Cloud Contraptions
TRAINING & CONSULTING

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

Modernizing Legacy Angular Applications

Class Duration

21 hours of live training delivered over 3 days

Student Prerequisites

- Professional experience building and maintaining Angular (2+) applications
- Working knowledge of TypeScript, RxJS basics, and the Angular CLI
- This course covers Angular 2+ modernization only — AngularJS (1.x) rewrites are out of scope

Target Audience

Development teams, tech leads, and architects responsible for large, long-lived Angular codebases — typically written between Angular 8 and Angular 16 — who need to bring them to current Angular without stopping feature delivery. Ideal for organizations planning a structured modernization program across multiple applications and teams, and for engineering managers who need realistic estimates, sequencing, and risk management for the migration.

Description

Angular has changed more in the last four major versions than in the previous eight: Angular 21 (November 2025, in long-term support) made **zoneless change detection the default**, and Angular 22 (June 2026) shipped **stable Signal Forms**, stable `resource()`/`httpResource()` APIs, OnPush-by-default components, and **Vitest as the default test runner**. This course teaches teams how to carry a legacy Angular codebase across that gap deliberately and incrementally. Participants learn the supported upgrade path — `ng update`, one major version at a time, guided by the official update tool — and then work through each modernization layer with the schematics Angular provides: **NgModules to standalone components**, `*ngIf`/`*ngFor`/`*ngSwitch` to the **built-in control flow** (`@if`/`@for`/`@switch`) via `ng generate` migrations, **zone.js to zoneless** change detection, and constructor injection to `inject()`. The state-management section addresses the hardest judgment calls: migrating RxJS-heavy services

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

and component state to **signals** (`signal`, `computed`, `effect`, `toSignal`, and the now-stable resource APIs) while recognizing where RxJS remains the right tool. Forms get the same treatment — untyped to typed reactive forms, then a pragmatic adoption path for stable **Signal Forms**, including `ControlValueAccessor` interop. Testing migration covers **Karma/Jasmine to Vitest** using the official `migrate-karma-to-vitest` and `refactor-jasmine-vitest` schematics, plus replacing Protractor remnants with **Playwright**. Throughout, the course teaches incremental strategies for large codebases — and how to use AI coding agents, the Angular CLI MCP server, and the official Angular agent skills to accelerate mechanical migration work safely.

Learning Outcomes

- Assess a legacy Angular codebase and produce a sequenced, estimable modernization plan that does not halt feature delivery.
- Execute version upgrades from Angular 14-era codebases to Angular 22 with `ng update`, one major at a time, handling breaking changes and third-party library constraints.
- Migrate NgModule-based applications to standalone components using the official schematics, including bootstrap and routing changes.
- Convert structural directives to built-in control flow (`@if`, `@for`, `@switch`, `@defer`) with `ng generate` migrations, and clean up what the codemod cannot do.
- Move from zone.js to zoneless change detection: OnPush readiness, signal-driven updates, and the migration tooling.
- Refactor RxJS-heavy state to signals — `signal()`, `computed()`, `effect()`, `toSignal()`, and the stable `resource()`/`httpResource()` APIs — and articulate where RxJS remains the right choice.
- Migrate untyped reactive forms to typed forms, and adopt Signal Forms (stable in Angular 22) incrementally with `ControlValueAccessor` compatibility.
- Migrate test suites from Karma/Jasmine to Vitest (the Angular 22 default) with the official schematics, and replace Protractor-era end-to-end tests with Playwright.
- Apply incremental migration strategies — dual test setups, module-by-module conversion, migration budgets — across large codebases and multiple teams.
- Use AI coding agents with the Angular CLI MCP server and official Angular skills to accelerate migrations while keeping review rigor.

Cloud Contraptions
TRAINING & CONSULTING

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

Training Materials

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub in the form of documentation and extensive code samples. Students practice the topics covered through challenging hands-on lab exercises against a realistic legacy Angular codebase.

