УДК 681.3

О.Б. Придибайло А.П. Бондарчук

РОЗРОБКА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СЕРВІСНО - ОРІЄНТОВНОГО ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

У статті розглянуто доцільність поширення та розгортання додатків за допомогою ізольованих процесів, які використовують ядро операційної системи для їх запуску. Проаналізовано відмінності контейнеризації від віртуальних машин та доведені переваги контейнеризації. Також розглянуті основні технології для реалізації віртуальних середовищ в операційній системі Linux, які дозволяють працювати з технологіями сервісно-орієнтованого програмування.

Ключові слова: сервісно-орієнтоване програмування, контейнеризація, віртуалізовані додатки, віртуальна машина, сервер, контейнер.

Вступ

Розробка і виробництво програми - досить трудомісткий і непередбачуваний процес, в якому часто зустрічається безліч труднощів і перешкод. Крім рутинних завдань з розробки можуть виникнути проблеми, пов'язані з відстеженням залежностей, масштабуванням, оновленням компонентів і тому подібними процесами.

Система контейнеризації та сервіс-орієнтоване проектування (СОП) призначені для вирішення більшості цих проблем. З їх допомогою додаток можна розділити на окремі компоненти, які, в свою чергу, можна упакувати разом з усіма залежностями. Такі компоненти в подальшому легко розгортаються на будь-якій архітектурі. Ця технологія значно спрощує процеси масштабування і оновлення компонентів.

Основна частина

Донедавна контейнери застосовувалися в основному на стадії розробки і тестування програмного забезпечення. Це спрощувало розробникам завдання, дозволяючи прискорити процес створення програм завдяки прозорості контролю над будь-якими параметрами обчислювального середовища в процесі розробки. Після цього готові продукти переносилися в віртуальні машини і вони були готові до експлуатації. [1,2]

Головними відмінностями програмних контейнерів від віртуальних машин є:

1. Середовище, в якому працює система.

Якщо запуск здійснюється у вигляді контейнера, то додаток потрапляє в те ж операційне середовище, в якому працюють інші додатки розробника і з якими відбувається взаємодія. Якщо ж запуск здійснюється у вигляді віртуальної машини на сервері, то будь-які звернені в її сторону запити з боку контактують додатків проходять довгий шлях з однієї операційного середовища в іншу і назад.

Перехід на використання програмних контейнерів і віртуальних машин покликаний вирішити головну задачу: виділити програмний об'єкт з базового середовища, щоб вести його обробку в особливих, оптимальних для нього умовах. Але, якщо контейнери дозволяють відокремити об'єкт від системної програмної оболонки, що працює на сервері, то віртуальна машина служить для зовсім інших цілей: вона підміняє апаратні можливості їх віртуальними абстракціями, щоб зробити процес обробки незалежним від реальних фізичних особливостей використовуваного обладнання.

2. Різниця в особливостях роботи клієнтів.

У разі вибору контейнера клієнт спільно використовує загальні ресурси базового ядра операційної системи; загальними іноді також є бінарні файли і бібліотеки модулів. У разі вибору віртуальної машини клієнт наділяється власним примірником програмного ядра операційної системи і інших супутніх бібліотек, тобто він працює абсолютно незалежно від базової середовища сервера.

3. Кількість клієнтських модулів, які можуть взаємодіяти одночасно з програмним об'єктом, перенесеним в віртуальне середовище.

Для контейнерів кількість модулів виявляється більше, ніж при виборі варіанту віртуальної машини, через те, що пам'ять, яка виділяється для роботи клієнтів не резервується жорстко, а управляється з боку базової операційної системи. Коли робота ведеться з віртуальною машиною, то взаємодія ускладнюється і кількість одночасно запущених клієнтів в ній стає менше, тому що пам'ять жорстко прив'язується до ресурсів віртуальної машини.

4. Час завантаження і переходу в активний режим.

Ця відмінність є самою істотною з точки зору споживчих властивостей. У разі контейнерів цей період завантаження та переходу обчислюється секундами, для віртуальної машини - хвилинами.[3,4]

Перевагами контейнеризації є:

1. Контейнеризація дозволяє автоматично робити однаковими середовище експлуатації і середовище розробки. Тобто додатки працюють в тому ж операційному середовищі, в якому велася їх розробка. Це дає гарантію відсутності проблем, пов'язаних з переходом від розробки до реального застосування

2. Контейнеризація дозволяє набагато спростити прив'язку всіх параметрів налаштування конфігурацій додатків.

3. Контейнеризація дозволяє легко відстежувати версійність впроваджуваного додатка, якщо його подальша розробка триває. У цьому випадку на сервері створюється спеціальний розділ, куди складуються образи нових версій програми. Якщо оновлення дає збій, то можна легко замінити додаток, що був зіпсований результаті збою, на колишню робочу версію.

4. Оформлення віртуалізованих додатків у вигляді набору незалежних мікросервісів. Кожен з них виконує в цьому випадку тільки певне, вузьке завдання.

5. Готовність до масштабування.

