ВСТУП

Інформаційні технології сьогодення дають змогу використовувати різні застосунки і бібліотеки у великому різноманітті середовищ. І хоча це являє собою гарний можливий ґрунт для клієнта та конкурентоспроможності, однак це і може бути складністю у подальшому розповсюдженні і використанні.

Виникає кілька проблем. Перша – це проблема розгортки на великій кількості різних робочих машин. У кожної має бути своя конфігурація, апаратні та програмні засоби, і індивідуальне налаштування кожної може відняти дуже багато часу та ресурсів.

Ще одна така проблема – це складність у передачі налаштувань клієнту, чи їх виправлення із необхідністю прямого контакту із ним. Для вирішення цього пишуть різноманітні гайди-інструкції, або скрипти задля більшої автоматизації процесу. Але це додає багато людського фактору, особливо якщо клієнт не дуже впевнено користується ПК та може припуститись помилок у інструкції чи скрипті.

Для вирішення цих та інших проблем часто користуються віртуальними машинами(VM). Однак цей вибір також має свої недоліки. Відразу «з коробки» віртуальна машина потребує багато вільного місця. Крім цього, після будь-яких внесених змін в образ машини її необхідно перезавантажувати наново, що також ускладнює їх використання. Серверне ж управління спільними ресурсами подекуди унеможливлюється через відсутність підтримки машиною спільного користування пам’яті та ресурсів процесора.

У цій магістерській роботі буде розглянуто більш сучасний спосіб вирішення цих проблем, а саме контейнеризацію. Вона полягає у розгортанні лише потрібних процесів у раніше налаштованому образі ОС. Та всі процеси протікають на рівні ядра машини, розподіляючи усі доступні CPU та всю ОЗП. Це є «золотою серединою» між віртуалізацією та прямого розгортання на сервері.

Протягом роботи буде проведено порівняння із віртуальними машинами такого методу контейнеризації як Docker, докладно розглянуті його переваги та недоліки, а також доцільність використання його з метою спрощення поставлених задач перед звичайним системним користувачем чи інженером.

1. РОЗГЛЯД ЗАДАЧІ КОНТЕЙНЕРИЗАЦІЇ

1.1 Технології віртуалізації

Поняття віртуалізація має безліч визначень, в даному випадку розуміються інструменти і підходи, які дозволяють абстрагуватися від апаратної основи обчислювальної системи. Робота з системою на більш високому рівні абстракції дозволяє більш гнучко працювати з ресурсами. Для прикладу розглянемо ситуацію роботи з деяким сервером. При відсутності віртуалізації, в разі виходу сервера з ладу, для продовження роботи, необхідно поставити новий сервер, конфігурувати його і провести заміну старого сервера на новий. Так як нова апаратура повністю незалежна від попередньої, то немає гарантій, що все налаштовано правильно і буде працювати так само як і раніше. У разі, якщо сервер є деякою віртуальної сутністю, то після виходу з ладу його можна підмінити копією, створеної на іншій апаратній основі, якщо це необхідно. При цьому віртуалізацію можна також використовувати для зменшення навантаження на сервер. У випадку з віртуалізацією є можливість оперативного динамічного розподілу навантаження на копії вихідного сервера. У випадку з відсутністю віртуалізації це важко реалізувати. Це тільки деякі з прикладів. В реальності існує безліч ситуацій, коли потрібне певне налаштоване середовище.

При наявності віртуалізації розгортання та налаштування нового оточення займає набагато менше часу. При цьому велика частина нового елемента буде повністю ідентична старому. Віртуалізація ділиться на наступні класи:

1.1.1 Повна віртуалізація (Full Virtualization).

Крім цього терміна в деяких роботах використовується термін hardware emulation. В рамках цієї технології використовується система управління віртуальними машинами (гіпервізор), яка управляє віртуальними машинами. При цьому відбувається повна ізоляція віртуальної машини від машини, на якій проводиться її запуск. Всі елементи віртуальної машини емулюються і система управління відповідає за виконання команд, емуляцію пристроїв і взаємодія між машинами. Деякі інструменти

дозволяють прискорити продуктивність, наприклад за рахунок кешування однакових інструкцій.

1.1.2 Часткова віртуалізація (Paravirtualization).

Як і в попередньому випадку дана технологія приймає до уваги систему управління. Однак, крім повної віртуалізації вона вносить зміни в операційну систему, на якій запускається. Це дозволяє оптимізувати процес виконання для більш високої продуктивності в порівнянні з повною віртуалізацією. Перевагою даного підходу є те, що всі пристрої залишаються повністю віртуалізованими. Недоліком підходу є те, що необхідно вносити зміни в операційну систему. У разі, коли вихідний код операційної системи закритий, це може привести до проблем.

1.1.3 Віртуалізація на рівні операційної системи.

На відміну від попередніх підходів, при використанні даної технології основних інструментів для роботи з віртуальними одиницями є ОС. Яка в свою чергу створює ізольовані екземпляри деякої операційної системи. Одне ядро керує ресурсами всіх примірників. Гостьова операційна система часто пов'язана з Virtual private servers (VPS). Перевагою даного підходу є ще більший приріст продуктивності в порівнянні з попередніми підходами. Відсутністю системи управління і відсутністю необхідності модифікації операційної системи. При цьому отримані показники продуктивності в цілому близькі до показників отриманими на вихідній машині.

Крім цього, використання одного ядра для всіх екземплярів зменшує кількість ресурсів, необхідних для підтримки декількох екземплярів. Головним недоліком є те, що при виході з ладу вузла, що відповідає за управління всіма примірниками, вони також виходять з ладу.

1.1.4 Нативна віртуалізація (Native virtualization).

Даний підхід являє собою наявність підтримки інструментів віртуалізації у апаратних пристроїв. Мета полягає в тому, щоб скоротити витрати при віртуалізації. Він дозволяє запускати безліч завдань, які працюють спільно і виконуються

безпосередньо на процесорі без емуляції. Виграш в продуктивності особливо помітний, якщо ми порівнюємо з випадком, коли відбувається емуляція іншою версією процесора.

Поточні версії процесорів підтримують цю технологію.

1.2. Віртуалізація ресурсів.

До даного класу відноситься об'єднання ресурсів. У тому числі й кластеризацію обчислювальних ресурсів, для об'єднання їх в єдину обчислювальну систему, з метою використання, наприклад, для розподілених обчислень. Крім цього, даний підхід включає в себе консолідацію серверної частини, тим самим дозволяючи виконувати завдання користувача паралельно, і збільшуючи продуктивність системи. Віртуалізація особливо успішна, якщо досягнуто досить високий рівень інкапсуляції і користувач працює з системою, як з єдиним цілим, не опускаючись на нижчі рівні взаємодії з системою.

Існує безліч аспектів віртуалізації в цілому. Почнемо з інструментів, які використовуються для віртуалізації мережі. Далі проведений огляд існуючих рішень для моделювання мережі, зокрема:

1 Mininet;

2 Cisco Packet Tracer;

3 GNS3;

4 NetBoxIT;

Використання даних програмних продуктів дозволяє змоделювати роботу мережі. Проводити тестування і налаштування використовуючи різні топології мережі і різні мережеві параметри. Незважаючи на загальну основу, вони мають різні цілі. При цьому всі рішення мають загальний недолік: вони спрямовані на моделювання мережі на локальному комп'ютері. При використанні їх в складі кластера виникають проблеми інтеграції. Нижче коротко розглянуто кожен з них.

1.2.1 Mininet.

Дозволяє працювати з software-defined networking (SDN) мережами. Відмінною особливістю даних мереж є те, що окремо існує рівень управління. Відповідно до даного підходу вся топологія мережі, побудови маршрутів передачі даних і в цілому вся логіка обробки даних в мережі відбувається на даному рівні.

Пристрої, які безпосередньо відповідають за передачу даних займаються тільки виконанням команд, що приходять з рівня управління і, по факту, займаються тільки передачею даних.

Mininet дає набір засобів для тестування і налагодження роботи SDN мереж. Використовуючи його є можливість розробити і налагодити контролер управління мережею і зробити моделювання роботи мережі з використанням даного контролера. Що дозволяє оцінити застосовність використання даної технології і зробити оцінку необхідності впровадження. В даний момент Mininet використовується в Стенфорді, Прінстоні та інших провідних вузах для вирішення практичних завдань в навчальних курсах по комп'ютерних мережах.

Недоліком є те, що Mininet емулює тільки те, що стосується безпосередньо мережі. Всі решта ресурсів - це ресурси локальної машини на якій відбуваються обчислення.

1.1.2 Cisco Packet Tracer.

Дозволяє емулювати більшу частину елементів мережі. При цьому даний продукт добре використовувати для початківців, наприклад.

При подальшому ускладненні архітектури мережі даний програмний продукт перестає задовольняти деяким критеріям. Труднощі виникають уже на етапі проектування VPN, а при детальному описі стає повністю не застовуваним. Для моделювання більш складних систем використовують наступний продукт.

1.1.3 Graphical Network Simulator 3 (GNS3)

Він не концентрується на якійсь конкретній технології і дозволяє емулювати локальну мережу в цілому. Він містить в собі функціональність попереднього програмного продукту і розширює її, охоплюючи більшу частину технологій. Дозволяє не тільки моделювати мережі, але також і управляти зовнішніми апаратними засобами. До всього іншого є безкоштовно поширюваним програмним забезпеченням з відкритим вихідним кодом. На жаль, у даного продукту є ряд недоліків. По-перше, його можливості породжують складності в застосуванні. Використовуючи попередній інструмент є можливість абстрагуватися від деяких елементів в мережі.

В даному ж випадку мережа розглядається детально, що вимагає наявності певного досвіду в цій сфері. По-друге вільно розповсюджується, часто, має низку проблем пов'язаних зі стабільністю роботи. На жаль, даний програмний продукт не є винятком.

1.2.4. NetBoxIT

Дозволяє емулювати різнорідні мережі, зокрема його можна використовувати для моделювання мережі аварійних служб, що включає в себе всі пристрої від портативних пристроїв у співробітників до взаємодії з супутниками. Однак, незважаючи на докладний опис принципів роботи і результати проведених експериментів, даний продукт не знаходиться у відкритому доступі, в зв'язку з цим немає можливості провести його повноцінний огляд через нестачу матеріалу.

Підводячи підсумок, існує безліч способів віртуалізації мережевих ресурсів. У даній роботі розглянуті деякі з них. Велика їх частина емулює роботу мережі на локальному комп'ютері, не задіюючи зовнішні ресурси. Це дозволяє спроектувати взаємодію мережевих елементів, однак використання зовнішніх апаратних пристроїв важко. Інструменти, які володіють необхідною функціональністю містять ряд інших недоліків, які ускладнюють процес їх використання.

В даний момент в якості основи побудови розподіленої системи можна використовувати такі елементи:

- Апаратний вузол.

Даний підхід має на увазі, повна відсутність віртуалізації і використання тільки апаратних засобів. Застосовується у випадках, коли кількість вузлів не велике і однорідно. У випадках, коли кількість апаратних засобів велике їх стає важко налаштовувати і конфігурувати. Крім цього, необхідно контролювати роботу користувачів і підтримувати ізольованість. Коли підтримувати апаратне рішення стає проблематично, використовують засоби віртуалізації і оперують з технологіями представленими нижче.

