

Лёгковесность Docker-контейнеров как фактор эффективности современных программных систем

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассматривается феномен лёгковесности контейнерных технологий на примере Docker. Анализируются архитектурные причины снижения накладных расходов по сравнению с виртуальными машинами, приводится классификация контейнерных образов и рассматриваются практические подходы к минимизации размера и времени запуска контейнеров. Приводятся сравнительное описание для облегчённых базовых образов, таких как alpine, slim и stretch дистрибутивов, и рассматриваются их методы использования в различных сценариях эксплуатации.

Ключевые слова: Docker, контейнеризация, изолированная среда.

Lightweight Docker containers as a factor of efficiency of modern software systems

Vikhlyayev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article examines the phenomenon of lightweight container technologies using Docker as an example. The architectural reasons for reducing overhead costs compared to virtual machines are analyzed, a classification of container images is given, and practical approaches to minimizing the size and startup time of containers are considered. A comparative description is provided for lightweight basic images such as alpine, slim, and stretch distributions, and their methods of use in various operating scenarios are considered.

Keywords: Docker, containerization, isolated environment.

1 Введение

1.1 Актуальность

Современные программные системы всё чаще строятся на принципах микросервисной архитектуры, непрерывной интеграции и доставки (CI/CD), а также масштабируемого облачного развертывания. В этих условиях критическое значение приобретают время запуска сервисов, потребление вычислительных ресурсов, объём передаваемых и хранимых артефактов, предсказуемость и воспроизводимость окружения. Традиционные

виртуальные машины, несмотря на высокий уровень изоляции, характеризуются значительными накладными расходами, связанными с эмуляцией аппаратных ресурсов и наличием полноценной гостевой операционной системы. Контейнерные технологии, в частности Docker, предлагают альтернативный подход, обеспечивающий существенно меньшие издержки за счёт совместного использования ядра хостовой системы.

1.2 Обзор исследований

В своей работе И.Н.Николаева провёл минимизацию размера образов контейнеров на примере Docker образов nginx и grafana [1]. Г.Д.Дыдалин оптимизировал размер Docker контейнера для развертывания нейронной сети [2]. А.Е.Буц исследовал методы оптимизации контейнеров Docker, включая управление слоями образов, минимизацию их размера и эффективное использование кэша [3]. М.К.Солонько, К.А.Краснов в своей работе продемонстрировали структурную составляющую Docker-контейнеров [4]. Б.Ю.Егоров, Е.В.Таций, И.Ю.Анохина использовали Docker для создания контейнера python приложения [5].

1.3 Цель исследования

Цель исследования – систематизация подходов к обеспечению лёгковесности Docker-контейнеров и анализ существующих разновидностей контейнерных образов с точки зрения их эффективности.

2 Материалы и методы

В описании рассматриваются облегчённые базовые образы alpine, slim, stretch в платформе для контейнеризации Docker. Рассматриваются архитектурные особенности Docker-контейнеров, обеспечивающие их легковесность.

3 Результаты и обсуждения

Принцип легковесности Docker образов основывается на том, что контейнеры используют ядро хостовой операционной системы и не включают его внутрь себя, за счёт чего отпадает необходимость хранить и загружать большое количество системных компонентов, которые характерны для виртуальных машин. Образ содержит набор слоёв файловой системы. Каждый слой содержит только изменения относительно предыдущего, поэтому одинаковые части могут пере использоваться между разными образами, что уменьшает общий объём занимаемого места и ускоряет доставку.

Использование минимальных базовых образов, которые содержат только самый необходимый набор библиотек и инструментов существенно уменьшает размер разворачиваемого впоследствии контейнера. Дополнительно можно поместить в образ зависимости, которые нужны приложению для работы, а не для разработки или сборки. Тогда итоговый размер образа и в последствии контейнера уменьшится ещё больше, но при этом потребуется написание более сложной конфигурации для сборки. Этот

метод способствует проведению сборки приложения отдельно от его запуска. За счёт этого временные файлы и инструменты не попадают в итоговый результат. В итоге Docker образ становится небольшим быстро загружаемым и простым в распространении, что напрямую влияет на скорость развертывания и экономию ресурсов системы. Так называемая многостадийная сборка, при которой приложение компилируется или подготавливается в одном образе, а в финальный образ копируется только результат сборки без исходных кодов и инструментов компиляции. Здесь важную роль играет корректное написание Dockerfile, поскольку каждая инструкция создаёт слой. Если установка пакетов и очистка временных файлов выполняются раздельно, лишние данные остаются в слоях, поэтому их нужно удалять в рамках одной команды. Затем исключаются ненужные файлы из контекста сборки с помощью `dockerignore`, чтобы в образ не попадали логи, тесты и вспомогательные каталоги. В результате образ становится компактнее, быстрее собирается, передаётся и запускается.

