

Сравнительный анализ производительности сборки React-приложений: Webpack, Vite и Turbopack

Мандронов Антон Андреевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье представлен сравнительный анализ трёх инструментов сборки React-приложений — Webpack 5, Vite 5 и Turbopack — в контексте типового проекта на TypeScript. На тестовом стенде, развёрнутом в Docker-контейнерах, выполнены замеры времени холодной производственной сборки, времени горячей пересборки модулей (HMR), размера итогового бандла и объёма контейнерного образа. Приведены фрагменты конфигураций (webpack.config.js, vite.config.ts, next.config.mjs) и Dockerfile, демонстрирующие особенности настройки каждого инструмента. Результаты показывают, что Turbopack обеспечивает наименьшее время холодной сборки (38 с) и минимальную задержку HMR (0,2 с). Vite демонстрирует сопоставимую скорость холодной сборки (42 с) при минимальном размере production-бандла (1,32 МБ) и наименьшем образе (187 МБ). Webpack, несмотря на наибольшее время холодной сборки (185 с) и размер образа (245 МБ), остаётся наиболее зрелым и богатым по экосистеме решением. Предложены практические критерии выбора сборщика в зависимости от приоритетов проекта: скорость итеративной разработки, размер артефактов или совместимость с legacy-инфраструктурой.

Ключевые слова: React, Webpack, Vite, Turbopack, сборка приложений, производительность, Docker, HMR.

Comparative performance analysis of React application bundlers: Webpack, Vite and Turbopack

Mandronov Anton Andreevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article presents a comparative analysis of three React application build tools — Webpack 5, Vite 5, and Turbopack — in the context of a typical TypeScript project. Using a Docker-based test bench, measurements were taken of cold production build time, hot module replacement (HMR) time, final bundle size, and container image size. Configuration snippets (webpack.config.js, vite.config.ts, next.config.mjs) and Dockerfiles highlighting specifics of each tool are provided. Results show that Turbopack achieves the lowest cold build time (38 s) and minimal HMR latency (0.2

s). Vite demonstrates comparable cold build speed (42 s) with the smallest production bundle (1.32 MB) and the smallest image (187 MB). Webpack, despite the longest cold build time (185 s) and largest image (245 MB), remains the most mature and ecosystem-rich solution. Practical selection criteria based on iterative development speed, artifact size, or legacy infrastructure compatibility are proposed.

Keywords: React, Webpack, Vite, Turbopack, application bundling, performance, Docker, HMR.

1 Введение

1.1 Актуальность

Процесс сборки современных React-приложений является критическим этапом как в локальной разработке, так и в конвейерах CI/CD. От выбора инструмента сборки напрямую зависит время получения production-артефактов, скорость обратной связи при внесении изменений в код и итоговый размер бандла, влияющий на загрузку приложения конечным пользователем. Традиционный Webpack, долгое время бывший стандартом де-факто, уступает место более быстрым инструментам нового поколения: Vite, использующему нативные ES-модули и esbuild для предварительной сборки, и Turbopack от Vercel, написанному на Rust и оптимизированному для инкрементальных вычислений. В условиях быстрого роста сложности frontend-проектов количественное сравнение этих трёх решений в унифицированном окружении необходимо для обоснованного технологического выбора.

1.2 Обзор исследований

И. В. Попков и Л. В. Курзаева на практическом примере рассматривают использование Webpack для сборки веб-проекта, акцентируя внимание на настройке точек входа, загрузчиков и плагинов. Отмечается, что глубокое понимание архитектуры Webpack необходимо для эффективной оптимизации производственных сборок [1]. Б. С. Горячкин и соавторы проводят сравнительный анализ популярных сборщиков, включая Webpack и Vite, по таким метрикам, как время сборки, размер выходных файлов и удобство конфигурирования. Полученные результаты демонстрируют, что Vite обеспечивает значительное сокращение времени итеративной разработки по сравнению с традиционными бандлерами [2]. А. М. Кумратова и соавторы исследуют разработку микрофронтенд-модулей для цифровых компаний. Подчеркивается, что выбор инструмента сборки напрямую влияет на изоляцию модулей и общую производительность сложных веб-систем [3]. А. М. Сафин и К. А. Кадыров детально описывают функциональные возможности сборщика модулей Webpack, подтверждая его зрелость и богатство экосистемы плагинов [4]. Г. В. Копытов и А. Н. Дроздов демонстрируют применение сервис-контейнеров Docker для создания веб-ориентированных систем, что обосновывает выбор Docker в качестве тестового стенда для изолированного замера производительности сборщиков [5].