- Віртуальна машина.

Використовуючи дану технологію відбувається програмно-апаратна емуляція для роботи деякої платформи. Всі взаємодії з апаратними ресурсами відбувається через гіпервізор. Даний підхід використовує повну віртуальну емуляцію апаратної системи, що дозволяє виконувати на ній різні операційні системи.

При цьому існують як рішення для кластерів, так і для окремих пристроїв. Існує ряд інструментів, які дають можливість працювати з ними, зокрема:

- Virtualbox;
- VMware;
- KVM;
- Xen;
- та інші

Одним з популярних рішень є VMWare vSphere, на його основі можна побудувати два види кластерів:

- HA (High-availability) кластер;
- DRS (Distributed Resource Scheduler) кластер.

Різниця даних видів полягає в тому, що в першому випадку наголос виробляється на доступність вузлів кластера. І при виході необхідного вузла з ладу створюється новий, який є його копією вузол.

Після чого всі запити, адресовані на впавший вузол перенаправляються на копію. Для контролювання процесу створення копій можна використовувати VMware Fault Tolerance. Однак дане рішення має низку вимог, які можуть ускладнити процес його використання або змусити від нього відмовитися взагалі.

У другому випадку всі ресурси кластера об'єднуються в єдину обчислювальну одиницю і здійснюється безліч процесів, необхідних для підтримання ефективної та постійної роботи. Як приклад можна привести автоматичний розподіл навантаження на апаратну складову кластера. Для більш ефективної роботи використовуються додаткові засоби, наприклад технологія vMotion. Перевага даної технології полягає в тому, що вона дозволяє зробити міграцію віртуальної машини з одного сервера на інший. При цьому немає необхідності зупиняти віртуальну машину або припиняти виконання процесів в ній. Все відбувається в режимі реального часу і користувач цього не помічає.

Треба розуміти, що віртуальна машина емулює більшу частину складових реальної обчислювальної машини. Це зручно, коли необхідно поділити обчислювальні потужності між групою користувачів, тим самим у кожного з користувачів буде створюватися відчуття, що він працює на реальній фізичній машині, яка розташовується віддалено. Наша мета не орієнтована на користувачів. Ми спрямовані на аналіз додатків на кластері.

З цієї точки зору використання віртуальних машин є зайвою тратою ресурсів, які ми хочемо мінімізувати.

1.2 Контейнери.

Контейнери - система віртуалізації. Використовує рівень операційної системи, дозволяє запускати кілька операційних систем.

У своїй основі використовує системи взаємодії з ядром системи.

Даний метод схожий на попередній, з різницею в тому, що в основі знаходяться не віртуальні машини, а контейнери додатків.

Використовуючи дану технологію взаємодію з віртуальної одиницею відбувається через операційну систему, при цьому гіпервізор відсутній. Існує кілька контейнерів, основоположним є LXC (LinuX Container), а також LXD, CGManager, LXCFS. Для більш простої і зручної роботи використовують програмні продукти, які в свою чергу базуються на контейнерах і представляють зручний інтерфейс взаємодії з ними.

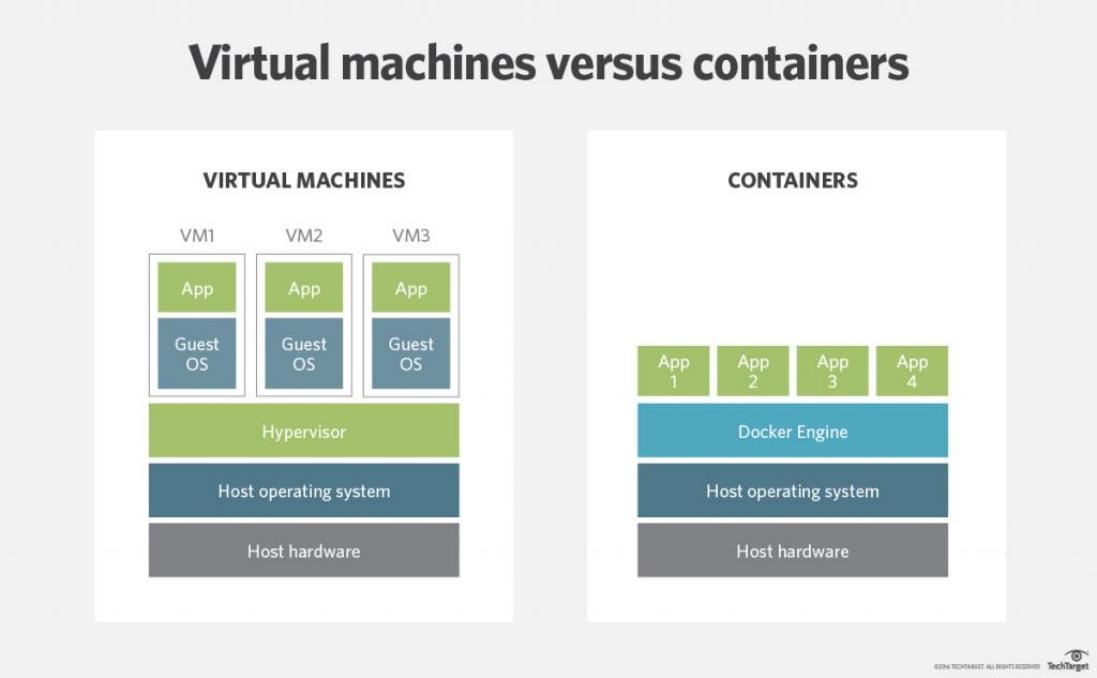
Як приклад можна привести одну з популярних платформ для роботи з контейнерами Docker. В цілому використання даного підходу дозволяє створювати віртуальні одиниці більш ефективно і менш ресурсовитратно в порівнянні зі створенням віртуальної машини. При цьому функціональні можливості, які надають контейнери додатків схожі з попереднім підходом.

На жаль, є ряд недоліків, головним з них є те, що механізми для роботи з контейнерами спираються на можливості ядра операційної системи. При роботі з новими версіями операційних систем Linux можливі ситуації, коли виникають проблеми сумісності і частина функціоналу не функціонує належним чином. Говорячи ж про інші операційні системи, наприклад Windows, виникають більш критичні проблеми. Через те, що дані операційні системи мають різні ядра, з першого погляду здається, що немає можливості використовувати функціональність, що надається контейнерів в ОС Windows. Однак існують рішення, коли на базі ОС Windows створюється мінімалістичний віртуальне середовище, яка емулює ядро Linux і на його базі є можливість працювати з контейнерами, саме такий підхід використовується в рішенні. Ситуація емуляції Windows на базі контейнерів Linux проблематична з огляду на те, що Windows закрита операційна система. Однак при використанні програми для емуляції Windows API - Wine є можливість емулювати роботу Windows. У будь-якому випадку в рамках даної роботи в використанні даної операційної системи немає необхідності.

1.3 Різниця між віртуалізацією та контейнеризацією

Існують два основні варіанти віртуалізації, а точніше два підходи до створення незалежних ізольованих обчислювальних просторів на одному фізичному сервері: віртуальні машини, яким потрібен гіпервізор, і віртуальні контейнери. У першому випадку для кожної віртуальної машини використовується власна гостьова ОС, а в другому для всіх контейнерів застосовується ядро однієї хостової ОС. Відповідно, перша дозволяє створювати неоднорідні обчислювальні середовища на одному комп'ютері, друга - тільки однорідні.

Однак, оскільки віртуальні машини включають операційну систему, їх розмір може складати кілька гігабайт. Також недоліком віртуальних машин можна назвати те, що для завантаження ОС і ініціалізації програми, яке в них розміщено, потрібно відносно більше часу. Контейнери легші і, в основному, їх розмір вимірюється в мегабайтах. Порівнюючи їх продуктивність з віртуальними машинами, контейнери можуть запускатися майже миттєво. При виборі між контейнерами та віртуальними машинами слід враховувати цілі, які потрібно досягти.



Багато проблем виявляються при зміні середовища додатків. Це може статися, наприклад, коли розробник запускає код в тестовому середовищі після розробки. Зміни

можуть стосуватися не тільки обчислювальних ресурсів, але також і мереж. Топологія мережі, політики безпеки і багато іншого може бути різним.

Переваги контейнерів

Гнучке середовище. Найбільша перевага у використанні технологій контейнерів полягає в тому, що їх можна створювати набагато швидше, ніж екземпляри віртуальних машин. Їх легка вага забезпечує менші накладні витрати з точки зору продуктивності і розміру.

Підвищена продуктивність. Контейнери підвищують продуктивність розробників за рахунок усунення міжмережевих залежностей і конфліктів. Кожен контейнер може розглядатися як окремий мікросервіс і, отже, може бути незалежно оновлений без будь-яких проблем з синхронізацією.

Управління версіями дозволяє відстежувати версії контейнера, стежити за відмінностями між ними і т.д.

Переносимість середовища обчислень. Контейнери інкапсулюють всі відповідні деталі, такі як залежності додатків і операційні системи, необхідні для запуску програми. Це полегшує переносимість способу контейнера з одного середовища в іншу. Наприклад, один і той же образ можна використовувати для роботи в середовищі Windows / Linux або dev / test / stage.

Стандартизація. Більшість контейнерів засновані на відкритих стандартах і можуть працювати в усіх основних дистрибутивах Linux, Microsoft і т. д.

Безпека. Контейнери ізолюють процеси одного контейнера від одного і від базової інфраструктури. Таким чином, будь-яке оновлення або зміна в одному контейнері не впливає на інший контейнер.

Недоліки контейнерів

Підвищена складність: при n числі контейнерів, які працюють з додатком, також збільшується коефіцієнт складності. Управління безліччю контейнерів може бути

складним завданням у виробничому середовищі. Такі інструменти, як Kubernetes і Mesos, можуть полегшити управління великою кількістю контейнерів.

Також складність полягає в тому, що зазвичай в контейнер «пакується» набагато більше ресурсів, ніж потрібно, а це призводить до розростання образу і великому розміру контейнера.

Класифікація контейнерів

Різні програми можуть бути встановлені, налаштовані і можуть виконуватися так само, як якщо ми запускаємо додаток на ОС хоста. Ресурси, призначені контейнеру, видно тільки йому. Для створення контейнерів ОС ми можемо використовувати такі контейнерні технології, як LXC, OpenVZ, Linux VServer, BSD Jails і Solaris.