Для підвищення масштабованості і гнучкості розгортання контейнерів призначено виявлення сервісів (service discovery) - одного з компонентів загальної стратегії,. Завдяки цьому контейнери можуть оцінити стан поточного середовища без допомоги адміністратора. Вони можуть знайти інформацію про підключення компонентів, з якими вони повинні взаємодіяти, а також повідомити про себе іншим інструментам. Такі інструменти зазвичай виступають в якості розподілених сховищ конфігурацій, які дозволяють встановлювати довільні параметри конфігурацій для сервісів деякої інфраструктури.

Після реєстрації контейнера додатки можуть встановити з'єднання за допомогою системи пошуку послуг.

Ці інструменти часто використовуються як сховища типу «ключ-значення» і розподіляються по всім вузлам мережі (хостам) кластера. Зазвичай такі сховища надають HTTP-інтерфейс для доступу до даних. Деякі з них надають додаткові функції безпеки: шифрування, контроль доступу і т.п. Розподілені сховища необхідні для управління кластерізованим оточенням сервісно-орієнтованого проектування.

Завдання сховищ пошуку послуг:

• Передача додатків даних, необхідних для взаємодії з залежностями.

• Реєстрація даних про підключення сервісів.

• Надання загальнодоступного сховища конфігурацій.

• Зберігання даних про кластер, які необхідні для програм управління ним.

Додатки, поміщені в контейнери, приймають сервіс-орієнтовану архітектуру, яка передбачає поділ функціоналу на окремі компоненти. Це значно спрощує управління і масштабування додатків, проте вимагає надійної мережі між компонентами.

Вбудовані мережеві функції і можливості сервісно-орієнтованого проектування надають два механізми зв'язування контейнерів:

1. Відкрити порти контейнера і налаштувати їх для взаємодії з вузлом мережі для зовнішньої маршрутизації. Це універсальний спосіб надання доступу до контейнера, відповідний в більшості випадків.

2. Дозволити контейнерам взаємодіяти за допомогою посилань сервісно-орієнтованого проектування. З їх допомогою контейнер буде отримувати дані про підключення на іншій стороні зв'язку. Це дозволить йому підключатися автоматично, якщо він відслідковує зміни. Таким чином можна встановлювати взаємодію між контейнерами на одному вузлу мережі і при цьому не ставити порт або адресу сервісу заздалегідь.

Контейнеризацією та ізоляцією компонентів Unix-подібні операційні системи користуються достатньо активно.

Для реалізації віртуальних середовищ у Linux існує декілька технологій:

- KVM
- Xen
- Linux VServer
- OpenVZ
- LXC (Linux Containers)
- Але основою Linux для розробки контейнеризації є система LXC, додана в ядро в 2008 році. LXC є відомим набором інструментів, шаблонів та бібліотек. Цей рівень підтримується в розробці ядра та є готовим до розробки з LXC 1.0 з 5-річною підтримкою оновлень безпеки та виправлень (до квітня 2019 року).

Таблиця 1-Характеристики технологій віртуалізації

Технологія	**Розробник**	**Ліцензія**	**Тип ізоляції (isolation)**	**Рівень інтеграції в ядро Linux**
KVM	RedHat (раніше Qumranet), OVA	GPLv2	full virtualization	повна, mainstream (весь код підтримується в коді ядра) з 2007 року
Xen	XenSourse, Citrix, XenProject	GPLv2	para/full virtualization	DomU (клієнт) код з 2012 року
Linux Server	Herbert Potzl,	GPLv2	os-level virtualization, containerization	Не mainstream, проект закрито
OpenVZ	SwSoft, containerization Parallels	GPLv2	os-level virtualization, containerization	90% інтеграції в mainstream, ведеться активна робота
Linux Containers, LXC	Kernel.org: Intel, IBM, Google, Parallels	GPLv2	os-level virtualization, containerization	повна, mainstream

Для забезпечення ізоляції процесів система LXC скомбінувала механізми ядра Linux:

- namespaces - механізми, які забезпечують роботу контейнерів всередині ядра
- cgroops - механізми, які обмежують різні ресурси контейнера (обсяг зайнятої пам'яті, навантаження на диск, навантаження на центральний процесор).

З початку листопаду 2014 року і до цього дня знаходиться в дуже активному розвитку є система LXD (скорочення от Linux Container Daemon), яка є представляє собою надбудову над LXC. Вона спрощує роботу з контейнерами і додає широкий спектр нових можливостей за допомогою всього одного інструмента: командного рядка для управління своїми контейнерами. LXD призначений для запуску повних операційних систем.

