейнерних робочих навантажень і служб. Це включає в себе широкий спектр речей, необхідних командам програмного забезпечення для керування життєвим циклом контейнера, включаючи надання, розгортання, масштабування (вгору та вниз), мережу, балансування навантаження тощо. Оркестровка контейнерів є ключовою для роботи з контейнерами, і це дозволяє організаціям розблокувати всі їхні переваги.

**Як працює контейнеризація**

Архітектура контейнеризації має декілька шарів:

* **Інфраструктура**. Це апаратний рівень. Належить до фізичного комп’ютера чи сервера без встановленої операційної системи, на якому виконується контейнерний додаток.
* **ОС**. Серед найпопулярніших операційних систем для контейнеризації є Linux. Працює як на локальній інфраструктурі, так і в хмарі.
* **Образи контейнерів**. Розробники ПЗ створюють і розгортають образи контейнерів, тобто файли, що містять необхідну інформацію для запуску контейнерного додатка. Відбувається це на основі Open Container Initiative (OCI). Образи контейнерів доступні тільки для читання і не можуть бути змінені комп'ютерною системою.
* **Контейнерний рушій**. Це програмне забезпечення, яке створює контейнери на основі їхніх образів. Він виступає як посередник між контейнерами та операційною системою, надаючи ресурси, необхідні застосунку, і керуючи ними. Наприклад, контейнерні рушії можуть керувати кількома контейнерами в одній операційній системі, зберігаючи їхню незалежність від базової інфраструктури та один від одного.
* **Додаток і залежності**. Найвищий рівень архітектури контейнеризації. Це код застосунку та інші файли, які необхідно запустити: залежності бібліотек і пов'язані з ними файли конфігурації. Цей шар також може містити гостьову операційну систему, яка встановлюється поверх основної.

**Переваги контейнеризації**

* **Переносимість**. Контейнери можуть працювати в різних обчислювальних середовищах: локальне обладнання, віртуальні машини або хмарні платформи.
* **Ізоляція**. Програми у контейнерах працюють незалежно одна від одної, що підвищує безпеку та стабільність, запобігає конфліктам та забезпечує середовище «пісочниці».
* **Безпека**. Ізоляція додатків запобігає проникненню шкідливого вірусу в інші контейнери або хост-систему. Крім того, можна визначити дозволи безпеки, щоб автоматично блокувати несанкціонований вхід в контейнери або обмежити зв'язок з непотрібними ресурсами.
* **Ефективне використання ресурсів**. Контейнери легкі та використовують ядро хост-системи, що дозволяє ефективно користуватися ресурсами. Декілька контейнерів можуть працювати одночасно на одному хості, ефективно розподіляючи ресурси та оптимізуючи використання апаратного забезпечення.
* **Масштабованість**. Контейнеризація полегшує горизонтальне масштабування, дозволяючи розгортати та масштабувати контейнери на декількох хостах або кластерах. Їх також можна легко реплікувати та організовувати за допомогою інструментів управління, таких як Kubernetes, що дозволяє додаткам масштабуватися на вимогу, забезпечуючи високу доступність та швидкість реагування.
* **Гнучкість та швидке розгортання**. Контейнери забезпечують швидке розгортання додатків, полегшуючи пакування додатків, залежностей та конфігурацій в єдине ціле.
* **Контроль версій та співпраця**. Розробники можуть працювати над різними компонентами програми незалежно у своїх контейнерах, забезпечуючи сумісність і зменшуючи кількість конфліктів. Такий контроль версій та спільна робота підвищує продуктивність і полегшує впровадження DevOps-практик.

**Недоліки**

**С**кладність розробки. Якщо вам потрібне швидше рішення (інший тип, невелика програма, стислі терміни), то мікросервіси можуть не підійти. Швидкість розробки – висока плата за ціну та роздрібленність. 11 Складність підтримки. Кожен мікросервіс потребує свого обслуговування, тому потрібен постійний ручний або автономний моніторинг.

**Найпопулярніші інструменти, які використовуються у контейнеризації**

**Docker**

Docker, або Docker Engine ― це популярне середовище для контейнеризації з відкритим вихідним кодом, що дає змогу розробникам створювати, розгортати та тестувати контейнерні додатки на різних платформах.

**Linux Containers або LXC**

Контейнери Linux дають змогу запускати декілька додатків на базі ОС Linux на одному хост-комп'ютері. Розробники використовують їх для розгортання додатків, які записують або зчитують великі обсяги даних. Вони не копіюють усю операційну систему у віртуалізоване середовище, а складаються з необхідних функцій, виділених у просторі імен Linux.