Параметр	Віртуальні машини	Контейнери
Гостьова ОС	Кожна віртуальна машина работает на віртуальному апарат-ном забезпеченні і ядро завантажується в свою власну область пам'яті.	Всі гості мають (розділяють) одну і ту ж ОС і ядро. Образ ядра завантажується в фізичну пам'ять.
Комунікація	Здійснюється через Ethernet-пристрої.	Стандартний механізм IPC, наприклад сигнали, pipes, сокети.
Безпека	Залежить від ініціалізації гіпервізору.	Конструктор Linux: простору імен, кон-контрольні групи, chroot.
Продуктивність	Потребує більше ресурсів для віртуальної машини.	Краща продуктивність з-за спільного використання ресурсів хоста.
Ізоляція	Не можна ділитися файлами та біб-лиотека між гостями хоста.	Піддиректорії можуть бути монтовані до контейнера і використовуватися.
Час запуску	Декілька хвилин	Декілька секунд

Сховище	Займають більше місця, так як всередині них повинні бути установлено ОС і програми.	Контейнери займають менше місця, так як розділяють одну ОС.

1.3. Аналіз і обґрунтування вибору засоби віртуалізації

На сьогоднішній день в середовищі традиційної віртуалізації є невеликий вибір платформ. В цілому, він обмежується двома основними варіантами - VMware Workstation і Oracle VirtualBox. Що стосується альтернативних рішень, то вони або істотно поступаються їм по функціональності, або їх випуск припинений.

VMware Workstation - платформа з закритим вихідним кодом, що розповсюджується на платній основі. Відкритий код присутній лише у її неповної версії - VMware Player. У той же час її аналог - VirtualBox - є откритим ПО. VirtualBox безкоштовний.

VirtualBox оперує віртуальними машинами, а значить містить в собі недоліки традиційної віртуалізації з використанням гіпервізора, головним чином розмір, так як кожна віртуальна машина включає в себе не тільки додаток і всі його бібліотеки, а й всю гостьову операційну систему, яка може мати розмір десятків гігабайт.

VirtualBox найчастіше використовується в зв'язці з Vagrant. Vagrant використовує набагато простішу архітектуру, ніж, наприклад, Docker. Кожне середовище має свою власну віртуальну машину і налаштовується за допомогою Vagrantfile. Vagrantfile говорить Vagrant, як налаштувати віртуальну машину і які сценарії повинні бути запущені в цілях надання навколишнього середовища. Vagrantfile схожий з Dockerfile в Docker.

В силу об’ємності ресурсів для запуску віртуальних машин та широкого поширення і легковажності контейнерів в даній роботі обрано контейнерна віртуалізація. Контейнери у багатьох відношеннях перевершують традиційну віртуалізацію, але особливо в продуктивності і масштабованості. Серед технологій контейнерної віртуалізації будуть розглянуті технології LXC, OpenVZ і Docker.

LXC контейнери часто розглядається як щось середнє між Chroot і повноцінною віртуальної машиною. LXC дозволяє створювати готові образи контейнерів на основі шаблону, який встановлює початкову файлову систему і конфігурацію контейнера. LXC поставляється в комплекті з шаблонами для кількох основних дистрибутивів Linux. Крім того, LXC може заморожувати і розморожувати контейнери, надаючи можливість призупиняти і згодом відновлювати їх виконання. Також можливо створити контрольну точку, отримати інформацію про стан контейнера. Контейнер може бути пізніше відновлений до свого попереднього стану, захопленого контрольною точкою. Також можливо клонування контейнерів.

Docker - відносно нова платформа, яка служить легкою альтернативою віртуальним машинам. Протягом останніх декількох років Docker активно розвивається і в останні місяці набирає популярність в галузі. Велика кількість нових проектів і платформ будується поверх Docker, що призводить до створення багатої і живої екосистеми.

Docker був започаткований як проект по створенню single-application контейнерів Linux (LXC) і представив численні удосконалення LXC, котрі зробили контейнери більш гнучкими і легшими для використання, ніж LXC, а також деякі інші більш старі контейнерні технології, такі як FreeBSD Jails і Solaris Zones.

Docker має ряд переваг перед LXC:

- Переносимість. Портативність дозволяє контейнеру працювати в різних дистрибутивах ОС і апаратних конфігураціях без будь-яких змін самого образу.

- Репозиторий з безліччю образів різних систем; управління версіями образів, використання архівів образів і спільне їх використання.

- Різні засоби автоматичного запуску контейнерів з налаштуваннями (Docker-compose, Docker-swarm);

- Зручне API для взаємодії з контейнерами;

- Створення образу контейнера на основі вже готових образів.

- Повторне використання компонентів

Ще одна ключова проблема в розробці програмного забезпечення – «пекло залежностей» яке також може бути вирішено за допомогою Docker, оскільки контейнери Docker мають всі залежності, необхідні всередині контейнеру. Docker також має інструменти проти "гнилі коду" за допомогою управління версіями образів [23].

OpenVZ є в деякому роді предком для проекту LXC, так як безліч кодових баз LXC або виробляються від OpenVZ, або надаються членами команди OpenVZ. Основна відмінність полягає в тому, що проект OpenVZ використовує своє власне ядро, спочатку отримане з версії ядра Linux 2.6. Використання технології OpenVZ вимагає установки модифікованого ядра операційної системи, яке може бути несумісним з використовуваним на суперкомп'ютері програмним і апаратним забезпеченням. У той же час LXC і Docker для поділу ресурсів і ізоляції процесів використовують штатну підсистему cgroups сучасних ядер Linux. OpenVZ дозволяють на одному фізичному сервері запускати безліч ізольованих копій операційної системи, званих Віртуальні Приватні Сервери (Virtual Private Servers, VPS) або Віртуальні Середовища (Virtual Environments, VE). OpenVZ часто використовують для створення таких серверів і називають технологією контейнерних VPS.

Таким чином, найбільш адекватними системами контейнеризації є LXC і Docker. LXC складно налаштовувати, Docker ж базується на LXC і максимально спрощує створення і налаштування контейнерів, тому він і був вибраний для розробки технології конфігурації контейнерів.

Docker задовольняє потреби розробників, яким необхідно надати своє програмне забезпечення іншим користувачам в середовищі, де вже присутні всі необхідні залежності і настройки, надаючи кожному користувачеві можливість запускати додаток в тому ж середовищі.

1.5. Екосистема Docker

1.5.1. Історія і плюси технології

Docker - це проект з відкритим вихідним кодом, який автоматизує розгортання додатків всередині програмних контейнерів, що є легким механізмом ізоляції для поділу одного фізичного хоста на декілька віртуальних середовищ [21].

Спочатку проект Docker був випущений компанією dotCloud (тепер Docker Inc.) під ліцензією Apache 2.0 як продукт з відкритим вихідним кодом в березні 2013 року. Через два місяці був запущений публічний реєстр Docker. У другій половині року Google, Yandex і Baidu (найбільш використовувані російські і китайські пошукові системи) інтегрували Docker в свої хмарні сервіси. Ідея Docker добре виражається з мети, поставленої командою його розробників: «Створити «кнопку», яка дозволить створювати і розгортати будь-який додаток на будь-якому сервері в будь-якому місці».

Docker почав з призначеної для користувача програми для запуску Lmux-контейнерів (LXC). З версії Docker 0.9 стали підтримуватися два драйвера: драйвер LXC, в якому використовувався раніше використовуваний liblxc і власний драйвер, що використовує бібліотеку libcontainer. Він обробляє управління контейнерами, використовуючи можливості ядра, простору імен і cgroups.

Але Docker - це більше, ніж просто бібліотека віртуалізації, вона абстрагує відмінності між збірками операційних систем і створює стандартизоване середовище для розробки додатків. Розробник програмного забезпечення може створити стандартний додаток, яке стає портативним і може працювати скрізь, де встановлений Docker Engine. Це значно економить час, оскільки більше не потрібно підтримувати безліч різних платформ і дистрибутивів операційної системи. Той факт, що кожен застосунок працює у власному контейнері, вирішує багато поширених проблеми, такі як повна деінсталяція або коли для двох додатків треба дві різні версії однієї і тієї ж залежності.

Частина програмного забезпечення упакована в файлову систему в контейнерах Docker, яка містить всі залежності, необхідні для запуску программного забезпечення: код, Виконавча, системні інструменти, сі¬стемние бібліотеки, які встановлені на сервері. Це гарантує, що незалежно від середовища програмне забезпечення завжди буде працювати одіна¬ково. Особливості контейнерів Docker наступні:

- легкість: контейнери Docker мають невелику вагу і спільно використовують один і той же ядро операційної системи на одній машині і ефективно використовують пам'ять (ОЗП), що дозволяє додатку запускатися практично миттєво. Завантаження образів набагато ефективніше через мінімальне використання диска, і вони структуровані з багаторівневих файлових систем.

- відкритість: Контейнери Docker засновані на відкритих стандартах і можуть працювати поверх будь-яких операційних систем, таких як Linux і Windows.

- безпека: доданий рівень захисту додатків надається в кожному контейнері для відділення від інших контейнерів та хмарної ін-фраструктури.

- виконання більшої кількості робочих навантажень на одному і тому ж обладнанні.

Docker забезпечує життєздатну, економічно ефективну альтернативу віртуальним машинам на основі гіпервізора. За останні роки Docker значно змінився. База користувачів продовжує зростати, і багато великих компаній публічно оголосили про те, що вони активно використовують Docker як частину своєї власної платформи.

1.5.2 Компоненти Docker

Docker Engine є основною екосистеми Докер, який має повністю відкритий вихідний код. Docker Engine необхідно встановити на кожну систему, яка взаємодіє з Docker.

Docker використовує клієнт-серверну архітектуру. Docker Engine має наступні основні компоненти: Docker Daemon, REST API, Docker Client.

Користувач використовує клієнт Docker для взаємодії з демоном. Вони можуть працювати в одній системі або ж клієнт Docker може підключатися до Docker демона віддаленого типу, оскільки клієнт і демон Docker взаємодіє за допомогою REST API, через UNIX сокети або за допомогою мережевого інтерфейсу. Компоненти Docker Engine будуть розглянуті нижче трохи докладніше:

- Docker Daemon.

Docker Daemon працює на хост-машині (як root) і відповідає за прослуховування і обробку запитів API від Docker Client. Він також керує об'єктами Docker, наприклад, образами, контейнерами, мережами і томами. Для створення образів він парсит так званий Dockerfile і виконує інструкції [4]. Daemon може бути запущений за допомогою команди docker при передачі -d прапора або команди systemctl start docker або service docker start.

- Docker Client – інтерфейс командного рядка (CLI), який дозволяє користувачеві спілкуватися і взаємодіяти з сервером через REST API. CLI пропонує команди для управління мережами, контейнерами і образами або томами для контейнерів.

Docker Client приймає команди та конфігураційні прапори від користувача і використовується для зв'язку з Docker Daemon. Клієнт Docker є основним способом взаємодії багатьох користувачів з Docker. CLI пропонує команди для управління мережами, контейнерами і образами або томами для контейнерів. При використанні таких команд, як docker run, клієнт відправляє ці команди в Docker Daemon, що спрямовує всі подальші дей¬ствія.

- REST API, що надається Docker Daemon і пропонує проміжний рівень для взаємодії з сервером Docker і управління всіма його функціями. API можна відкрити або через локальний сокет на тому ж хості, або через порт, відкритий через мережу.

Docker виконує свої функції завдяки поєднанню певних со¬ставляющіх:

1. Компонента збірки для створення контейнерів (докер-образи або read-only шаблон).

