УДК 004.42

О. Б. ПРИДИБАЙЛО, здобувач,
Державний університет телекомунікацій, Київ

СТВОРЕННЯ ДОДАТКІВ У СЕРВІСНО-ОРІЄНТОВАНОМУ ПРОЕКТУВАННІ

У статті розглянуто, в який спосіб сервісно-орієнтоване проектування та віддалені веб-сервери застосовуються для розгортання додатків на персональному комп'ютері користувача. Сервісно-орієнтоване проектування вигідне тим, що при роботі користувачів із додатками вже немає необхідності в установленні певного програмного забезпечення на локальних комп'ютерах, оскільки створення додатків відбувається на незалежному програмному забезпеченні за допомогою віртуалізації. У технології, яка розглядається в цій статті, існує велика бібліотека стандартних функцій та модулів, за допомогою яких програмується необхідна для створення програмних додатків файл-інструкція. Кожна файл-інструкція проектується в певному порядку, містить певні службові слова, команди, розділові та спеціальні знаки. Етапи формування файл-інструкції достатньо нескладні та зрозумілі завдяки існуванню необхідних стандартних бібліотек, які під'єднуються до файла за допомогою виклику їх за іменем. Створення програмних додатків також складається з певних етапів, які включають у себе запуск процесу створення образу, перевірку існуючих образів, перевірку на помилки та запуск уже зібраного й перевіреного додатка. У статті розглядаються також етапи контейнеризації програмних додатків та механізми, що запускають додатки в процес роботи. Механізми запуску додатків ґрунтуються на двох базових підсистемах, таких як простір імен і контрольні групи. У свою чергу, простір імен, необхідний для забезпечення ізоляції процесів, дозволяє створювати окремі відгалуження дерева процесів із власним персональним ідентифікаційним номером (PID), який у дереві процесів є кореневим. Створення контрольних груп має на меті оптимізацію використання існуючих ресурсів, а також перевірку наявності вільного місця для розташування програмних додатків.

Ключові слова: сервісно-орієнтоване проектування; ізоляція програмних додатків; середовище налаштування; операційна система; образ системи; проміжні образи; програмний додаток; файл-інструкція; контейнер; системи контейнеризації; дерево процесів; персональний ідентифікаційний номер, простір імен; батьківський простір імен; дочірній простір імен; контрольні групи; ізоляція ресурсів.

Вступ

Процес розгортання додатків, написаних будь-якою мовою програмування, складається з певної послідовності кроків, а саме: установлення необхідної операційної системи; установлення вебсервера (наприклад, apache); установлення сервера бази даних (наприклад, mysql); установлення розширення для apache; установлення інших необхідних розширень; конфігурація virtualhost (одного чи кількох); перенесення файлів програми, конфігурація.

Виконати ці кроки досить складно, особливо якщо йдеться про встановлення специфічних розширень або розгортання існуючого проекту без документації.

Уникнути таких трудомістких дій дає змогу ***сервісно-орієнтоване проектування*** — технологія, яка дозволяє упакувати додаток у ізольований контейнер. За його наявності відпадає необхідність установлювати і конфігурувати все потрібне програмне забезпечення на локальному комп'ютері, і це ніяк не позначиться на якості роботи з додатком.

Основна частина

Сервісно-орієнтоване проектування залежно від завдання пропонує створити образ (зображення) настроєної системи з проектом усередині чи без нього. Для цього формується файл-інструкція, що включає в себе всі необхідні команди для налаштування середовища та проекту.

Після створення образу його потрібно тільки запустити на хост-машині (головній ЕОМ, що підтримує інформаційні та розрахункові ресурси і надає їх віддаленим користувачам), скориставшись сервісно-орієнтованим проектуванням. Це може бути і виробничий сервер, і робоча станція програміста чи тестувальника тощо.

Запущений екземпляр образу називається ***контейнером***. Контейнер — це самостійний ресурс всередині поточної операційної системи. Він складається з операційної системи, установлених розширень і файлів користувача. Контейнер здатний і запускати процеси.