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Students will need permission to install Node.js and Visual Studio Code on their computers. Also, students will need permission to install NPM Packages and Visual Studio Code Extensions. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

Introduction

- The modern Angular baseline: what Angular 21 (zoneless default, LTS) and Angular 22 (stable Signal Forms, Vitest default, OnPush default) changed
- Anatomy of a legacy Angular codebase: NgModules, structural directives, zone.js, untyped forms, Karma
- Scope: Angular 2+ only — why AngularJS rewrites are a different project

Assessment and Planning

- Auditing the codebase: version, dependencies, deprecated API usage
- Third-party library risk: component libraries, state libraries, build tooling
- Sequencing the layers: upgrade first, then modernize
- Estimating and selling the plan: migration budgets alongside feature work

Version Upgrades with `ng update`

- The supported path: one major version at a time
- Using the official Angular update guide to generate a checklist
- `ng update` schematics: what they fix automatically
- Breaking-change hotspots from v14 to v22 (TypeScript and Node requirements, removed APIs, router changes)

- Verifying each hop: build, test, smoke-test before the next major

NgModules to Standalone

- Why standalone components are the modern default
- The official standalone migration schematic
- Migrating bootstrap: `bootstrapApplication` and `app.config.ts`
- Routing without modules: `provideRouter`, lazy `loadComponent`
- Living with a hybrid codebase mid-migration

Control Flow Migration

- `*ngIf` / `*ngFor` / `*ngSwitch` to `@if` / `@for` / `@switch`
- The `ng generate` control-flow migration and its limits
- `track` expressions replacing `trackBy`
- `@defer` for lazy template blocks
- `strictTemplates` (default in v22) and fixing what it surfaces

Zone.js to Zoneless

- How zoneless change detection works: signals drive updates
- OnPush readiness as the stepping stone (OnPush is the v22 default)
- Finding zone-dependent code: timers, third-party events, manual `detectChanges`
- The zoneless migration tooling and incremental rollout
- DI modernization along the way: constructor injection to `inject()`

RxJS to Signals

- What moves to signals: component state, derived state, template bindings
- `signal()`, `computed()`, `effect()`, `linkedSignal()`
- Interop: `toSignal()` and `toObservable()`
- Replacing fetch-and-subscribe services with `resource()` and `httpResource()` (stable in v22)
- Where RxJS remains right: event streams, websockets, complex operator pipelines
- Killing the `async` pipe + `BehaviorSubject` state pattern safely

Forms Modernization

- Untyped to typed reactive forms: migration mechanics and payoff
- Typed forms as the foundation for everything after
- Signal Forms (stable in Angular 22): the model, `form()`, validators

- Incremental adoption: `ControlValueAccessor` compatibility and `FormValueControl`
- Choosing per-form: leave as typed reactive, or move to Signal Forms

Testing Migration

- Why Karma is over: Vitest as the Angular default test runner
- The `migrate-karma-to-vitest` schematic
- Converting Jasmine specs with `refactor-jasmine-vitest` (including `--fake-async`)
- Running Karma and Vitest side by side during migration
- Zone.js test support in Vitest (`zone.js/plugins/vitest-patch`) as a bridge
- Vitest Browser Mode for component tests
- Protractor remnants: porting end-to-end suites to Playwright

Incremental Strategies at Scale

- Strangler-style migration: boundaries, seams, and feature flags
- Module-by-module and team-by-team rollout
- Keeping `main` releasable throughout
- Metrics: tracking migration progress honestly
- Coordinating shared component libraries across applications

AI-Accelerated Migration

- Where coding agents excel: mechanical, pattern-based rewrites at volume
- The Angular CLI MCP server and the official Angular agent skills
- Prompting agents with migration recipes; constraining the blast radius
- Review discipline: schematics first, agents second, humans always
- Pitfalls: hallucinated APIs, stale training data vs. Angular 22 idioms