2. Компонента поширення і зберігання образів (докер-реєстр).

3. Компонента роботи для функціонування створених додатків(докер-контейнери).

Образи Docker є основою для всіх контейнерів та надають файлову систему тільки для читання, що містить базову операційну сі¬стему. Наприклад, образ може містити в собі ОС Ubuntu з web-сервером Apache і встановленим всередину web-додатком. Кожен образ складається з по¬следовательності шарів, як показано на рис. 1.2.

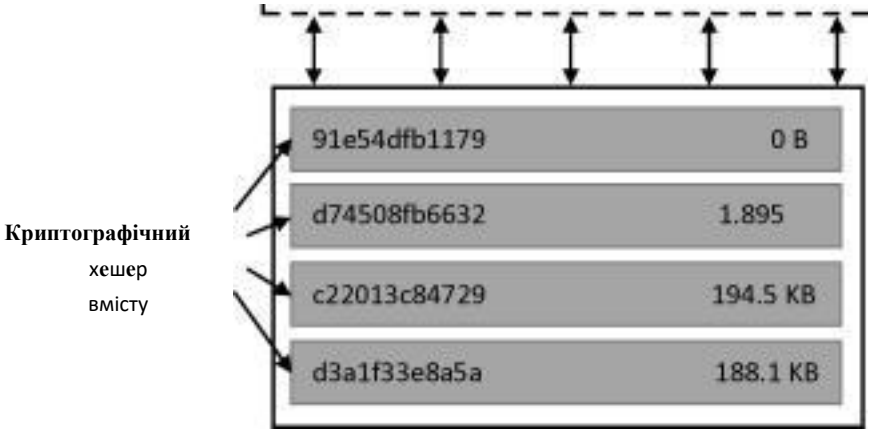


Рис.1.2. Система шарів Docker

Як показано на малюнку вище, кожен з шарів має криптографічний хеш, який його ідентифікує. Шари доступні тільки для читання, тобто всі зміни, виконані в занедбаному контейнері, будуть записуватися в новий шар контейнера поверх інших, з випадковим UUID (універсальним унікальним ідентифікатором). Коли контейнер буде видалений, цей шар також буде видалений, але шари під ним будуть існувати.

Цей спосіб створення образів дозволяє зручно обмінюватися одними і тими ж шарами між різними запущеними контейнерами, зберігаючи при цьому свій власний стан даних (контейнерний шар). Наприклад, якщо десять різних контейнерів використовують один і той же образ Ubuntu, і кожен з них має свою мету (один – це веб-сервер, інший – база даних і т. Д), то всі вони будуть використовувати одні і ті ж

базові шари з базового образу, що не дублює їх, маючи тонкий шар поверх них з даними, що використовуються для призначення користувачів.

Образ, який не має батьківського образу, називається базовим або кореневим. Базові образи являють собою повноцінну ОС і використовується в якості відправних точок, з яких можуть успадковуватися інші образи. Стандартними прикладами кореневих образів є образи дістрібутивів Linux.

Кожен образ успадковується від послідовності інших образів, аж до базового, і являє собою набір або послідовність змін в файлової системі.

Наприклад, якщо є два образи на основі одного і того ж дистрибутива Linux, але дві різні системи хостингу, наприклад nginx і apache. Ці два образи вказують на один і той же базовий рівень (який існує тільки один раз), а потім додають на нього ще один шар. Коли додаток запускається в якості контейнера, поверх нього накладається додатковий шар. Коли необхідно змінити файл з рівня, доступного тільки для читання, цей файл копіюється на записуваний шар, і зміна робиться там.

Часто образ заснований на іншому образі з деякою додаткової налаштуванням. Образи Docker можуть бути створені, оновлені або видалені. Можна створювати свої власні образи з нуля або завантажити і використовувати тільки ті, які були створені іншими і опубліковані в реєстрі. Щоб ствоорити власний образ, потрібен Dockerfile з кроками, необхідними для створення образу і його запуску. Кожна команда в Dockerfile створює шар в образі, цей процес представлений на рис.1.3. При зміні Dockerfile і відновленні образу, відновлюються тільки ті верстви, які були змінені. Це і робить образи такими легкими, компактними і швидкими, в порівнянні з іншими технологіями віртуалізації.

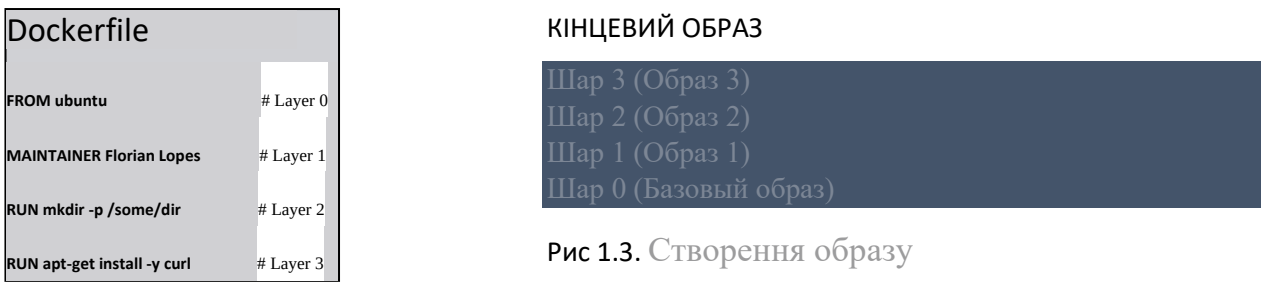


Рис 1.3. Створення образу

Іншими словами, образи Docker – це просто шаблони, які містять все, що потрібно з додатком, і з них запускаються контейнери. Процес створення способу додатків для Docker часто називають докерізацією.

Docker використовує архітектуру шарів, щоб спростити процес перемещення контейнерів в різних хостингових середовищах. AUFS (розширена багаторівнева файлова система уніфікації) є основним засобом, завдяки якому Docker забезпечує економію пам'яті і більш швидке розгортання контейнерів [24]. Завдяки AUFS контейнери можуть спільно використовувати однакові частини файлової системи, доступні тільки на читання. При запуску контейнера Docker використовує механізм під назвою union mount – файлові системи не монтуються в різних місцях, а поверх один одного, тому зміст каталогу може складатися з файлів каталогів з різних файлових систем.

Такий розподіл файлових систем і образів забезпечує дві основних переваги. По-перше, це забезпечує високу ступінь економії памяті. Якщо в двох контейнерах працює одна і та ж ОС і обмінюються деякими бібліотеками та залежностями, більшість їх файлових систем будут представлені тільки один раз на диску і не дублюються. По-друге, при завантаженні і розгортанні контейнера, якщо хост вже має попередні рівні файлової системи, від яких залежить конкретний контейнер, йому потрібно завантажити тільки інкрементні зміни.

Контейнери представляють собою виконувані екземпляри образів. Щоб запустити контейнер, потрібно вказати спосіб і команду для запуску. Усі зміни в файловій системі переходять на самий верхній контейнерний рівень файлової системи, зберігаючи всі верстви образів нижче.

Контейнери повністю ізолюють наступні елементи:

- файлову системи;

- IP адреса; мережеві інтерфейси;

- внутрішні процеси;

- простору імен;

- бібліотеки операційної системи;

- бінарні файли програми;

- файли конфігурації програми.

Можна запустити безліч контейнерів, заснованих на одному і тому ж образі. Кожен контейнер має унікальний ідентифікатор, наданий демоном Docker, ідентифікатор також може бути визначений користувачем.

Контейнер має доступ тільки до свого екземпляру багаторівневої файлової системи. Контейнер отримує рівень читання-запису при його створенні. Якщо контейнеру необхідно відредагувати файл, який є частиною одного з нижніх рівнів тільки для читання, цей файл буде скопійований на рівень читання-запису верхнього рівня, що демонструє рис.1.4. Контейнер може редагувати файл, проте файл не зберігається в базовому образі. Це забезпечує ізоляцію для кожного контейнера. Варто відзначити, що контейнери поділяють шари тільки в тому сенсі, що їх можна модифікувати одним артефактом. Цей артефакт – це образ, створений при створенні образу з Dockerfile.

Контейнери Docker є основним компонентом в Docker, і вони в основному більш-менш схожі на каталог, що містить все, що потрібно для запуску певної програми. Контейнери можна створювати, запускати, зупиняти, переміщати або видаляти в будь-який час. CLI Docker пропонує команди для управління контейнерами. Крім того, контейнери мають механізм обміну змінними середовища між двома або більше контейнерами.

Більш цікаві можливості - це вивчення логів (docker logs), перегляд процесів всередині контейнерів (docker top) або підключення до запущеного контейнеру (docker attach). Однією з важливих опцій є можливість запуску контейнера в фоновому режимі з використанням режиму демона з ключом -d.

При запуску контейнера можна ввести обмеження на кількість ресурсів, які він може використовувати. Щоб примусово використовувати максимальну межу пам'яті для даного контейнера, використовується параметр -m, а параметр -з - рівень пріоритету CPU. Ці параметри фактично передаються модулю cgroups. Маппінг портів контейнера з портами хоста дозволяє використовувати кілька контейнерів, в яких процеси запускаються на одному і тому ж порту.

Движок Docker розрахований на максимально можливу кількість ресурсів. Наприклад, при використанні традиційної віртуальної машини кожен додаток, кожна копія програми та кожна невелика модифікація застосунку вимагають створення повністю нової віртуальної машини, тоді як всі контейнери використовують одну і ту ж операційну систему, виконавчі файли і бібліотеки і не потребують створення нової віртуальної машини.

Тома даних дають можливість обмінюватися даними між контейнерами, вносячи безпосередні зміни в файлову систему, і ці зміни не включаються при Ком оновлень в використовуваний образ. Можна створити порожній том даних або змонтувати файл або ж каталог хоста. Том даних може бути або доданий в конкретний контейнер, або може бути створений контейнер виділеного обсягу даних, який може не тільки ділитися між контейнерами, а й також надавати такі функції, як міграція, резервне копіювання і відновлення [21].

Контейнери використовують конфігурацію хост-системи для визначення її основного DNS-сервера. Кожен контейнер може перевизначити цей параметр, якщо переданий конкретний параметр запуску.

Під час створення контейнера необхідно вказати кожен порт, який повинен мати доступ за межами контейнера. Це робиться зіставленням портів між вузлами. На малюнку 5, кожен контейнер вказує на один і той же порт 80, але кожен зіставляється зі списком оголошених портів хостом по-різному. Це ключовий аспект для запуску декількох копій одного і того ж образу.

Докери-реєстри - це сховища, в яких лежать образи контейнерів. Реєстр образів є центральним репозиторієм для образів контейнерів [8]. Docker Registry - це реєстр образів, підтримуваний Docker Inc.