Чтобы сделать образы легче, из операционной системы вырезают лишние файлы, не требовательные для разработки программ. Например, Slim-образы строятся на известных дистрибутивах типа Debian или Ubuntu, но из них исключены документации, локализации и сторонние утилиты. При этом остаётся поддержка экосистемы пакетов и удобства для разработчиков. Более суровый подход на примере минималистичных дистрибутивов, используется в Alpine Linux. Данный дистрибутив создан специально для контейнеров. Он содержит библиотеку `musl`, урезанную версию стандартной библиотеки `glibc` и упрощённого набора инструментов `BusyBox`. Этого оказывается достаточно для большинства решений, но могут возникнуть проблемы с совместимостью пакетов и функций.

Ещё один способ уменьшения веса – создание `runtime-only` образов, содержащих только самое необходимое для запуска приложения, без инструментов разработки и сборки. Многоступенчатые сборки позволяют разделить процесс на два этапа: сначала собирается приложение, потом оно упаковывается для выполнения. Так снижается общий размер и повышается уровень защиты.

Чтобы достичь максимального эффекта по памяти используют `scratch`-образы. Они работают без операционной системы, имеются только минимальные компоненты для запуска исполняемого файла. Docker запускает бинарный файл напрямую как процесс `id` (PID) с номером 1. Возникает “голый” системный вызов к ядру хоста. Такой метод подходит для программ, состоящих из одного исполняемого файла без внешних зависимостей. Такие программы должны быть написаны на языках программирования поддерживающих статическую компиляцию, таких как Go или Rust.

В таблице 1 представлено сравнение образов типизированных по занимаемой постоянной памяти.

Таблица 1. Сравнительная таблица лёгковесных Docker-образов

Тип образа	Базовая ОС / libc	Размер образа	Уровень совместимости	Основные области применения	Характер лёгковесности
Full	glibc	100–300 МБ	Очень высокий	Разработка, отладка	Отсутствует
Slim	glibc	20–80 МБ	Высокий	Production-сервисы, микросервисы	Удаление лишних компонентов
Debian stretch / stable	glibc	60–150 МБ	Очень высокий	Корпоративные системы	Стабильность и предсказуемость
Alpine	musl	5–10 МБ	Средний	Микросервисы, serverless	Минималистичный дистрибутив
Runtime-образы	glibc / musl	10–50 МБ	Высокий	Production-контейнеры	Отказ от build-инструментов
Scratch	отсутствует	< 5 МБ	Низкий	Инфраструктурные утилиты	Полный отказ от ОС

Slim-образы являются сбалансированным решением для большинства production-систем. Они поддерживают максимально высокое соответствие пакета инструментов, что и полные образы. В Alpine-образах обеспечивается максимальная компактность, но их использование требует контроля совместимости. Использование полных версий Debian, таких как stretch, всё ещё является приоритетным, где есть сомнения в поддержке и критичны стабильность и предсказуемость поведения среды выполнения.

3 Вывод

В результате исследования представлена типизация Docker-образов по методу уменьшения занимаемого пространства постоянной памяти. Приведена сравнительная таблица легковесных типов образов. Даны описания стратегий реализующих минимизацию пользовательского пространства, замену стандартных библиотек, разделение этапов сборки и выполнения, а также полный отказ от универсальной операционной среды.

Библиографический список

1. Николаева И.Н. Минимизация размера образов контейнеров на примере Docker // Процессы управления и устойчивость. 2021. Т. 8. № 1. С. 303-307.
2. Дыдалин Г.Д. Оптимизация размера Docker контейнера для развертывания нейронной сети // В сборнике: Тинчуринские чтения - 2025 «Энергетика и цифровая трансформация». статьи по материалам Международной молодежной научной конференции. В четырех томах. Казань, 2025. С. 46-49.
3. Буц А.Е. Оптимизация работы Docker контейнеров // Парадигма. 2025. № 3-2. С. 3-8.
4. Солонько М.К., Краснов К.А. Docker-контейнеры // Телекоммуникации и информационные технологии. 2020. Т. 7. № 1. С. 65-69.
5. Егоров Б.Ю., Таций Е.В., Анохина И.Ю. Докеризация python приложения // В сборнике: Современные информационные технологии в образовании и научных исследованиях (СИТОНИ-2023). Сборник материалов VIII

- Всероссийской научно-технической конференции. Донецк, 2023. С. 98-102.
6. Титанов П.О. Преимущества использования docker контейнеров в разработке ПО // В сборнике: Проблемы управления в социально-экономических и технических системах. Материалы XX Международной научно-практической конференции. Сборник научных статей. Саратов, 2024. С. 453-455.
 7. Мареев Н.А. Сравнение контейнерных оркестраторов для реализации микросервисной архитектуры // В сборнике: Технологии информационного общества. Сборник трудов XVII Международной отраслевой научно-технической конференции. 2023. С. 228-230.