Етапи, необхідні для створення та запуску додатків, унаочнює схема, подана на рис. 1.

Етап введення даних виконує дії з основними (вбудованими) командами, які будуть використані в основному файлі.

Наступний етап — ***створення файл-інструкції***:

1) створення образу за інструкціями, які записуються в спеціальний файл;

2) створення самої файл-інструкції, що включатиме в себе всі необхідні команди для налаштування оточення і проекту.

Файл-інструкція — це текстовий документ, що містить всі команди, які користувач може викликати в командному рядку для ***складання***

Рис. 1. Етапи створення додатків

зображення. Зображення в сервісно-об'єктному проектуванні буде створено автоматично, після прочитання програмним забезпеченням інструкцій [1; 2].

Далі йде виконання ***етапу створення та складання зображення***.

Дії на решті етапів очевидні (див. рис. 1).

Файл-інструкція має певний формат:

Comment INSTRUCTION arguments.

Інструкція не чутлива до регістру. І все ж існує негласна домовленість стосовно того, що регістри мають відігравати важливішу роль, щоб легше було відрізняти інструкції від аргументів.

У сервісно-орієнтованому проектуванні інструкції виконуються в певному порядку. Файл повинен починатися з інструкції FROM. Команда FROM визначає базовий образ, з якого потім створюється необхідна конфігурація. FROM може передувати одна або кілька інструкцій ARG, які оголошують аргументи, що використовуються в рядках FROM у сервісно-орієнтованому проектуванні.

Кожний рядок, який починається зі знака #, сприймається як коментар, за винятком випадків, коли ця лінія є директивою Парсера, поданою об'єктно-орієнтованою скриптовою мовою програмування, яку було створено для генерування HTML-сторінок на веб-сервері. Якщо елемент # з'являється десь іще в рядку, то він розглядається як аргумент. Це дозволяє записувати такі, скажімо, команди:

Comment RUN echo ' we are running some # of cool things'

Символи продовження рядка не підтримуються в коментарях.

Приклад файл-інструкції

```
FROM wacken/meteor:base
RUN
mkdir /opt/meteor
COPY . /opt/meteor
WORKDIR /opt/meteor
EXPOSE 3000
ENTRYPOINT ["/usr/local/bin/meteor",
"--allow-superuser"]
```

Після створення файл-інструкції переходять до створення та складання образів на виконання ***команди складання***. При цьому діємо в такий спосіб:

1) запускаємо командою процес, після чого переглядаємо всі кроки файл-інструкції. У результаті маємо отримати повідомлення про успішне створення образу. Йому буде присвоєно унікальний ідентифікатор (ID);

2) перевіряємо існуючі образи.

Після кожної інструкції в сервісно-орієнтованому проектуванні створюються проміжні образи. Вони зберігаються і за потреби можуть бути повторно використані. Саме завдяки цій особливості сервісно-орієнтоване проектування виступає як легке й гнучке вирішення порівняно зі звичайними віртуальними машинами.

Коли образ створено, він підлягає перевірці щодо помилок в інструкціях. Якщо їх не виявлено, контейнер запускається на виконання. Усі

інструменти контейнеризації, такі як сервісно-орієнтоване проектування, LXC або systemd-nspawn, мають у своїй основі дві підсистеми ядра Linux: *простір імен* і *контрольні групи* [3].

Простір імен — це механізм ядра Linux, що забезпечує ізоляцію процесів один від одного. Роботу з його реалізації було розпочато у версії ядра 2.4.19. Нині Linux підтримує шість типів ізолювального простору імен:

1. **PID** — простір PID процесів.
2. **NETWORK** — мережні пристрої, стеки, порти тощо.
3. **USER** — ID користувачів і груп.
4. **MOUNT** — точки монтування.
5. **IPC** — SystemV IPC, черги повідомлень POSIX.
6. **UTS** — ім'я хост-машини та ім'я домена NIS.