Усередині реєстру різні образи додатки знаходяться в репозиторії. Робота з репозиторієм аналогічна використанню системи управління версіями git. Зазвичай образи створюються локально або спеціальним сервісом збірки, а потім відправляються в реєстр командою push (завантаження нової версії). Звідси образи можна завантажити командою pull (завантаження останньої версії) і запустити (команда run) в контейнерному кластері, такому як Docker Swarm. Також можна використовувати команду пошуку Docker для відображення доступних образів. Команда пошуку фактично повертає репозиторії, які можуть містити кілька версій певного образу, кожен зі своїм тегом версії, але одна з них відзначена тегом «latest» (остання версія). Образ докера також ідентифікується за допомогою id.

DockerHub і Quay.io є двома прикладами публічних реєстрів, розміщених в хмарі. Якщо жоден інший реєстр не вказано в Docker Engine, Docker Hub використовується за умовчанням [1].

DockerHub - це сервіс, який надає інструменти для управління і адміністрування образів Docker. Репозиторій Docker-Hub містить велику колекцію додатків і бібліотек, готових до розгортання в будь-якому середовищі Docker.

DockerHub обробляє аутентифікацію і авторизацію користувачів і містить три типи репозиторіїв:

- офіційні репозиторії, які містять образи від постачальників і вкладників (vendors and contributors) Docker;

- приватні репозиторії, де можуть зберігатися непублічні образи;

Приватний репозиторій виключається з результатів пошуку, і лише деякі люди можуть завантажувати образи і використовувати їх для запуску контейнерів.

- публічні репозиторії для обміну публічними образами.

Публічний репозиторій доступний для пошуку і образ з нього може бути завантажений ким завгодно.

Цей величезний реєстр включає в себе тисячі зображень, створених спільнотою з безліччю різних форм і властивостей, які можуть бути дійсно корисні для інших розробок. Станом на квітень 2015 року в державному реєстрі Docker Hub було представлено більше 45 000 образів, а це означає, що в основному всі основні Linux-програми вже докерезіровані.

Основний ізоляцією контейнерних технологій є три основні функції ядра: простору імен (namespaces), контрольні групи (cgroups) і SELinux. Простору імен та cgroups були основою з самого початку, в той час як SELinux був доданий пізніше. SELinux важливий при запуску контейнерів, де він використовується для захисту хоста і контейнерів один від одного.

Простір імен - це механізм, за допомогою якого кожен контейнер ізольований від хоста і інших контейнерів. Вони ізолюють кілька процесів, логічно ділячи їх простір ядра на кілька середовищ (наприклад, мережа, процеси або файлова система). Це призводить до того, що процеси не можуть бачити або керувати іншими процесами, запущеними на одному і тому ж хості. Зміни в глобальному ресурсі видимі для інших процесів, які є членами простору імен, але невидимі для інших процесів. Простір імен також надають власні мережеві пристрої для контейнерів, тому кожен контейнер матиме унікальне ім'я хоста і IP- адресу.

Використовуючи простору імен, контейнери можуть монтувати частину файлової системи, доступної всередині контейнера.

В даний час існує шість різних просторів імен. П'ять з них використовуються для створення контейнера Docker:

- Простір імен pid: Ізоляція процесу (PID: ID процесу).

- Простір імен net: мережева ізоляція (NET: Мережа).

- Простір імен ipc: управління взаємодією між процесами (IPC: взаємодії між процесами).

- Простір імен mnt: Управління точками монтування файлової сі-стеми (MNT: Монтування).

- Простір імен uts: Ізоляція ідентифікаторів ядра і версій. (UTS: Unix система поділу часу).

Контрольні групи (Control Groups) - ще одна важлива функція в ядрі Linux, спочатку розпочата в 2006 році інженерами Google. Cgroups використовується для обмеження, обліку та ізоляції використання апаратних ресурсів, таких як використання CPU, обсяг використовуваної пам'яті, дискового введення-виведення і мережевого зв'язку. Ці обмеження можуть застосовуватися до одного процесу або до набору процесів. Cgroups також забезпечують ізоляцію простору імен, щоб повністю ізолювати подання додатка від операційного середовища, включаючи дерева процесів, мережа, ідентифікатори користувачів і змонтовані файлові системи.

Контрольні групи гарантують, що кожен контейнер має достатні ресурси для належного функціонування, але, що ще важливіше, вони запобігають вичерпання цих ресурсів, обмежуючи використання ресурсів контейнерів, щоб один контейнер не міг вичерпати систему, використовуючи всі її ресурси. Наприклад, шляхом обмеження використання ресурсів для сервера apache, apache і всі його дочірні процеси не будуть перевищувати це обмеження ресурсів.

Під час розвитку Linux виникла ідея ізолювати процес від файлової системи хоста, і для цієї мети була створена команда chroot. Більшість застосунків для контейнерів сьогодні побудовані на основі chroot. Chroot, скорочений від change root - стара функція в Linux, яка змінює видимий кореневий каталог з поточною діяльністю. Chroot використовується як для виклику утиліти, так і для системного виклику і дозволяє вказати нову кореневу директорію, що відрізняється від «/». Процес і його нащадки потім не можуть отримати доступ до файлів вище нового корневого каталогу, в той час як програми з іншого місця все ще видно всередині нового кореня.

Linux є основною платформою Docker, так як останній заснований на особливостях, що надаються ядром операційної системи. Незважаючи на це, його можна запустити на macOS або Windows. Різниця полягає лише в тому, що Docker на них інкапсульований в маленьку віртуальну машину. Нині Docker для цих ОС досяг значного рівня зручності використання і дуже схожий на оригінальний додаток.

1.5.3. Формати файлів

Dockerfile автоматизує процес створення образів. Dockerfile описує середовище і виконується через інтерфейс Docker. Dockerfile є текстовим файлом, який містить набір інструкцій, які використовуються Docker для створення образу.

Щоб зберегти аналогію з мовами програмування, Dockerfile є кодом і компілюється в запущений образ додатком Docker. Потім скомпільований образ може бути додатково налаштований або розгорнуто через інтерфейс Docker, який використовує базовий Docker-daemon для взаємодії з операційною системою.

Нижче перераховані деякі ключові функції, дозволені Dockerfile:

- додавання програмного забезпечення;

- створення змінної оточення;

- визначення TCP-портів, які будуть відображатися поза контейнера;

- запуск команди всередині контейнера під час збирання (виконується в обо-оболонка bash);

- опціональна установка точки входу для контейнера. Точкою входу є-ється певний процес запуску для контейнера;

- додавання файлу або директорії.

Кожна команда (за деякими винятками) в Dockerfile створить новий шар в багаторівневій файловій системі образу. Docker Engine відповідає за розбір і інтерпретацію цих інструкцій, а потім за побудову остаточного Docker образу на їх основі. Процес проілюстрований на рис.1.7.

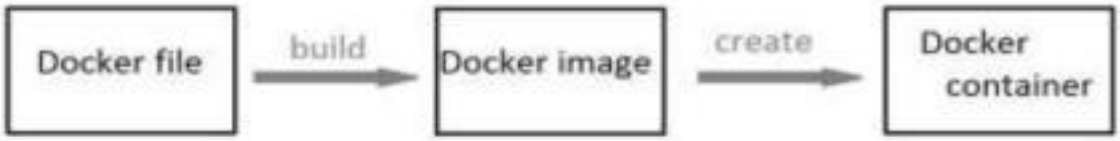


Рис.1.7. Процес побудови контейнера

Docker Compose - це додаток CLI, який аналізує файл docker-compose і створює з нього багатоконтейнерне середовище. Сам файл docker-compose визначає додаток, що складається з одного або декількох сервісів. Docker Compose ділить заданий файл docker-compose, переводить визначення сервісів, мереж і томів в команди docker run, docker volume create і docker network create і виконує їх. У цих файлах описуються основні параметри запуску контейнерів, що значно спрощує запуск складових додатків, які вимагають кілька контейнерів, прописуються під'єднуючі розділи і порти які потрібно вивести назовні, порти можна перенаправляти. Завдяки даному інструменту можна задавати структуру застосунку, його обчислювальні ресурси і залежності.

Конфігурація також може розширювати існуючу, так що, наприклад, додаток може мати просту загальну конфігурацію плюс дві спеціалізі¬рованние, один для розробки, а інший для продакшена.

Основна ідея полягає в можливості з'єднання декількох контейнерів разом, що дозволяє складати складний додаток з декількох образів. Часто буває так, що програмне забезпечення складається з декількох компонентів, наприклад, веб-сервера і бази даних. Установка кожної окремого компонента вручну повторюється і, можливо, призводить до помилок завданням, тому був створений проект під назвою Fig. У лютому 2015 року Fig вже вважався застарілим на користь Docker Compose, який заснований на кодової базі Fig.

1.6. Огляд аналогів і виявлення їх недоліків

Спочатку буде описаний процес розгортання сайтів на локальному сервері вручну. Багато розробників при розробці веб-додатків використовують локальне

середовище своєї операційної системи. В процесі розгортання на локальному сервері веб-програми, написаної на php, дії зводяться до такої послідовності завдань:

1. Установка операційної системи.

2. Встановлення та налаштування apache.

3. Встановлення та налаштування сервера MySQL.

4. Установка php і модулів PHP (підтримка Apache, MySQL та GD).

5. Встановлення інших необхідних розширень.

6. Конфігурація virtmlhost.

7. Встановлення прав доступу до файлів.

8. Перенесення файлів програми, конфігурація.

9. Перезапуск Apache і MySQL для застосування змін.

Цей процес забирає досить багато часу, особливо якщо потрібна установка специфічних розширень а також при розгортанні існуючого проекту без документації.

Для розгортання локальної версії готового сайту створюються папки з файлами в домашній директорії, але в цьому випадку для кожного нового хоста необхідно виконати наступні дії:

1. У домашній директорії /home/usename потрібно створити папку www, а в ній підкаталог з ім'ям майбутнього сайту, наприклад, mysite. Саме в цей підкаталог згодом і потрібно буде завантажувати файли сайту.

2. Завантажити в папку файли з сервера або git-репозиторію.

3. Відкрити термінал і перейти в директорію /etc/apache2/sites-available.

Такі операції, як установка програмного забезпечення або зміна конфігураційних файлів вимагають повноважень root. Щоб їх тимчасово отримати, потрібно використовувати команду sudo.

Тож для того, щоб перейти в зазначену вище категорію, потрібно використовувати команду sudo nano /etc/apache2/sites-available.

nano - це консольний текстовий редактор для Unix і Unix-подібних операційних систем, їй передається один параметр - назва файлу, який потрібно відкрити.

У цій папці містяться конфігураційні файли віртуальних хостів.

4. Далі можна створити новий файл, наприклад mysite.conf

Прописуємо в ньому такі рядки:

<VirtualHost antelas.dev:80>

ServerName antelas.dev ServerAdmin webmaster @ localhost DocumentRoot / var / www / antelas / public_html <Directory "/ var / www / antelas / public_html /">

AllowOverride All Allow from All </ Directory>

ErrorLog $ {APACHE_LOG_DIR} /error.log CustomLog $ {APACHE_LOG_DIR} /access.log combined </ VirtualHost>

У рядку DocumentRoot потрібно задати шлях до створеної в першому пункті папці.

