To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Migrating JavaScript to TypeScript

Class Duration

14 hours of live training delivered over 2 days.

Student Prerequisites

- Professional JavaScript development experience
- Basic TypeScript familiarity (TypeScript Essentials or equivalent self-study)
- Experience working in an existing production codebase
- Familiarity with npm tooling and CI pipelines

Target Audience

This course is designed for developers and tech leads responsible for moving existing JavaScript codebases to TypeScript. It suits teams maintaining long-lived applications, platform engineers planning a monorepo-wide migration, and engineering managers who need a realistic strategy with measurable progress — not a rewrite. Participants leave with a concrete, incremental migration plan they can apply immediately.

Description

Migrating JavaScript to TypeScript is a strategy-and-execution course for teams that need to adopt TypeScript in an existing codebase without stopping feature work. It opens with the business case and the migration strategies that actually succeed, then moves into incremental adoption mechanics: enabling `allowJs` and `checkJs`, using JSDoc annotations as an on-ramp, and converting files in dependency order. Participants learn the strictness-ratcheting approach — start loose, then enable strict flags one at a time — so the migration delivers value early and never blocks the team.

The second day tackles the hard parts: typing legacy patterns such as dynamic objects, monkey patching, and CommonJS interop; managing third-party types with `@types` packages, declaration merging, and module augmentation; and migrating monorepos with project references. The course gives substantial attention to automation — codemods and conversion tooling, and AI-agent-assisted migration, where coding agents convert and verify modules at scale with the type checker in the loop as the reviewer that

keeps generated changes honest. (For deeper coverage of agentic workflows, see our AI-Assisted Development with JavaScript and TypeScript course.) Throughout, participants learn to measure progress objectively and keep CI green at every step.

Learning Outcomes

- Articulate the business case for TypeScript migration and choose a strategy suited to the codebase
- Configure `allowJs` and `checkJs` to type-check JavaScript before converting it
- Use JSDoc annotations as a low-friction on-ramp to typed code
- Plan file conversion order using dependency analysis
- Apply strictness ratcheting: migrate loose first, then enable strict flags one at a time
- Type legacy patterns including dynamic objects, monkey patching, and CommonJS/ESM interop
- Manage third-party types with `@types` packages, declaration merging, and module augmentation
- Migrate monorepos incrementally using project references
- Apply codemods and automated tooling to accelerate mechanical conversion
- Run AI-agent-assisted migration with type-checker-in-the-loop verification and human review
- Define and track migration metrics that show real progress
- Keep CI green throughout the migration without freezing feature development

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

- Node.js 24 LTS (or current Node.js 22+)
- Visual Studio Code or another TypeScript-aware editor
- Permission to install npm packages and editor extensions
- Git and a free GitHub account for lab repositories
- Access to an AI coding agent for the agent-assisted labs (a course-provided option is available)

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Training Topics

Migration Strategy and Business Case

- What TypeScript buys an existing codebase: defects, refactoring, onboarding
- Costs and risks: timeline, ecosystem, team skills
- Big bang versus incremental: why incremental wins
- Scoping: which code to migrate first and which to leave alone
- Setting expectations with stakeholders

Incremental Adoption Mechanics

- Adding TypeScript to a JavaScript build without breaking it
- `allowJs`: mixing `.js` and `.ts` files
- `checkJs`: type-checking JavaScript in place
- JSDoc annotations as an on-ramp: `@param`, `@returns`, `@type`, `@typedef`
- Converting files in dependency order
- Coexistence with existing bundlers and test runners

Strictness Ratcheting

- Starting loose: minimal compiler options that build
- Enabling strict flags one at a time: `noImplicitAny`, `strictNullChecks`, and beyond
- TypeScript 6 strict-by-default and what it means for new configs
- Ratchet tooling: preventing new violations while old ones burn down
- Suppression hygiene: `@ts-expect-error` over `@ts-ignore`

Typing Legacy Patterns

- Dynamic objects: index signatures, `Record`, and discriminated alternatives
- Monkey patching and global augmentation
- CommonJS interop: `esModuleInterop` and default import pitfalls
- Untyped event buses and string-keyed APIs
- `any` quarantine: containing the untypeable

Third-Party Types

- DefinitelyTyped and `@types` packages
- Libraries that ship their own types
- Writing local declaration files for untyped dependencies

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

- Declaration merging and module augmentation
- Pinning and upgrading type packages safely

Monorepo Migration with Project References

- Composite projects and build orchestration
- Migrating package by package
- Sharing types across packages during a partial migration
- Incremental builds and CI caching

Codemods and Automated Tooling

- What codemods do well: mechanical, repetitive transforms
- Conversion tooling: ts-migrate-style pipelines and jscodeshift
- Limits of automation and where humans must review

AI-Agent-Assisted Migration

- Using coding agents to convert and verify modules at scale
- Type-checker-in-the-loop: the compiler as the agent’s reviewer
- Prompting patterns: one module at a time, tests must pass
- Reviewing agent output: trust but verify
- Going deeper: our AI-Assisted Development with JavaScript and TypeScript course

Measuring Progress and Keeping CI Green

- Metrics: typed-file percentage, `any` count, strict-flag coverage
- Dashboards and ratchet reports
- CI gates that fail on regression, not on legacy debt
- Shipping features mid-migration without conflict pain