И. А. Михневич

КОНТЕЙНЕРИЗАЦИЯ ПРИЛОЖЕНИЙ С ПРИМЕНЕНИЕМ DOCKER

Работа выявляет факторы, влияющие на надёжность контейнеризированных приложений.

Введение

Docker Engine представляет собой совокупность взаимодействующих подсистем, каждая из которых определяет свойства контейнеризированной среды[1]. Daemon Docker управляет жизненным циклом контейнеров и образов, взаимодействуя с ядром Linux через cgroups, namespaces и драйверы хранения[1]. Docker Client обеспечивает доступ к API демона[1], а среда хоста определяет доступные механизмы изоляции и ограничения ресурсов[1]. В рамках работы анализируется, как эти механизмы влияют на изоляцию процессов, устойчивость контейнеров и безопасность приложений.

I. Слоистая структура образов

Docker использует слоистую модель хранения, в которой каждый слой является неизменяемым. В ходе экспериментов было установлено, что количество слоёв и порядок инструкций в Dockerfile напрямую влияют на скорость сборки и размер итогового образа. Перестановка инструкций, связанных с зависимостями, в начало файла позволяет значительно сократить время повторной сборки. Применение многостадийной сборки уменьшает размер образов в несколько раз. Это демонстрирует, что архитектурные решения внутри Dockerfile оказывают существенное влияние на эффективность контейнеризации.

II. Механизмы изоляции

Изоляция контейнеров обеспечивается комбинацией PID-, network- и mount-namespaces, а также ограничениями ресурсов через cgroups. Исследование показало, что контейнеры без ограничений ресурсов способны потреблять вычислительные мощности хоста вплоть до деградации системы. Ограничение CPU и памяти снижает риск отказа сервисов и делает поведение контейнеров более предсказуемым. Запуск контейнеров от имени root увеличивает поверхность атаки, поскольку процессы внутри контейнера получают расширенные возможности взаимодействия с хостом.

III. Уязвимости

Анализ базовых образов показал, что даже официальные образы могут содержать уязвимости. Проверка зависимостей Java-приложений выявила, что риски могут быть связаны не с Docker, а с библиотеками, используемыми внутри контейнера. Эксперименты также подтвердили, что контейнеризация не защищает от логических ошибок в приложении: SQL-инъекции, XSS и удалённое выполнение кода остаются актуальными угрозами. Отдельное внимание было уделено Dockerfile, где ошибки конфигурации (root-пользователя или копирование лишних файлов) приводят к увеличению поверхности атаки.

Для анализа безопасности были использованы инструменты Docker Bench for Security, Trivy и Snyk. Исследование показало, что Docker Bench наиболее эффективен для проверки конфигурации хоста, тогда как Trivy обеспечивает наиболее полное покрытие уязвимостей образов. Это позволяет утверждать, что комплексное использование инструментов анализа является обязательным условием безопасной эксплуатации контейнеров.

IV. Выводы

Исследование показало, что надёжность зависит от состояния хоста, на котором работает Docker, поскольку именно ядро обеспечивает изоляцию и управление ресурсами. Существенную роль играет качество и структура образов: чем меньше слоёв, зависимостей и лишних пакетов, тем стабильнее среда. Конфигурация контейнеров также влияет на устойчивость — ограничения ресурсов, корректные права доступа и отказ от запуска от root делают работу предсказуемее. Наконец, важным остаётся состояние самого приложения и его зависимостей, поскольку уязвимости и ошибки внутри кода сохраняются независимо от контейнеризации.

Список литературы

1. Docker Documentation [Электронный ресурс] // Docker Docs. – Режим доступа: https://docs.docker.com/. – Дата доступа: 29.03.2026.

Михневич Иван Александрович, студент 3 курса ФИТиУ БГУИР, ivanmichnevih2@gmail.com.

Научный руководитель: Радченко Анастасия Дмитриевна, ассистент кафедры ВМиП БГУИР, a.radchenko@bsuir