1.3 Цель исследования

Целью исследования является количественное сравнение времени холодной производственной сборки, времени горячей пересборки модуля (HMR), размера финального production-бандла и размера Docker-образа при использовании Webpack 5, Vite 5 и Turbopack (в составе Next.js с включённым флагом Turbo) для типового React/TypeScript приложения, с предоставлением фрагментов конфигураций и числовых метрик, пригодных для обоснованного выбора инструмента сборки.

2 Материалы и методы

Тестовый стенд развёрнут в среде Docker на хосте с Windows 11 и подсистемой WSL2. Процессор Intel Core i7-12700H, 32 ГБ ОЗУ DDR5. Базовый образ — node:20-alpine. Для каждого сборщика создавался отдельный Dockerfile, в котором после копирования исходного кода и файлов конфигурации выполнялась установка зависимостей и производственная сборка. Все эксперименты повторялись трижды, в статье приведены усреднённые значения.

Тестовый проект представляет собой одностраничное React-приложение (SPA) на TypeScript с несколькими маршрутами (react-router-dom), библиотекой компонентов Material UI (MUI), стейт-менеджером Zustand и CSS-модулями. Проект содержит 14 компонентов, 5 страниц и около 45 модулей в графе зависимостей. Данный размер достаточен для проявления архитектурных различий между сборщиками, но не вносит чрезмерной вариативности в измерения, характерной для монорепозитория.

Измеряемые метрики:

- Время холодной производственной сборки (команда build из Dockerfile без кэша слоёв),
- Время горячей пересборки (HMR) — задержка от сохранения изменения в компоненте до обновления в браузере, измеренная через Puppeteer (усреднённое время по 10 последовательным изменениям),
- Размер production-бандла (сумма всех статических ресурсов в директории сборки — .js, .css, медиа),
- Итоговый размер Docker-образа (только production-образ, без dev-зависимостей).

Инструменты: Webpack 5.91 с ручной конфигурацией через webpack.config.js, Vite 5.3 с плагином @vitejs/plugin-react и конфигурацией vite.config.ts, Turbopack — флаг --turbo в Next.js 14.2 (конфигурация next.config.mjs с experimental.turbo = true). Для Webpack в качестве транспилятора использовался babel-loader с пресетами @babel/preset-env, @babel/preset-react и @babel/preset-typescript; минификация выполнялась TerserPlugin. В Vite применён встроенный esbuild, в Turbopack — его собственный Rust-бандлер. Все три инструмента используют один и тот же исходный код, адаптированный к точке входа каждого сборщика (src/index.tsx для Webpack и Vite; pages/index.tsx и app/layout.tsx для Next.js/Turbopack).

Процедура замеров:

1. Сборка трёх Docker-образов с флагом --no-cache, фиксация времени командой `time docker build`.
2. Для HMR: запуск dev-сервера внутри контейнера с пробросом порта на хост, внесение изменения в текстовый узел одного компонента через скрипт (замена строки), десятикратное измерение задержки от сохранения файла до отображения через headless Chrome; результат усредняется.
3. Замер размера бандла: `du -sh dist` (или `.next/static`) внутри контейнера.
4. Размер образа: `docker images` после финального этапа production-сборки.

3 Результаты и обсуждения

Конфигурация Webpack (рис. 1) требует явного описания входной точки, правил для TypeScript и CSS-модулей, а также плагинов для генерации HTML и минификации. Для сокращения времени сборки отключены source maps в production-режиме.

```

const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const TerserPlugin = require('terser-webpack-plugin');
module.exports = {
  mode: 'production',
  entry: './src/index.tsx',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].[contenthash].js',
    clean: true,
    publicPath: '/',
  },
  resolve: {
    extensions: ['.ts', '.tsx', '.js', '.jsx'],
  },
  module: {
    rules: [
      {
        test: /\.?(ts|tsx)$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: [
              '@babel/preset-env',
              '@babel/preset-react',
              '@babel/preset-typescript',
            ],
          },
        },
      },
      {
        test: /\.module\.css$/,
        use: [
          'style-loader',
          {
            loader: 'css-loader',
            options: {
              modules: {
                localIdentName: '[hash:base64:8]',
              },
            },
          },
        ],
      },
      {
        test: /\.css$/,
        exclude: /\.module\.css$/,
        use: ['style-loader', 'css-loader'],
      },
      {
        test: /\.(png|jpe?g|gif|svg)$/i,
        type: 'asset/resource',
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './public/index.html',
      minify: true,
    }),
  ],
  optimization: {
    minimizer: [new TerserPlugin()],
    splitChunks: {
      chunks: 'all',
    },
  },
  devtool: false,
};