**Kubernetes**

Kubernetes ― популярна платформа з відкритим вихідним кодом від Google, яку використовують для розгортання, масштабування величезної кількості мікросервісів і управління ним. Вона має декларативну модель, яка спрощує автоматизацію контейнерів.

**VMware Tanzu**

Це портфель продуктів VMware, орієнтований на управління сучасними хмарними середовищами на основі контейнерів. За допомогою [VMware Tanzu](https://gigacloud.ua/blog/navchannja/vmware-tanzu-vse-pro-kubernetes-na-vmware" \o "" \t "_blank) користувачі можуть розробляти, підтримувати та запускати контейнерні додатки в кластерах Kubernetes у приватних та публічних хмарах на VMware.

Аналіз рішення Docker Docker — це платформа контейнеризації з відкритим вихідним кодом, за допомогою якої можна автоматизувати створення програм, їх доставку та керування. Платформа дозволяє швидше тестувати та викладати програми, запускати на одній машині необхідну кількість контейнерів. Завдяки Docker можна легко запускати контейнер у хмарній інфраструктурі та на будь-якому локальному пристрої. Можна створити базові шаблони контейнерів і повторно використовувати нескінченну кількість разів. Легка переносимість та простота розгортання – важливі переваги цієї технології. Docker дозволяє використовувати будь-які мови програмування та стек технологій на сервері Контейнери ізольовані та можуть мати спільне програмне забезпечення, бібліотеки та файли конфігурації. Вони можуть спілкуватися за допомогою внутрішньої мережи, яка має більше 5 разних варіантів нашалтування. Оскільки всі контейнери спільно використовують служби одного ядра операційної системи, використовують менше ресурсів, ніж віртуальні машини, тому запуск великої кількості контейнерів на одній машині не несе ризиків.

**Основні компоненти Docker**

– Dockerfile – текстовий файл із послідовно розташованими інструкціями для створення образу Docker. Файл створюється за принципом один рядок - одна команда.

– Daemon – фонова служба на хості, яка відповідає за створення, запуск та знищення контейнерів.

– Image – незмінний файл (образ), з якого можна необмежену кількість разів розгорнути контейнер.

– Container – запущена програма, яку розгорнули з образу

– Registry – служба Docker, що виконує функції репозиторію (сховища). Дозволяє стежити за версіями образів, створювати приватні репозиторії.

– Docker Hub – популярний публічний репозиторій, який використовується за замовчуванням у Docker. Забезпечує інтеграцію з системою VCS

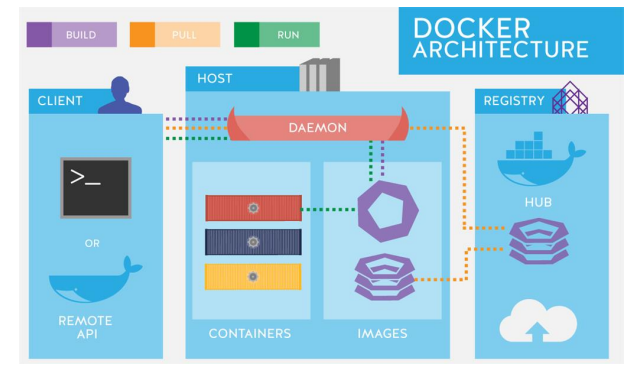


Рисунок 1 − Архітектура технології Docker

Що, якщо замість традиційних методів управління серверами та ресурсами в хмарі можна створювати, змінювати та керувати всією інфраструктурою через чітко написані та керовані конфігураційні файли? “Тerraform – що це?”. Це питання все частіше звучить у світі сучасних технологій.

Цей інструмент здатний перетворити ваш підхід до управління інфраструктурою. Але що робить його настільки унікальним і затребуваним у сфері IT? Давайте поглянемо на цей інструмент ближче…

Terraform – це програмне забезпечення з відкритим вихідним кодом, створене і підтримуване компанією HashiCorp. Його основна мета – надати можливість керувати інфраструктурою як кодом (IaC). Але що таке інфраструктура як код? Це парадигма управління IT-інфраструктурою, в якій вся необхідна конфігурація і параметри визначаються і управляються програмним чином, через спеціальні файли конфігурації. Замість ручного втручання адміністратора, вся інфраструктура стає керованою кодом. Це дає змогу автоматизувати процеси розгортання, налаштування та керування серверами, мережами, базами даних та іншими ресурсами.