5. Внести сайт в список хостів. Відкрити на редагування файл: sudo nano / etc / hosts

Додати сайт, повинно вийти наступне:

127.0.0.1 localhost mysite.dev

6. Активувати новий хост за допомогою утиліти a2ensite, яка включає сайт в конфігурації apache і робить симлінк сайту в папку /etc/apache2/sites-enabled [3]: sudo a2ensite mysite

6. Далі залишається перезапустити сервер apache: sudo systemctl reload apache2

7. Сайт буде доступний за адресою http://mysite.dev

Після налаштування віртуального хоста необхідно також розгорнути базу даних і відредагувати конфігураційний файл з урахуванням локальних налаштувань підключення до бази даних.

5. Додати сайт в git-репозиторій.

Був також проведений пошук аналогів, представлені нижче сервіси мають свої недоліки, які не дозволять впровадити їх в процес роботи компанії:

- http: //www.startwithdocker.com/

Є інструментом для командного рядка, який формує три файли: Dockerfile, docker-compose.yml і service.yml. Однак, даний сервіс не збирає параметри оточення, не передбачає автоматичного запуску контейнерів і запуск з командного рядка може бути неможливий за умови, якщо на сервер можна зайти по ssh.

- https://phpdocker.io/generator

Даний сервіс має інтерфейс, який відображений на рис.1.8 де можна задати параметри конфігурації, вибрати версію PHP і розширення, тип і версію бази даних. В результаті завантажується папка з docker-compose і Dockerfile файлами для серверів ngnix і apache. Мінусом даного інструменту є те, що він також не збирає параметри оточення і формує файли не для конкретного додатка, а за запитом.

Рис.1.8. Інтерфейс сервісу phpdocker.io

В таблиці 1.2 описане порівняння даних сервісів

Таблиця 1.2 Порівняння аналогів

	Формування файлів	Збір даних про середовище	Можливість не користуватись cmd
startwithdocker	+	-	-
phpdocker.io	+	-	+
generator-	+/-	-	-
docker-compose	+		+

Виходячи з результатів можна зробити висновок, що не було знайдено аналога, який би повністю задовольняв потреби. Жоден сервіс не розрахований на роботу з конкретним додатком, тобто не збирає інформацію про оточення сайту для забезпечення ідентичності середовища в подальшому, що є основним аспектом.

Таким чином, жоден із способів не підходить для автоматичної конфігурації файлів конкретно обраного сайту, тому було прийнято рішення розробити власну технологію і її реалізацію.

2.РОЗРОБКА МЕТОДИКИ АВТОМАТИЧНОЇ КОНФІГУРАЦІЇ КОНТЕЙНЕРІВ

2.1 Аналіз вимог

Стандартний спосіб розгортання нового проекту наступний:

- 1. Підготовка оточення відповідно до вимог (OS і налаштування);
- 2. Клонування проекту;
- 3. Установка пакетів, необхідних для розробки;
- 4. Підготовка проекту до запуску (копіювання директорій, симлінк, налаштування конфігурацій, хостів і т.д.).

Вищеописаний спосіб було розглянуто більш докладно в попередньому розділі і він є досить поширеним способом розгортання сайту на локальній машині, проте, має низку істотних недоліків. В ідеалі, конфігурація системи розробника повинна збігатися з конфігурацією майбутньої системи, як мінімум в частині версій ПЗ і особливостей його збірки. При використанні ж даного способу часто виникають складнощі з доступністю необхідних версій програмних засобів, а особливо з налаштуванням і оновленням потрібних компонентів. Так як оточення на локальному комп'ютері не збігається з оточенням сервера, на якому знаходиться сайт, то можуть відрізнятися версії, наприклад, php і різних бібліотек, що призводить до виникнення помилок. Часто доводиться підключати додаткові модулі, вносити зміни в конфігураційний файл php. Все це займає тривалий час, яке розробник міг би витратити на вирішення завдань. Велика кількість проектів на одному комп'ютері також створює певні проблеми, наприклад, необхідність зміни версій php для різних сайтів і вирішення виникаючих конфліктів.

- Проблема ідентичності оточення може бути вирішена за допомогою внесення додатка з усіма його залежностями в одне середовище, яка за своїми параметрами буде копіювати середовище бойового сервера. Це можна досягти при іс¬пользованіі віртуальних машин або контейнерів. У першому розділі були рас¬смотрени їх

переваги та недоліки і зроблено висновок про те, що контейнерна технологія підходить більшою мірою через свою легкість та гнучкість у перенесенні.

- Проблему тимчасових витрат також можна вирішити за допомогою контейнерів. Як технологією для реалізації продукту був обраний Docker. Пошук аналогів з використанням даної технології не привів до результату, який повністю задовольнив би потреби, що стоять перед компанією.

- Для скорочення часу на процес локального розгортання сайтів бу¬дет розроблена методика для автоматичної конфігурації контейнерів і запуску додатків.

- Перед тим, як приступити до розробки даної технології, необхідно визначити, яким вимогам повинен відповідати додаток, створене в даль¬нейшем на її основі.

- Додаток буде автоматичним генератором конфігураціон¬них файлів Docker. Можна виділити наступні вимоги:

1. Скорочення часу, що витрачається на локальне розгортання суще-ного сайту.

2. Оптимізація роботи розробника шляхом автоматизації рутинних опера-цій.

3. Рішення проблеми ідентичності оточення.

4. легковаге і швидкість.

5. Володіння можливістю масштабованості.

2.2. Вибір інструментарію

У компанії, для якої буде розроблено додаток на основі нової технології, програмісти для створення сайтів або тестування змін на вже існуючих використовують ОС Ubuntu і стек LAMP, що є досить широко використовуваним варіантом настройки серверів з цією операційною системою і одним з найпоширеніших при розробці сайтів. Тобто, даний стек буде використаний в додатку, заснованої на розроблюваній технології, однак, згодом методика може бути модифікована шляхом зміни стека інструментів.

- LAMP - це абревіатура, перші літери слів Linux + Apache + MySQL +

- PHP:

- Linux - операційна система;

- Apache - веб-сервер, який є кросплатформним і підтримується майже всіма сучасними операційними системами. Основними перевагами Apache вважаються надійність і гнучкість конфігурації;

- MySQL - система управління базами даних (СКБД);

- PHP - скриптова мова програмування, що використовується для створення серверної частини сайтів. В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів;

- Додатково встановлюється phpMyAdmin. phpMyAdmin - веб-додаток з відкритим кодом, написаний на мові PHP і представляє собою веб-інтерфейс для адміністрування MySQL. PHPMyAdmin дозволяє через браузер здійснювати адміністрування сервера MySQL, запускати команди SQL і переглядати вміст таблиць і баз даних;

- WordPress - система керування вмістом сайту. Дозволяє швидко створити і розгорнути практично будь-який сайт або додаток;

- Також з великою ймовірністю можуть знадобиться деякі бібліотеки PHP, наприклад, бібліотека для роботи із зображеннями GD і бібліотека для роботи з URL - CURL;

- Встановлений на локальний комп'ютер Docker;

- Для використання Docker в Linux-середовищі необхідно мати ядро версії не нижче 3.10.

2.3. Створення технології

Першим етапом розробки методики є визначення движка сайту, який необхідно розгорнути локально. В даний момент буде використана CMS Wordpress, так як на

сьогоднішній день вона є однією з найпопулярніших платформ і використовується у великій кількості проектів (в компанії, для якої розробляється цей продукт, частка сайтів на Wordpress становить близько 50%). Надалі ж методика може легко масштабуватися для сайтів на інших CMS або PHP-фреймворк. Так як різні CMS мають різний вміст і назви конфігураційних файлів, цей етап необхідний для визначення їх шаблону, на основі якого буде формуватися цей файл з урахуванням даних, отриманих скриптом.

Далі потрібно продумати структуру файлів, необхідних для реалізації програми по конфігурації контейнерів. У цій роботі буде представлена наступна структура:

- Основний файл index.php, який буде запускати роботу додатка. До нього слід підключити наступні файли:

- env.php, що відповідає за збір оточення.

- generator.php, який формує файли конфігурації з шаблонів з використанням даних, отриманих env.php.

- dumper.php, що експортує базу даних сайту.

- Папка з необхідними шаблонами для формування конфігураційних файлів.

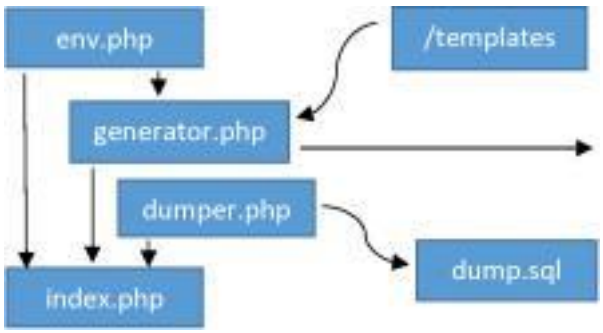
Наступним етапом потрібно за допомогою скрипта якомога точніше і детально визначити оточення існуючого сайту, який розміщений на сервері. На даному етапі збираються версії мови, бази даних, що використовуються бібліотеки і розширення, налаштування сервера, які потім використовуються в конфігураційних файлах. Цей етап відповідає за ідентичність яку переносять середовища і дуже важливий, тому що від нього залежить коректність побудови скриптом цих файлів на наступному етапі і успішний запуск сайту в контейнері.

Далі сформувати правильні шаблони для наступних файлів: файли Docker - Dockerfile і docker-compose.yml, файл конфігурації CMS і налаштування PHP. Шаблони можна видозмінювати і доповнювати під свої потреби. Тут також важлива

правильність вказівки директорій, які будуть монтуватися з локального комп'ютера в контейнер.

Написати скрипт, який буде експортувати базу даних потрібного сайту. Створити основний скрипт генерації конфігураційних файлів з шаблонів з використанням даних про оточення і формування архіву з ними на скачування.

Схема попередніх етапів зображена на рисунку 2.



Dockerfile

docker-compose.yml

wp-config.php

uploads.php

Рис.2. Структура додатку

Далі потрібно помістити завантажені файли в папку з сайтом, який необхідно розгорнути на локальному комп'ютері.

Запустити Docker-контейнер на локальному комп'ютері за допомогою однієї команди. Для цього Docker буде використовувати створені файли конфігурації.

Висновки

У цьому розділі були розглянуті інструменти і технічні засоби, обрані для розробки програми, що буде засновано на створену технологію, а саме стек LAMP, CMS Wordpress і засіб контейнерної вір¬туалізації Docker.

Була також розроблена власна методика автоматичної конфігурації Docker-контейнерів, яку коротко можна описати таким чином:

1. Визначення рушія сайту, який потрібно розгорнути.

2. Розробка структури файлів для майбутнього програми.

3. Продумування і створення шаблонів конфігураційних файлів.

4. Написання скрипта, що збирає оточення сайту на сервері.

5. Створення скрипта, експортує базу даних.

6. Розробка скрипта генератора, який формує необхідні файли з шаблонів з урахуванням даних про оточення і поміщає їх в архів для скачування.

7. Завантаження файлів з архіву в папку сайту на локальному сервері.

8. Запуск сайту в контейнері однією командою Docker.