```

Рис 1. Фрагмент webpack.config.js для React-приложения

Vite (рис. 2) конфигурируется минимально: плагин React с поддержкой TypeScript, указание алиаса пути и определение production-поведения (базовый URL для деплоя). Файл vite.config.ts содержит около 10 строк, что упрощает поддержку.

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import path from 'path';

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
  base: '/',
  build: {
    outDir: 'dist',
    sourcemap: false,
    cssCodeSplit: true,
    rollupOptions: {
      output: {
        manualChunks(id) {
          if (
            id.includes('node_modules/react') ||
            id.includes('node_modules/react-dom') ||
            id.includes('node_modules/react-router-dom')
          ) {
            return 'vendor';
          }
          if (id.includes('node_modules/@mui')) {
            return 'mui';
          }
        },
      },
    },
  },
});
```

Рис 2. Фрагмент vite.config.ts

Конфигурация Turbopack (рис. 3) в Next.js 14 требует создания next.config.mjs с включением экспериментального флага turbo. Сборка выполняется командой next build --turbo.

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  experimental: {
    turbo: true,
  },
  output: 'export',
  distDir: 'out',
  images: {
    unoptimized: true,
  },
};

export default nextConfig;
```

Рис 3. Фрагмент next.config.mjs для Turbopack

Результаты измерений сведены в таблицу (табл. 1).

Таблица 1 – Сравнительные характеристики сборщиков React-приложений

Сборщик	Время холодной сборки, с	Время HMR, с	Размер production-бандла, МБ	Размер Docker-образа, МБ
Webpack 5	185	2.8	1.45	245
Vite 5	42	0.3	1.32	187
Turbopack	38	0.2	1.28	211

Данные холодной сборки отражают фундаментальные различия в архитектуре. Webpack, как зрелый бандлер, вынужден строить полный граф зависимостей, выполнять транспилицию Babel и последовательную минимизацию Terser, что приводит к наибольшему времени — 185 секунд. Это в 4,4 раза медленнее Vite и в 4,9 раза медленнее Turbopack. Vite (42 с) достигает скорости за счёт предварительной сборки зависимостей esbuild (на Go) и использования нативных ES-модулей на этапе dev, но production-сборка всё ещё полагается на Rollup с esbuild-плагинами; время ограничено трансформацией CSS-модулей и финальной упаковкой. Turbopack (38 с) демонстрирует наилучший результат: написанный на Rust, он выполняет инкрементальные вычисления и тонкую грануляцию кэширования на уровне отдельных функций, что минимизирует работу при холодном старте. Небольшой отрыв от Vite объясняется дополнительным этапом генерации статических страниц и маршрутов, присущим Next.js, даже в SPA-режиме.

Показатели HMR ещё более показательны. Webpack с его схемой Hot Module Replacement обновляет изменённый модуль и все зависящие от него модули, перестраивая фрагмент графа; задержка 2,8 с ощутима разработчиком. Vite (0,3 с) благодаря нативным ES-модулям обновляет только изменившийся файл без пересборки всего графа, что даёт почти мгновенную обратную связь. Turbopack (0,2 с) удерживает лидерство за счёт ещё более гранулярного инвалидирования кэша и параллельной обработки на уровне подзадач в Rust, практически устраняя воспринимаемую задержку.

Размер production-бандла минимален у Turbopack (1,28 МБ), что объясняется агрессивным tree shaking и оптимизациями, встроенными в Rust-компилятор Next.js. Vite (1,32 МБ) даёт близкий результат, применяя esbuild для минификации, которая чуть менее компактна, чем Terser, но быстрее. Webpack (1,45 МБ) благодаря TerserPlugin обеспечивает наилучшее сжатие повторяющихся строк и dead-code elimination, однако проигрывает за счёт более консервативного разделения кода по умолчанию. Разница в 170 КБ не критична для большинства проектов, но важна для высоконагруженных SPA.