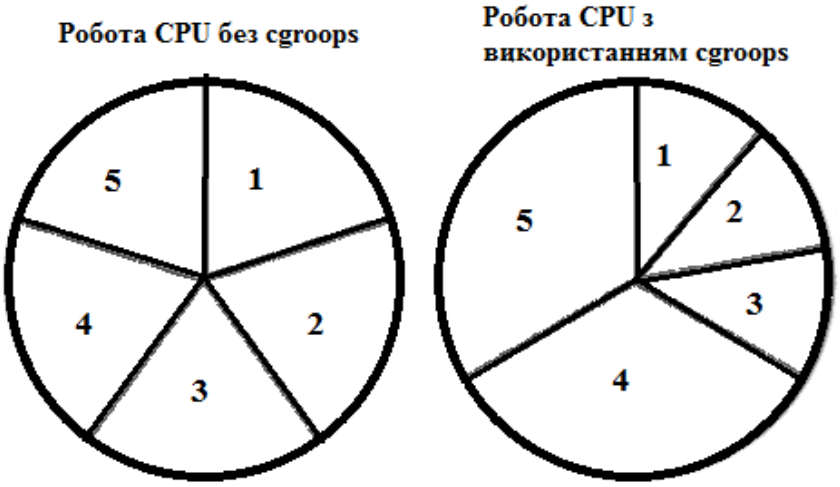
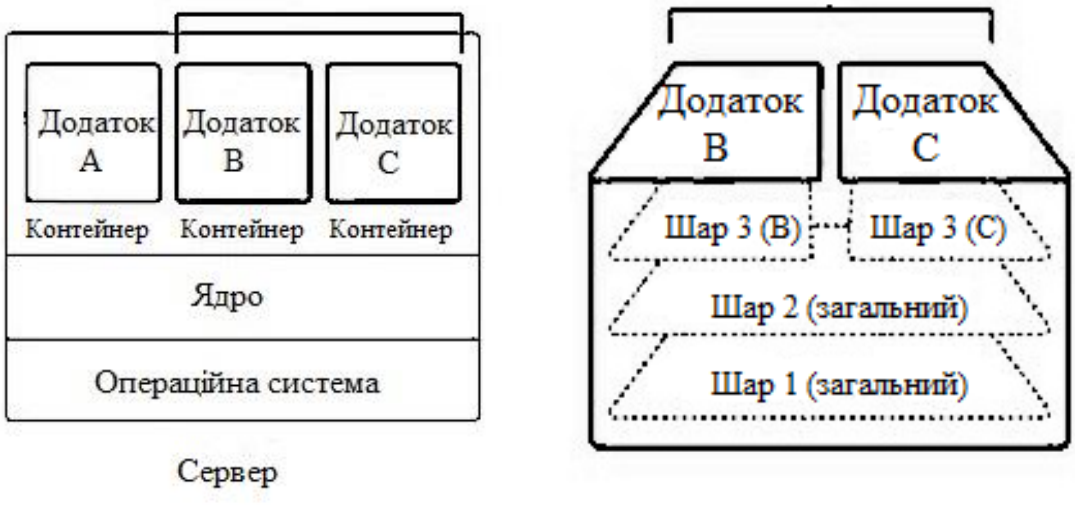


Рис. 1. Розподіл ресурсів контейнерів

На початку 2014 року отримало визнання сервісно-орієнтовне проектування (СОП), яке показало на основних операціях (по масовому створенню, перезавантаженню та знищенню віртуальних вузлів) істотно вищу продуктивність, ніж KVM. Зокрема, на тест масового створення віртуальних обчислювальних вузлів збільшення використання процесорних ресурсів у сервісно-орієнтовному проектуванні (СОП) зафіксовано в 26 разів нижче, ніж у KVM, а приріст споживання ресурсів оперативної пам'яті - втричі нижче. З 2017 року до вільно поширюваної під ліцензією Apache 2.0 редакції продукту випускається редакція для організацій. [5]



а) б)

Рис. 2. Загальний вигляд сервера з контейнерами та вміст контейнера.

Висновок.

Отже даний підхід дозволяє вирішити проблему оптимізації використання апаратного та програмного забезпечення за рахунок високої продуктивності та можливості зберігання інформації на хмарних платформах замість жорстких дисків на комп'ютерах та серверах. Це дозволяє отримати не тільки набір високоефективних інструментів, а також цілу екосистему додаткових програмних продуктів та сервісів, що допомагає розробляти запускати додатки, використовуючи технологію віртуалізації з перших кроків розробки програми.

Список використаних джерел:

1. Ишкина Е. Г., Щербинина О. В. Архитектура адаптивного сервисно-ориентированного промежуточного программного обеспечения //Известия Волгоградского государственного технического университета. – 2010. – Т. 11. – №. 9.

2. Грекул В. И., Пырлина И. В. Сервисно-ориентированное моделирование функционирования центров обработки данных //Бизнес-информатика. – 2010. – №. 1.

3. Феофантов К. В., Власов А. В., Афанасьев Г. И. Создание Docker-образа PostgreSQL //Современные научные исследования и инновации. – 2017. – №. 2. – С. 86-89.

4. Merkel D. Docker: lightweight linux containers for consistent development and deployment //Linux Journal. – 2014. – Т. 2014. – №. 239. – С. 2.

5. Dua R., Raja A. R., Kakadia D. Virtualization vs containerization to support paas //Cloud Engineering (IC2E), 2014 IEEE International Conference on. – IEEE, 2014. – С. 610-614.

Надійшла: 11.01.2018 Рецензент: к.т.н. Довбешко С.В.