Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Domain-Driven Design and Microservices

Class Duration

21 hours of live training delivered over 3 days.

Student Prerequisites

- Professional software development experience
- Comfort designing classes and interfaces in C#, Java, Python, or TypeScript
- Familiarity with relational databases and HTTP APIs
- Design patterns experience is helpful but not required
- No prior domain-driven design experience required

Target Audience

Developers, tech leads, and architects who build business software with complex rules, or who are splitting a monolith into services. Code examples use C#, Java, Python, or TypeScript to match your team. Teams that want object-oriented design first can take Design Patterns and SOLID in C#, Python, or TypeScript. For the contracts between services, add API Design: REST, GraphQL, and gRPC.

Description

This course teaches domain-driven design for teams building software with complex business rules. Day one covers strategic design: subdomains, ubiquitous language, bounded contexts, and context maps. Students explore a sample business with EventStorming and turn what they find into code. Day two covers the tactical patterns, including entities, value objects, aggregates, domain events, and repositories. Day three applies DDD to architecture: modular monoliths, microservice boundaries, CQRS, event sourcing, sagas, and legacy systems.

Learning Outcomes

- Explain when domain-driven design is worth the effort and when a simpler design is better.
- Classify subdomains as core, supporting, or generic.

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

- Build a ubiquitous language with domain experts and keep it in the code.
- Find bounded contexts and draw a context map.
- Explore a domain with EventStorming and domain storytelling.
- Model entities and value objects that enforce business rules.
- Design aggregates around invariants and transaction boundaries.
- Publish and handle domain events.
- Keep the domain model independent of frameworks with hexagonal architecture.
- Choose between a modular monolith and microservices for each context.
- Apply CQRS, event sourcing, sagas, and the transactional outbox where they fit.
- Protect a new model from legacy systems with an anti-corruption layer.
- Split a monolith along bounded contexts with the strangler fig pattern.

Training Materials

Comprehensive courseware is distributed online at the start of class.

Software Requirements

Students will need a free, personal GitHub account to access the courseware, an IDE for their team’s language (Visual Studio, JetBrains Rider, IntelliJ IDEA, PyCharm, or VS Code), the matching SDK (.NET 10+, Java 21+, Python 3.12+, or Node.js 22+), and Docker Desktop or Podman to run a database and message broker locally. EventStorming uses a shared online whiteboard such as Miro. If students are unable to configure a local environment, a cloud-based environment can be provided.

Training Topics

Why Domain-Driven Design

- Complexity in business software
- The domain model as code
- When DDD fits
- When simpler designs win
- Strategic versus tactical design

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Subdomains and Ubiquitous Language

- Core, supporting, and generic subdomains
- Where to spend design effort
- Building a ubiquitous language
- Working with domain experts
- Keeping the language in code

Bounded Contexts

- What a bounded context is
- Finding context boundaries
- One model per context
- Contexts and team ownership
- Conway’s law in practice

Context Mapping

- Upstream and downstream contexts
- Partnership and shared kernel
- Customer-supplier and conformist
- Anti-corruption layers
- Open host service
- Published language
- Drawing a context map

Exploring the Domain

- Big Picture EventStorming
- Process-level EventStorming
- Design-level EventStorming
- Domain storytelling
- Capturing business rules
- From sticky notes to code

Entities and Value Objects

- Identity and life cycle
- Immutable value objects
- Replacing primitives with value objects
- Validation inside the model
- Value equality

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Aggregates

- Invariants and consistency boundaries
- Choosing the aggregate root
- Keeping aggregates small
- Referencing aggregates by ID
- One aggregate per transaction
- Optimistic concurrency

Domain Events

- Events in the ubiquitous language
- Raising events from aggregates
- Domain versus integration events
- Handling events in process
- Eventual consistency between aggregates

Repositories and Services

- One repository per aggregate
- Persisting aggregates with an ORM
- Domain services
- Application services and use cases
- Factories for complex creation

Architecture Around the Domain

- Hexagonal architecture
- Ports and adapters
- Clean and onion architecture
- Keeping frameworks out of the domain
- Vertical slices
- Testing the domain model

Modular Monoliths and Microservices

- Modules aligned to bounded contexts
- When to split into services
- Data ownership per service
- Synchronous versus asynchronous calls
- Contracts between contexts
- Avoiding the distributed monolith

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

CQRS and Event Sourcing

- Separating reads and writes
- Read models and projections
- Event-sourced aggregates
- Event store design
- Versioning stored events
- When event sourcing is too much

Long-Running Processes

- Sagas and process managers
- Orchestration versus choreography
- The transactional outbox
- Idempotent message handlers
- Compensating actions

Legacy Systems and Evolution

- Wrapping legacy systems
- The strangler fig pattern
- Finding seams in a monolith
- Refactoring toward a richer model
- Revisiting context boundaries
- Domain language for AI coding assistants

Copyright 2026 Cloud Contraptions LLC – All Rights Reserved