Multi-Stage Docker Builds: When to Use vs When to Avoid

Decision Framework Table

Project Type	Use Multi-Stage?	When TO Use	When NOT to Use	Size Impact
Frontend Apps				
React/Vue/Angular	✅ **Yes**	Building from TypeScript/JSX source, bundling assets, optimizing for production	Serving pre-built static files, development environment setup	500MB+ → 20-50MB
Static Site Generators (Gatsby, Next.js)	✅ **Yes**	Generating static sites from markdown/templates, image optimization	Simple HTML/CSS sites without build process	300MB+ → 15-30MB
Backend Applications				
Node.js with TypeScript	✅ **Yes**	Compiling TS to JS, installing dev dependencies, running build scripts	Simple JS files without compilation step	200MB+ → 80-120MB
Go Applications	✅ **Yes**	Compiling to binary, using build tools and dependencies	Already have pre-compiled binary	300MB+ → 5-15MB
Java/Spring Boot	✅ **Yes**	Maven/Gradle builds, creating JAR files, dependency resolution	Running pre-built JAR files	400MB+ → 150-200MB
Python with Dependencies	✅ **Usually**	Installing build tools, compiling native extensions, creating wheels	Simple scripts with pure Python libraries	250MB+ → 100-150MB
APIs & Services				
REST/GraphQL APIs	✅ **Yes**	When written in compiled languages or requiring build steps	Simple script-based APIs (Flask, Express without build)	Varies by language
Microservices	✅ **Yes**	Complex services with shared libraries, optimization needed	Single-responsibility services with minimal dependencies	200MB+ → 50-100MB
Specialized Applications				
CLI Tools (Go, Rust, C++)	✅ **Yes**	Creating standalone binaries, cross-compilation	Interpreted script-based tools	300MB+ → 2-10MB
Machine Learning Models	✅ **Usually**	Model optimization, dependency pruning, custom builds	Simple inference with standard libraries	1GB+ → 200-500MB
Desktop Apps (Electron)	✅ **Yes**	Building and packaging application, asset optimization	Development or debugging	500MB+ → 200-300MB

Project Type	Use Multi-Stage?	When TO Use	When NOT to Use	Size Impact
			environments	
Infrastructure & Services				
Web Servers (Nginx, Apache)	❌ **No**	Custom modules, special compilation needs	Standard configuration with official images	No benefit
Databases (MySQL, PostgreSQL)	❌ **No**	Custom extensions, special builds	Standard database deployment	No benefit
Message Queues (Redis, RabbitMQ)	❌ **No**	Custom plugins, special compilation	Standard message queue setup	No benefit
Monitoring Tools (Prometheus, Grafana)	❌ **No**	Custom dashboards, plugin development	Standard monitoring deployment	No benefit
Development & Testing				
Development Environments	❌ **Usually No**	Need debugging tools and build capabilities at runtime	Creating separate prod/dev images from same Dockerfile	Counter-productive
Testing Containers	❌ **Usually No**	Require test frameworks and tools during execution	Building test artifacts for external use	Counter-productive
CI/CD Build Agents	❌ **No**	Need all build tools available during runtime	—	Counter-productive
Content & Documentation				
Documentation Sites (Hugo, Jekyll)	✅ **Yes**	Generating static sites, theme compilation, asset processing	Serving pre-generated documentation	200MB+ → 15-25MB
CMS Applications	✅ **Sometimes**	Custom themes, plugin compilation, asset optimization	Standard CMS deployment without customization	150MB+ → 80-120MB
PHP Applications				
Laravel Applications	✅ **Yes**	Frontend asset compilation (Mix/Vite), Composer dev deps, artisan optimizations	Simple Laravel apps without frontend builds	400MB+ → 150-200MB
Symfony Applications	✅ **Yes**	Webpack Encore builds, cache warming, autoloader optimization	Basic Symfony apps without asset compilation	350MB+ → 140-180MB

Project Type	Use Multi-Stage?	When TO Use	When NOT to Use	Size Impact
WordPress (Modern)	✅ **Sometimes**	Custom themes with build processes, Composer dependencies	Traditional WordPress without build tools	300MB+ → 120-160MB
CodeIgniter	❌ **Usually No**	Complex apps with extensive tooling and asset builds	Standard lightweight CodeIgniter deployments	Minimal benefit
CakePHP	✅ **Sometimes**	Asset compilation, extensive testing, optimization workflows	Simple convention-based apps without builds	250MB+ → 100-140MB
PHP-Apache (General)	✅ **Yes**	Modern apps with Composer deps, asset builds, dev tools	Simple PHP scripts without dependencies or build steps	300MB+ → 120-180MB
Legacy PHP Applications	❌ **Usually No**	Custom builds with clear separation of concerns	Tightly coupled build/runtime dependencies	Counter-productive

Quick Decision Rules

✅ USE Multi-Stage When:

- Your build requirements are significantly larger than runtime requirements
- You're compiling code (TypeScript → JavaScript, Go → binary, Java → JAR)
- You're bundling/optimizing assets (webpack, rollup, image optimization)
- Security is important (reducing attack surface by removing build tools)
- You're working with package managers that install dev dependencies
- Final image size matters for deployment speed or storage costs

❌ AVOID Multi-Stage When:

- Build and runtime requirements are nearly identical
- You're using official pre-built images without modification
- Development environments where you need build tools available
- Simple scripts without compilation or bundling steps
- Legacy applications with tightly coupled build/runtime dependencies

- Time constraints don't allow for optimization complexity

Size Impact Categories

- **Dramatic Reduction (10x+):** Compiled languages (Go, Rust, C++) to minimal base images
- **Significant Reduction (3-5x):** Frontend applications, complex build processes
- **Moderate Reduction (2-3x):** Framework applications, bundled backends
- **Minimal Benefit (<2x):** Simple interpreted applications, pre-built services