9.Перевірка в браузері коректності роботи запущеного контейнера. Запропоновану технологію можна модифікувати під свої потреби.

3.ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДИКИ

3.1. Постановка практичної частини задачі

Додаток, який буде розроблено за запропонованою в попередньому розділі технології повинний відповідати наступним вимогам:

- значне скорочення часу на розгортання сайту;

- забезпечення мінімальної кількості дій розробника при розгортанні;

- наявність можливості масштабування;

- забезпечення ідентичності оточення сайту на продакшні і локальному сервері.

Після розробки буде проведена його апробація та оцінка за певними критеріями:

- Правильність формування конфігураційних файлів;

- Швидкість запуску контейнера з додатком;

- Час, витрачений на розгортання сайту локально;

- Наявність або відсутність помилок при запуску контейнера.

Для запуску сайту в контейнері після конфігурації файлів необхідний встановлений на комп'ютері Docker. Щоб використовувати Docker, потрібно перейти на офіційний сайт в розділ установки і пройти по простій інструкції. У Linux docker-compose необхідно встановлювати окремо, разом з Docker Engine він не ставиться. Docker-compose знадобиться надалі для запуску контейнерів.

В цілому робота Docker складається з декількох етапів:

1. Завантаження образів з Docker Hub або створення, як в нашому випадку.

2. Створення контейнерів.

3. Ініціалізація файлової системи.

4. Монтування рівнем тільки для читання.

5. Ініціалізація мережі

6. Встановлення IP-адреси.

7. Запуск додатку.

Як було сказано раніше, в цій роботі будуть використані сайти на CMS Wordpress, тому етап ідентифікації рушія визначений.

Далі потрібно створити структуру файлів для майбутнього програми. Вона буде виглядати як показано нижче:

| templates

| I Dockerfile.tpl

| I docker-compose.tpl

| I uploads.tpl

| I wp-config.tpl

| index.php

| env.php

| generator.php

| dumper.php

| results

Наступним етапом є розробка шаблонів для конфігурації. В папці templates знаходяться шаблони для Dockerfile, docker-compose, uploads і wp-config. Перші три шаблони будуть розглянуті докладно нижче. Шаблон wp-config є стандартним конфігураційним файлом CMS Wordpress.

Шаблони будуть розглянуті з точки зору вже готових файлів, необхідних для запуску контейнера. Шаблони відрізняються лише наявністю змінних, які будуть передавати дані з оточення сервера, отримані скриптом env.php і додані в data.json,

наприклад версії PHP і MySQL, паролі або вбудовані директиви PHP. Зміст файлів шаблонів буде представлено в лістингу.

Слід зазначити, що в інструкціях, які використовуються нижче, передбачається, що розробник має користувача Linux з привілеями sudo, тобто він може виконувати команди як привілейований користувач, також відомий як root.

Dockerfile.tpl. Даний шаблон призначений для створення Dockerfile, який є основним файлом для побудови образу. Конфігурація кожного контейнера описується як простий текстовий файл Dockerfile, що складається з набору спеціальних команд і на основі котрого будується контейнер. Dockerfile буде містити інформацію, необхідну для створення образу Apache + PHP. Цей образ можна завантажити з реєстру Docker, але він має безліч різних конкретно зазначених змін, наприклад, версія PHP, що суперечить розробленій технології, що забезпечує точне перенесення середовища з сервера.

У випадку з Wordpress найчастіше використовують офіційний образ Wordpress, який вже містить в собі зв'язку PHP + Apache, але він також має певну версію PHP, що не підходить в нашому випадку.

Образ Docker збирається на основі базового образу за допомогою інструкцій, які послідовно записані в Dockerfile. Кожна інструкція створює новий шар всередині образу. Зазвичай він починається з вказівки на образ, котрий потрібно наслідувати. Це може бути або базовий образ, або інший образ, який був створений раніше.

Для написання Dockerfile, необхідно трохи знати команди управління Linux, такі як «apt-get install». На даний момент Dockerfile має 13 інструкцій [11]. Ці команди також можуть виконуватися вручну через консоль. Деякі приклади інструкцій Dockerfile:

- FROM - встановлює базовий образ-систему. Параметр двокрапки використовується для вибору конкретної версії, найчастіше використовується: latest.

Latest - це вказівка на найновіший доступний спосіб, включаючи виправлення (патчі) безпеки і виправлення помилок.

- EXPOSE - відкриває порти для зв'язку з зовнішньою стороною і вмістом контейнера.

- RUN - виконує команду в контейнері. Після кожного застосування RUN стан образу зберігається. У рядку, наступному після RUN буде вже "оновлена" версія образу.

- WORKDIR - встановлює робочу директорію.

- VOLUME - створює точку монтування для томи.

- ADD - переміщує вміст, файли і цілі каталоги з батьківської системи в структуру файлу контейнера.

- CMD - Точка входу або команда для запуску при ініціалізації контейнеру.

- ENV - встановлює змінну оточення.

- COPY копіює файли з поточної директорії в файлову систему створюваного образу.

Інструкції можуть використовуватися не всі. Можна їх додавати або видаляти в залежності від потреб. Docker читає Dockerfile при запуску збірки (build) образу, виконує інструкції і повертає образ.

Послідовність інструкцій Dockerfile для докерізації додатку повинна виглядати приблизно так:

FROM php: 5.6-apache

RUN apt-get update && apt-get install -y libfreetype6-dev Hbjpeg62-turbo-dev libmcrypt-dev libpng12-dev \

&& pecl install xdebug \

&& docker-php-ext-install mbstring xml mcrypt mysqli gd \

&& docker-php-ext-configure gd --with-jpeg-dir = / usr / include / \

&& docker-php-ext-enable xdebug RUN a2enmod rewrite

В даному додатку перша і основна команда буде виглядати так: FROM php: <? Php echo $ php_image; ?> - apache, встановлюючи необхідну вер¬сію PHP, певну скриптом.

Розширення PHP також підставляються в шаблон в змінної, яка містить в собі масив розширень, встановлених на сайті.

docker-php-ext-instah mbstring zip <? php echo $ extensions; ?>

Розширення будуть взяти не всі, так як частина з них іде разом з образом PHP + Apache, тому були виділені найбільш часто використовувані расшіре¬нія PHP в зв'язці з Wordpress (список може розширюватися):

- xml

- gd

- mbyte

- mcrypt

- curl

- mysqli

Коректність побудови образу перевіряється за допомогою команди для побудови способу docker build, можна використовувати прапор -t, що використовується для вказвки імені образу. Список існуючих образів викликається командою docker images.

docker-compose. tpl

Використання одного Dockerfile підходить, коли досить побудови одного образу. У нашому випадку буде три контейнера: PHP + Apache, MySQL, PHPMyAdmin. Для того, щоб зв'язати їх, і щоб один контейнер міг підключитися до іншого, Docker пропонує інструмент під назвою Docker Compose для управління

мультіконтейнерними конфігураціями. Цей інструмент може бути викликаний командою $ sudo docker-compose.

При цьому у вас файл, який за замовчуванням називається docker-compose.yaml і повинен бути присутнім в каталозі, в якому виконується ця команда.

Docker Compose не є строго необхідним для зв'язку контейнерів, але дозволяє користувачеві краще організовувати і підтримувати ресурси.

Файл docker-compose.yaml, який сформується з шаблону, повинен починатися з версії, на поточний момент існує вже 3.1 версія формату. У docker-compose файлах буде використовуватися версія 3.

Першим кроком для запису Docker Compose є вибір образів, котрі будуть використовуватися, і їх версій. В цьому випадку будуть використовуватися наступне:

- web - образ від apache / php, побудований з Dockerfile

- db - офіційний образ mysql з DockerHub

- phpmyadmin - офіційний образ з DockerHub

Для образів db і web задаються параметри для бази даних в розділі environment. У нашому випадку дані прописуються в docker-compose, однак є ще варіант підключення окремого файлу з розширенням .env з змінними оточення, наприклад: env_file:

- ./db.env

depends_on використовується для запуску сервісів командою docker-compose up у заданій послідовності. У прикладі нижче образ db запуститися перед web. web:

depends_on:

- db volumes:

- ./app:/var/www/html

- ./config/uploads .ini: /usr/local/etc/php/conf.d/uploads .ini db:

volumes:

- ./db_data:/var/lib/mysql

- ./db_init:/docker-entrypoint-initdb.d

Всередині контейнера для даних створюються томи. Наприклад, каталог з іменем db_data всередині локальної папки проекту зіставляється з каталогом MySQL. Таким способом дані MySQL будуть зберігатися всередині проекту, так як інакше вони можуть загубитися при видаленні контейнера.

Порти задаються для образів web і phpmyadmin для доступу до запущених контейнерів через браузер. Порти дозволяють налаштувати взаємодію контейнера і хост-машини. І.Міелль в своїй книзі підкреслює, що за замовчуванням контейнери є повністю недоступними, ізольованими всередині Docker-машини. Спочатку вказує порт хост-машини, а через двокрапку порт контейнера. Наприклад, в даному випадку додаток буде доступно по адресу http://localhost:8000. Порт 8000 буде відображатися безпосередньо на порт 80 в контейнері. Надалі можна реалізувати рандомних генерацію портів для різних сайтів.

ports:- «8000: 80»

Порти також можна прокинути всередині Dockerfile наступним чином: EXPOSE 22 80 або при запуску контейнера з консолі: -p 22:22 -p 8080: 80. Сам по собі expose не відчиняє порти контейнера, але дає можливість «прів'язатися» до порту контейнера під час запуску контейнера.

uploads.tpl

Цей шаблон створює файл uploads.ini. Сюди заносяться вбудовані директиви php.ini, які використовуються для настройки PHP.

Згодом цей файл монтується в контейнер в docker-compose.yaml: ./uploads.ini: / usr / local / etc / php / conf. d / uploads. ini

Готовий файл буде складатися зі списку найбільш важливих директив з параметрами (параметри також будуть братися скриптом з середовища сервера):

- short_open_tag

Директива включає/відключає використання короткої форми записи тегів, тобто <?, а не <? Php.

- allow_url_fopen

Дана директива включає підтримку обгорток URL (URL wrappers), які дозволяють працювати з об'єктами URL як зі звичайними файлами.

- upload_max_filesize

Максимальний розмір файлу баз даних.

За замовчуванням максимальний розмір файлу обмежений 2 МБ. Це стосується і імпорту баз даних в phpMyAdmin, а бази іноді важать значно більше.

- post_max_size

Встановлює максимально допустимий розмір даних, що відправляються методом POST. Це значення також впливає на завантаження файлів. Для завантаження великих файлів це значення має бути більше значення директиви upload_max_filesize. По суті, memory_limit повинна бути більше ніж post_max_size.

- memory_limit

Ця директива задає максимальний обсяг пам'яті в байтах, який дозволяють використовувати скрипту. Це допомагає запобігти ситуації, при якій погано написаний скрипт «з'їдає» всю доступну пам'ять сервера.

- max_file_uploads

Максимально дозволена кількість одночасно закачуваних файлів.