При запуску програм усі типи простору імен використовують сучасні системи контейнеризації: сервісно-орієнтоване проектування, Docker, LXC та ін.

Історично в ядрі Linux підтримувалося лише одне дерево процесів — ієрархічна структура, подібна до дерева каталогів файлової системи.

Із появою механізму простору імен уможливилась підтримка кількох дерев процесів, повністю ізольованих один від одного.

При завантаженні в Linux спочатку запускається процес із персональним ідентифікаційним номером (PID) 1. У дереві процесів він виступає як кореневий, запускаючи інші процеси та служби. Механізм простору імен дозволяє створювати окреме відгалуження дерева процесів із власним PID 1. Процес, який створює таке відгалуження, є частиною основного дерева, але його дочірній процес вже буде кореневим у новому дереві.

Процеси в новому дереві ніяк не взаємодіють з батьківським процесом і навіть не «бачать» його. Натомість процесам в основному дереві доступні всі процеси дочірнього дерева. Цей факт унаочнює рис. 2 [4].

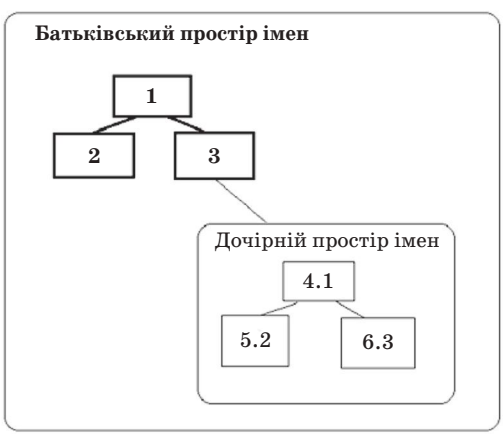

Рис. 2. Ізоляція ресурсів у просторах імен

Утім для контейнеризації самої лише ізоляції ресурсів недостатньо. Адже при запуску будь-якої програми в ізольованому оточенні потрібно мати впевненість у тому, що для неї виділено достатньо ресурсів і що вона не споживатиме не передбачених для неї ресурсів, порушуючи тим самим роботу всієї системи.

Для вирішення таких проблем у ядрі Linux є спеціальний механізм — ***контрольні групи***, що включає в себе дві складові: *ядро* і так звані *підсистеми*.

Кожна підсистема являє собою директорію з керуючими файлами, в яких прописуються всі налаштування. У кожній із цих директорій є керуючі файли, які дозволяють передавати дочірнім контрольним групам властивості батьківських, а також містять список персональних ідентифікаційних номерів усіх процесів, включених у контрольні групи, і список TGID груп процесів, включених у контрольні групи. Зазначені файли дозволяють відправляти повідомлення в разі зміни статусу контрольної групи; містять команду, яку буде виконано, якщо включено опцію notify_on_release; містять бульову змінну (0 або 1), що вмикає (або, навпаки, відмикає) виконання команди, зазначеної в release_agent. У кожної підсистеми є також власні керуючі файли.

Щоб створити контрольну групу, достатньо створити вкладену директорію в будь-якій із підсистем. У цю вкладену директорію будуть автоматично додані керуючі файли. Додати процеси в групу дуже просто: потрібно лише записати відповідний PID у керуючий файл tasks.

Сукупність контрольних груп, убудованих у підсистему, становить певну ієрархією. Після проходження перевірок та збирання контейнера його знову потрібно перевірити на помилки. Якщо таких не виявлено, контейнер запускається на виконання [5].

Висновок

Пропонований підхід дозволяє розв'язати проблему оптимізації використання як апаратного, так і програмного забезпечення за рахунок високої продуктивності та можливості зберігання інформації на хмарних платформах замість жорстких дисків на комп'ютерах і серверах. Усе це дає змогу отримати набір високоефективних інструментів, у поєднанні з екосистемою додаткових програмних продуктів і сервісів, що, у свою чергу, допомагає розробляти й запускати додатки, використовуючи технологію віртуалізації з перших кроків розробки програми.