Стосовно Terraform, IaC означає, що ви можете створювати, змінювати й керувати інфраструктурою вашого застосунку, використовуючи декларативну мову конфігурації. Такий підхід гарантує, що інфраструктура буде передбачуваною, надійною і масштабованою, а процес розроблення та управління – набагато ефективнішим і зручнішим.

## Історія створення та розвитку

З появою хмарних обчислень і масштабного розвитку віртуалізації, компанії зіткнулися з новими викликами. Управління всією інфраструктурою вручну стало трудомістким процесом, схильним, до того ж, до помилок. Тому, ручне налаштування кожного сервера, мережі та інших ресурсів стало непрактичним, особливо у великих і складних середовищах.

Саме в цей момент виникла потреба в інструменті, який дав би змогу описувати й керувати інфраструктурою у вигляді коду. Terraform був створений для того, щоб зробити цей процес автоматизованим і керованим.

Нагадаємо основні етапи розвитку:

* Версія 0.1 (2014 рік): перша версія і перший крок до створення універсального інструменту для управління інфраструктурою як кодом. На той момент він підтримував обмежений набір хмарних провайдерів і базові функції.
* Розвиток екосистеми (2015-2017 роки): інструмент став швидко набирати популярність завдяки своїй простоті та гнучкості. За цей час було додано нових провайдерів, даючи змогу взаємодіяти з різними хмарними та інфраструктурними сервісами. Також з’явилися модулі, які дають змогу перевикористовувати код і спрощують управління великими конфігураціями.
* Версія 0.9 і 1.0 (2017 рік): ця версія внесла безліч поліпшень у Terraform модулі, включно з розширеними можливостями управління станом інфраструктури. 1.0 стала версією, що позначила початок більш стабільного і надійного шляху розвитку.
* Terraform Cloud (2018 рік і далі): це хмарний сервіс, що надає можливість спільного використання конфігурацій, управління станом та інші інструменти для командної роботи. Це посилило позиції Terraform як ключового інструменту в хмарному управлінні.
* Останні оновлення та перспективи (останні роки): Terraform продовжує розвиватися, додаючи нові функції та інтеграції, як-от покращена робота з Kubernetes і надійніше управління станом. Водночас він продовжує підтримувати велику спільноту розробників, які активно роблять свій внесок у розвиток проєкту.

Таким чином, Terraform продовжує еволюціонувати, зберігаючи свою репутацію надійного і гнучкого інструменту, і продовжує залишатися одним з найпопулярніших інструментів у своєму класі.

## Terraform основи

Розглянемо деякі ключові принципи, на яких будується терраформ. Це – фундамент, який дає нам розуміння того, як керувати інфраструктурою нашого проєкту.

* **Конфігураційні файли**

Являють собою текстові файли, написані мовою HashiCorp Configuration Language (HCL) або у форматі JSON. У цих файлах описується бажаний стан інфраструктури, включаючи сервери, мережі, бази даних та інші ресурси. Цей опис у вигляді коду визначає, який вигляд і як має працювати ваша інфраструктура. У цих файлах ви вказуєте, які ресурси вам потрібні, як вони мають бути налаштовані і як вони взаємодіють один з одним. Terraform використовує ці файли як вихідні дані для створення та управління ресурсами у вашій інфраструктурі.

* **Планування та застосування змін**

Перед тим, як Terraform внесе будь-які зміни у вашу інфраструктуру, він створює план (plan), який детально описує, що саме буде змінено і як це вплине на поточну інфраструктуру. Це важливий етап, який дає змогу вам розуміти, які ресурси буде створено, змінено або видалено.

Після цього зміни застосовуються до вашої інфраструктури. Це забезпечує контроль і безпеку, оскільки ви бачите всі зміни, які будуть внесені, перш ніж вони фактично відбудуться.

* **Стан інфраструктури**

Це файл (зазвичай у форматі JSON), який містить інформацію про поточний стан ваших ресурсів і зв’язків між ними. Зберігається локально на вашому комп’ютері або в хмарному сховищі, залежно від того, як налаштоване ваше середовище. Стан у Terraform – це внутрішній механізм, який дає змогу відстежувати, які ресурси вже існують в інфраструктурі та як вони були налаштовані. Це позбавляє від необхідності вручну відстежувати стан кожного ресурсу і дозволяє визначити, які зміни потрібно внести, щоб досягти бажаного стану, описаного в конфігураційних файлах.

Усі ці принципи забезпечують надійне і передбачуване управління вашою IT-інфраструктурою.

## Переваги

Поговоримо про те, чому Terraform стає невід’ємним компонентом для багатьох професіоналів у галузі інформаційних технологій.