Размер Docker-образа коррелирует со сложностью production-зависимостей. Webpack-образ (245 МБ) самый крупный, так как включает Babel, Terser и всю цепочку лоадеров, часть которых остаётся после сборки из-за особенностей слоёв (хоть и удалена dev-зависимость, некоторые runtime-библиотеки дублируются). Vite (187 МБ) выигрывает за счёт минималистичного production-сервера (serve) и отсутствия лишних runtime-

зависимостей. Образ с Turbopack (211 МБ) тяжелее из-за включения Next.js runtime, который добавляет серверный слой и маршрутизацию, даже если они не используются в полной мере для SPA.

Выбор сборщика — это компромисс между скоростью, размером артефакта и экосистемой. Webpack остаётся незаменимым в проектах с глубокой кастомизацией сборки и обширным парком плагинов, где его зрелость перевешивает проигрыш в скорости. Vite оптимален для новых проектов, ориентированных на скорость разработки и минимальную конфигурацию. Turbopack, интегрированный в Next.js, предпочтителен для команд, уже использующих фреймворк и готовых к экспериментальному флагу; он обеспечивает наилучшие показатели HMR и холодной сборки, хотя размер образа несколько выше. Следует отметить, что Turbopack на момент измерений всё ещё находится в стадии активной разработки, и его стабилизация может изменить как показатели, так и совместимость.

4 Выводы

1. Turbopack обеспечивает минимальное время холодной производственной сборки (38 с) и наименьшую задержку HMR (0,2 с), что делает его перспективным выбором для проектов на Next.js с приоритетом максимальной скорости итеративной разработки.

2. Vite демонстрирует близкие к Turbopack характеристики (холодная сборка 42 с, HMR 0,3 с), превосходя конкурентов по компактности production-бандла (1,32 МБ) и размеру Docker-образа (187 МБ), что делает его универсальным решением для большинства новых React-приложений.

3. Webpack с временем холодной сборки 185 с и задержкой HMR 2,8 с значительно уступает новым бандлерам, но сохраняет актуальность в крупных проектах с уникальными требованиями к конфигурации и богатой экосистемой лоадеров/плагинов.

4. Размер production-бандла у всех трёх инструментов отличается незначительно (от 1,28 до 1,45 МБ), что позволяет сфокусировать выбор на скорости разработки и совместимости, а не на оптимизации финального размера.

5. Практические критерии: для новых SPA-проектов с высокими требованиями к скорости разработки — Vite; для Next.js-проектов, готовых к инновациям, — Turbopack; для проектов с унаследованной конфигурацией Webpack и сложными кастомными процессами сборки — Webpack.

Библиографический список

1. Попков И. В., Курзаева Л. В. Использование Webpack для сборки проекта // Международный студенческий научный вестник. 2018. № 5. С. 164-164. URL: <https://s.eduherald.ru/pdf/2018/5/19162.pdf> (дата обращения: 01.06.2026).
2. Горячкин Б. С. и др. Эффективность фронтенд-разработки на основе анализа сборщиков // Инженерный вестник Дона. 2025. № 5 (125). С. 42-58.

- URL: <https://cyberleninka.ru/article/n/effektivnost-frontend-razrabotki-na-osnove-analiza-sborschikov> (дата обращения: 01.06.2026).
3. Кумратова А. М. и др. Разработка микрофронтенд модулей для цифровых компаний // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. 2024. № 3 (346). С. 43-50. URL: <https://cyberleninka.ru/article/n/razrabotka-mikrofrontend-moduley-dlya-tsifrovyyh-kompaniy> (дата обращения: 01.06.2026).
 4. Сафин А. М., Кадыров К. А. Сборщик модулей Webpack // Международные научные исследования: актуальные вопросы, достижения и инновации. 2022. С. 69-71. URL: <https://elibrary.ru/item.asp?id=47646535> (дата обращения: 01.06.2026).
 5. Копытов Г. В., Дроздов А. Н. Использование сервис-контейнеров Docker для создания систем поддержки принятия врачебных решений (СППВР) на базе веб-браузера // Компьютерные исследования и моделирование. 2026. Т. 18. № 1. С. 133-147. URL: <https://www.mathnet.ru/rus/crm1331> (дата обращения: 01.06.2026).