Список може доповнюватися при необхідності, наприклад, можна також додати директиву "display_errors" з параметром "On". Показ помилок найчастіше вимкнений

на робочих сайтах, так як відвідувачам не потрібно бачити помилки і попередження, на тестовому ж сервері цей параметр можна включити.

Скрипт повинен бути написаний для чотирьох файлів: index.php, env.php, generator.php і dumper.php.

- index.php перевіряє права на запис в папках, підключає файли env.php, generator.php і dumper.php.

- env.php визначає параметри середовища, записує в окремий файл data.json і виводить дані користувачеві в браузер.

- generator.php - генерує конфігураційні файли з шаблонів, формує архів.

- dumper.php - займається експортом бази даних.

Після завантаження папки у програмі на сервер в директорію сайту потрібно запустити на виконання файл index.php. Як показано на рис.3., Користувач може подивитися інформацію про оточення і завантажити архів з готовими файлами конфігурації.

Розпаковану папку потрібно покласти в одну директорію на локальному комп'ютері з папкою, яка містить файли сайту. Нова файлова структура повинна виглядати так:

app

db-init

db.sql

Dockerfile

docker-compose.yml

uploads.ini

wp-config.php

На цьому етапі можна запускати контейнер з урахуванням встановленого на комп’пьютер docker і docker-compose. Контейнери можна запускати командою:

docker-compose up.

При цьому консольний висновок обох контейнерів буде видно прямо в консолі. З цього моменту контейнер запущений. Контейнер функціонує як своєрідна "віртуальна машина", де все всередині вже приготовлено для розробки. Перша збірка контейнера займає більше часу, однак наступні запуски відбуваються набагато швидше. Контейнером можна управляти і взаємодіяти з ним, використовувати сервіси і додатки, які він надає, а також при необхідності зупинити і видалити його. Перевага полягає в тому, що, якщо розробник випадково щось зламав, контейнер легко може бути перезапущений з початкового стану.

Зупиняються контейнери натисканням Ctrl + C, або ж командою docker-compose stop з іншої консолі, а віддаляються - командою docker-compose rm.

- docker-compose up -d

При використанні цієї команди контейнери будуть запускатися в фоновому режимі.

Щоб перевірити, які типи контейнерів запущені:

- docker-compose ps

- run docker-compose up -d --build

Команда для перебудови контейнерів при додаванні нових драйверів і розширень.

Ця команда видаляє також всі побудовані образи:

- sudo docker-compose down --remove-orphans --rmi 'all'

Щоб зайти всередину контейнера потрібно використовувати команду:

- sudo docker exec -ti wp_container / bin / bash

Основні прапори, які застосовуються при роботі з контейнерами:

- -t - прапор, який додає псевдотермінал всередині нового контейнера.

- -i - прапор, який дозволяє встановити інтерактивне з'єднання, взявши стандартне введення (STDIN) контейнера.

- --rw - прапор, автоматично видаляє контейнер після завершення процесу. За замовчуванням контейнери не видаляються. Цей контейнер існує, поки встановлена поточна сесія оболонки, і знищується, коли ми залишаємо сесію (наприклад, SSH-сесію з віддаленим сервером).

- -d - прапор запускає контейнер у фоновому режимі (демонізуватиме його).

- -f - прапор для rm, який зупиняє контейнер, якщо він працює (примусове видалення).

- -q - прапор для ps, який виводить тільки ID контейнерів.

При запуску команди, наприклад, $ docker run -i -t ubuntu / bin / bash, Docker Engine виконує наступні дії:

1. Викачує (pull) образ ubuntu: Docker Engine перевіряє наявність образу ubuntu в кеші на хості, на якому працює Docker Engine. Якщо образ уже існує локально, Docker Engine використовує його для запуску нового контейнера. В іншому випадку Docker Engine викачує цей образ з DockerHub.

2. Створює новий контейнер: Docker використовує зазначений образ для створення контейнера.

3. Виділяє місце на файлової системи хоста і монтує шар доступний для запису: Контейнер створений на файлової системи і доступний для запису шар доданий до його образу.

4. Виділяє мережевий/bridge інтерфейс: створює мережевий інтерфейс, який дозволяє Docker контейнеру взаємодіяти з локальним хостом, на якому встановлений Docker Engine.

5. Призначає IP адреси: знаходить і приєднує доступні IP-адреси.

6. Виконує процес, який ви визначили: виконує команду / bin / bash.

7. Захоплює і надає висновок програми: підключається і датує стандартний потік введення, виведення і помилок, щоб показати, як виконується ваш додаток, так як використовується інтерактивний режим. Далі сайт можна відкривати в браузері на заданому порту контейнера. Таким чином, алгоритм роботи програми на основі технології, розробленій в попередньому розділі, складатиметься з наступних кроків:

1. Завантаження папки з файлами програми на сервер, де розміщений сайт, в папку самого сайту.

2. Збір інформації про оточення при запуску скрипта і її висновок користувачу в браузер. Паралельно з цим процесом відбувається експортування бази даних.

3. Також одночасно зі збором інформації скрипт-генератор формує конфігураційні файли, необхідні для побудови контейнерів надалі.

4. Разом з виведенням інформації про оточення з'являється кнопка, при натисканні на яку користувач може завантажити архів, що містить файли для конфігурації контейнера.

5. Разархіварованная папка міститься в локальну папку з файлами сайту, який необхідно розгорнути локально.

6. За допомогою однієї docker-команди сайт запускається в контейнері, використовуючи згенеровані конфігураційні файли, і стає доступним при відкритті на порту localhost, який був визначений в налаштуваннях. Такий підхід, який можна розширити практично для будь-якого типу стека, має деякі переваги в порівнянні зі звичайним ручним налаштуванням:

- Користувачеві не потрібно піклуватися про встановлення кожного окремого компонента і залежностей, оскільки в образах контейнера вже є все необхідне для запуску стека.

- Весь стек підсумовується в одному файлі (docker-compose.yml). Це зручно, наприклад, для оновлення версій деяких компонентів або зміни властивостей бази даних.

- Файл docker-compose.yml може бути перенесений на інші комп'ютери за допомогою Docker Compose, і він буде створювати той же самий стек з тими ж версіями. Якщо у локального демона вже є деякі образи контейнера, він може повторно їх використовувати.

Підводячи підсумки, варто сказати, що Docker може спростити життєвий цикл розробки, дозволяючи розробникам працювати в стандартизованих середовищах, використовуючи локальні контейнери, в яких може працювати додаток або сервіси. Контейнер створює образ додатки, ідентичний для всіх типів систем і варіантів використання. Docker також можна інтегрувати в процеси безперервної інтеграції (continuous integration) і безперервної доставки (continuous deployment). Розробник може запускати образ додатки локально на робочому комп'ютері в середовищі, ідентичною використовуваної в продакшені. Використовуючи docker, розробники можуть писати код і ділитися своєю роботою з колегами за допомогою контейнерів. Також Docker можна використовувати для включення додатків в тестові середовища і запуску автоматичних і ручних тестів. Коли розробники знаходять проблему, вони можуть виправити її в своєму оточенні і повторно помістити контейнери в тестове оточення.

У практичному прикладі стека для WordPress з Docker Compose було показано, як Docker корисний, коли один файл у вигляді простого тексту представляє собою цілий стек і його сполуки.

Розроблений додаток відповідає висунутим вимогам, воно виводить час, який потрібен розробнику на розгортання сайту. Розробнику потрібно здійснювати меншу кількість дій в порівнянні з ручним розгортанням сайту на локальному сервері. Серед сайтів на комп'ютері розробника ідентична його середовищі на сервері, що зводить до мінімуму появу помилок.

3.2. Апробація створеного додатка

У попередніх розділах була розроблена методика автоматичної конфігурації docker-контейнерів і на її основі створено програму, яка може значно оптимізувати роботу розробників.

Код програми доступний за посиланням: https://bitbucket.org/ann_caly/docker-configurator /.

Додаток було апробовано, і в таблиці 4 представлені результати даної апробації на підставі критеріїв, які були виділені в 3 чолі. Робота програми була перевірена на 5 сайтах, заснованих на CMS Wordpress.

Табл. 3 Результати апробації застосунку

Сервіс	Коректне Формування файлів	Час запуску контейнеру	Час на розгортання сайту	Відсутність помилок
cosmed-	-	85 сек.	~ 10 мин.	-
iqpromo.ru	+	67 сек.	~ 4 мин.	+
antelas.ru	+	79 сек.	~ 4 мин.	+
nefafood.ru	+	83 сек.	~ 5 мин.	+
100bike.ru	+	101 сек.	~ 7 мин.	-

Результати тестування показали, що середній час, який потрібен на запуск контейнера - в межах двох хвилин. Слід врахувати, це перший запуск контейнера, який зазвичай відбувається довше, так як docker спочатку будує образ з Dockerfile, встановлює всі необхідні бібліотеки і роширення, монтує дані. Наступні запуски займають набагато менше часу.

Середній час на розгортання локальної версії сайту - 5 хвилин, довше - у випадках наявності помилок. Розгортання сайту розробників вручну займає близько 20 хвилин. Трапляється, що на вивчення і виправлення помилок потрібна не одна

годину. Такі ситуації найчастіше бувають через відмінності оточення сайту на локальному сервері, конфліктів версій і бібліотек.

Помилки при розгортанні з'явилися в двох випадках. У першому випадку через не зовсім вірно сконфигурированного Dockerfile. Збірка способу по ньому завершилась з наступною помилкою:

ERROR: Service 'web' failed to build: The command '/ bin / sh -c docker-php-ext-install pdo pdo_mysql gd' returned a non-zero code: 1

При дослідженні стало зрозуміло, що помилка виникає, коли установник починає перевіряти gd залежності. Для виправлення помилки було встановлено пакет Linux libpng-dev за допомогою apt. У Dockerfile додано рядок перед завантаженням розширень PHP:

RUN apt-get update -y && apt-get install -y libpng-dev

Також, для додавання бібліотеки curl було додано curl розширення PHP плюс пакети curl і libcurl4-openssl-dev через apt.

Це відбувається тому, що ряд розширень PHP використовують базові частини операційної системи Linux. Щоб мати можливість використовувати ці розширення, необхідно також встановити компоненти ОС.

У другому випадку з помилкою при розгортанні сайту 100bike.ru, помилка була виправлена зміною файлу .htaccess. Коли даний сайт розгортався раніше вручну, то було набагато більше помилок через різні версії PHP і відсутніх пакетів.

Підводячи підсумки можна сказати, що додаток пройшов апробацію успішно - скоротився час на розгортання сайтів і виникнення помилок в процесі.

Помилки, які були виявлені при тестуванні, дозволили трохи вдосконалити додаток шляхом модифікації шаблону Dockerfile задля уникання подібних помилок в майбутньому. Помилки, що виникли з бібліотеками gd і curl можуть виникати і з іншими розширеннями.

Однак, в роботі практично будь-якої програми часто виникають помилки, особливо на ранніх етапах його роботи, тому в подальшому за цим потрібно стежити і своєчасно вносити виправлення і поліпшення. Додаток можна масштабувати, а також ще більше автоматизувати процес локального розгортання файлу з використанням даної технології.