* **Масштабованість і гнучкість**

Однією з ключових переваг тераформ є його масштабованість і гнучкість. Це дає змогу описувати й керувати інфраструктурою будь-якого розміру та складності. Незалежно від того, чи потрібні вам десятки віртуальних машин або складні мережеві конфігурації, цей інструмент допомагає створювати та змінювати ресурси відповідно до ваших потреб. Завдяки цій масштабованості – ідеально підходить для невеликих стартапів, а також для великих корпорацій з величезними і складними IT-інфраструктурами.

* **Підтримка безлічі хмарних провайдерів**

Ще однією значною перевагою є здатність цього ПЗ працювати з безліччю хмарних провайдерів. Незалежно від того, чи використовуєте ви Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) або інші хмарні платформи, він забезпечує єдиний і зручний спосіб управління вашою інфраструктурою. Це дає змогу створювати гібридні інфраструктури, комбінуючи ресурси з різних хмарних і локальних джерел.

* **Суспільство та екосистема**

Має активну та широку спільноту розробників і користувачів, які обмінюються знаннями, створюють корисні модулі та вирішують складні проблеми разом. Приєднавшись до цієї спільноти, розробники можуть отримати підтримку, поради та ідеї для оптимізації своїх конфігурацій і поліпшення продуктивності.

Крім того, Terraform має багату екосистему плагінів і модулів, що надаються як самою HashiCorp, так і зовнішніми розробниками.

## Практичне використання

* **Створення та керування віртуальними машинами**

Одним із поширених сценаріїв використання Terraform є створення та управління віртуальними машинами в хмарних середовищах. Припустимо, у вас є проєкт, що вимагає кілька віртуальних машин для хостингу програми. Ось як ви можете це зробити з Terraform:

1. **Визначення конфігурації:**

Створіть конфігураційний файл, наприклад main.tf, у якому визначте параметри віртуальних машин, такі як типи, кількість, образи та інші параметри. Приклад:

provider "aws" {

  region = "us-west-2"

}

resource "aws\_instance" "example" {

  ami           = "ami-0c55b159cbfafe1f0"

  instance\_type = "t2.micro"

  count         = 3

  tags = {

    Name = "example-instance-${count.index}"

  }

}

У цьому прикладі використовується провайдер AWS. Він створює три віртуальні машини типу t2.micro з певним образом.

1. **Ініціалізація:**

У командному рядку виконайте команду terraform init у директорії, де знаходиться ваш конфігураційний файл. Це завантажить необхідні плагіни та налаштує робоче середовище.

1. **Планування та застосування:**

Після успішної ініціалізації виконайте terraform plan для перегляду плану змін. Після перевірки, що все коректно, виконайте terraform apply для створення віртуальних машин.

1. **Управління інфраструктурою:**

Ви можете використовувати Terraform для зміни кількості віртуальних машин, їхні типи та інші параметри. ПЗ виявить різницю між поточним станом і бажаним станом (визначеним у конфігураційних файлах) і застосує необхідні зміни.

* **Налаштування мережевої інфраструктури**

Terraform також дає змогу легко налаштовувати мережеві аспекти вашої інфраструктури. Припустимо, ви хочете створити віртуальну приватну мережу (VPC) і налаштувати в ній підмережі та правила фаєрвола:

1. **Визначення мережевої конфігурації:**

У конфігураційному файлі визначте вашу VPC, підмережі та правила фаєрвола. Приклад:

resource "aws\_vpc" "example" {

  cidr\_block = "10.0.0.0/16"

}

resource "aws\_subnet" "example\_subnet" {

  count = 2

  vpc\_id     = aws\_vpc.example.id

  cidr\_block = "10.0.1.0/24"

}

resource "aws\_security\_group" "example\_security\_group" {

  vpc\_id = aws\_vpc.example.id

  ingress {

    from\_port   = 80

    to\_port     = 80

    protocol    = "tcp"

    cidr\_blocks = ["0.0.0.0/0"]

  }

}

Цей приклад створює VPC з двома підмережами і дозволяє вхідний трафік на порт 80 з будь-якого джерела.

1. **Планування та застосування змін:**

Після визначення конфігурації виконайте terraform plan і terraform apply так само, як і для віртуальних машин. Terraform створить і налаштує VPC, підмережі та правила фаєрвола відповідно до ваших вказівок.

1. **Управління:**

Подібно до віртуальних машин, ви можете вільно змінювати параметри, як-от підмережі та правила фаєрвола, шляхом редагування конфігураційних файлів і впровадження необхідних змін.