Список використаної літератури

*1. **Ишкина Е. Г., Щербинина О. В.** Архитектура адаптивного сервисно-ориентированного промежуточного программного обеспечения // Известия Волгоград. гос. техн. ун-та. 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), IEEE International Conference. 2014. С. 610–614.*

Рецензент: канд. техн. наук, доцент **К. П. Сторчак,** Державний університет телекомунікацій, Київ.

О. Б. Придыбайло

СОЗДАНИЕ ПРИЛОЖЕНИЙ В СЕРВИСНО-ОРИЕНТИРОВАННОМ ПРОЕКТИРОВАНИИ

В статье рассмотрены как сервисно-ориентированное проектирование и отдаленные веб-сервера применяются для развертывания программных приложений на персональном компьютере пользователя. Сервисно-ориентированное проектирование выгодно тем, что при работе пользователей с программными приложениями уже нет необходимости в установке определенного программного обеспечения на локальных компьютерах, поскольку создание программных приложений происходит на независимом программном обеспечении с помощью виртуализации. В технологии, рассматриваемой в данной статье, существует большая библиотека стандартных функций и модулей, с помощью которых программируется необходимая для создания программных приложений файл-инструкция. Каждая файл-инструкция проектируется в определенном порядке, содержит в себе определенные служебные слова, команды, разделительные и специальные символы. Этапы формирования файл-инструкции достаточно несложные и понятные, благодаря существованию необходимых стандартных библиотек, которые подключаются к файлу с помощью вызова их по имени. Создание программных приложений также состоит из определенных этапов, которые включают в себя запуск процесса создания образа, проверку существующих образов, проверку на ошибки и запуск уже собранного и проверенного программного приложения. В статье рассматриваются также этапы и механизмы контейнеризации программных приложений, основанных на двух базовых подсистемах, таких как пространство имен и контрольные группы. В свою очередь, пространство имен, необходимое для обеспечения изоляции процессов, позволяет создавать отдельные ответвления дерева процессов с собственным персональным идентификационным номером (PID), который в дереве процессов является корневым. Создание контрольных групп имеет целью оптимизацию использования существующих ресурсов, а также проверку наличия свободного места для размещения программных приложений.

Ключевые слова: сервисно-ориентированное проектирование; изоляция программных приложений; среда настройки; операционная система; образ системы; промежуточные образы; программное приложение; файл-инструкция; контейнер; системы контейнеризации; дерево процессов; персональный идентификационный номер; пространство имен; родительское пространство имен; дочернее пространство имен; контрольные группы; изоляция ресурсов.

O. B. Prydybailo

CREATING AN APPLICATIONS IN SERVICE-ORIENTED DESIGNING

The article discusses both service-oriented design and remote web servers used for deploying software applications on a user's personal computer. Service-oriented design is advantageous because users of software applications no longer need to install certain software on local computers, since the creation of software applications takes place on independent software through virtualization. In the technology that is discussed in this article, there is a large library of standard functions and modules, with which programming is required to create software applications file instruction. Each file instruction is designed in a specific order, contains certain service words, commands and punctuation, and special characters. The steps in creating a file-instruction are quite simple and understandable, due to the existence of the necessary standard libraries that connect to the file by calling them by name. The steps to create a file-command are simple because of the existence of the required standard libraries. Creating software applications also consists of certain steps, which include launching the process of creating an image, checking existing images, checking for errors, and launching an already assembled and verified software application. Also, the article discusses the steps and mechanisms for containerization of software applications based on two basic subsystems, namely, namespaces and control groups. In turn, the namespace needed to ensure process isolation allows you to create separate branches of the process tree with its own personal identification number (PID), which is in the process tree in the root. The creation of control groups is aimed at optimizing the use of existing resources, as well as checking the availability of free space for the placement of software applications

Keywords: service-oriented design; isolation of software applications; setup environment; operating system; system image; intermediate images; software application; file instruction; container; containerization systems; process tree; personal identification number; namespace; parent namespace; child namespace; control groups; resource isolation.