Case Management Domain
Directory Structure Guide

Datadog Case Management Team

January 2026

Abstract

This document provides a comprehensive guide to the Case Management domain directory structure in the dd-source monorepo. It explains the organization of applications, libraries, modules, and protocol definitions, along with architectural patterns and design decisions that shape the codebase.

Contents

1 Executive Summary

The Case Management domain is a sophisticated, event-driven microservices architecture located at:

`domains/case_management/`

The domain is organized into six primary directories, each serving a distinct architectural purpose:

- **apps/** - 12 standalone microservices and CLI applications
- **libs/** - 46 reusable libraries providing cross-cutting functionality
- **modules/** - 22 feature modules implementing core business logic
- **proto/** - 28 Protocol Buffer definitions for gRPC services
- **config/** - Kubernetes and deployment configuration
- **shared/** - Cross-domain shared libraries

The architecture follows a clean separation of concerns with synchronous gRPC APIs for interactive operations, asynchronous event processing via Kafka for background tasks, and pluggable event handlers for external integrations. The system supports multi-tenancy (case-management and on-call), uses PostgreSQL for primary storage, Elasticsearch for search, and Kafka for event streaming.

2 Top-Level Directory Structure

2.1 Directory Overview

```
domains/case_management/
          apps/                    # Microservices & applications
                apis/
                      case-rapid-api/  # HTTP REST API wrapper
                case-api/            # Core gRPC server
                case-intake/         # Case creation entry point
                case-event-relay/    # Event relay to Kafka
                case-event-handler/  # Event consumer (12 flavors)
                case-indexer/        # Elasticsearch indexing
                case-third-party-reader/  # External integrations
                case-cli/            # Command-line interface
                case-chat-interactions/   # Chat integrations
                case-synthetics/     # Synthetic testing
                caseworker/          # Work platform integration
          libs/                    # Reusable libraries (46 total)
                auth/                # Authorization & RBAC
                kafka/               # Kafka abstractions
                pg/                  # PostgreSQL client
                jira/                # Jira integration
                ...                  # (42 more libraries)
          modules/                 # Feature modules (22 total)
                casem/               # Core case aggregate
                analytic/            # Analytics & reporting
                timeline/            # Activity tracking
                notification/        # Notifications
                ...                  # (18 more modules)
          proto/                      # Protocol Buffer definitions
                analyticpb/             # Analytics service
                timelinepb/             # Timeline service
                ...                     # (26 more proto packages)
          config/                     # Deployment configuration
                k8s/                    # Kubernetes/Helm charts
                fabric-remotes/         # Datacenter configs
                cnab/                   # Cloud Native bundles
          shared/                     # Cross-domain libraries
              libs/
                  go/
                      case_api/       # Shared gRPC client
                      file_api/       # File service client
```

2.2 Architectural Philosophy

The directory structure reflects key architectural principles:

1. **Separation of Concerns** - Apps, libs, and modules have distinct responsibilities
2. **Reusability** - Common functionality extracted into shared libraries
3. **Domain-Driven Design** - Modules organized around business capabilities
4. **Service-Oriented** - Each app is independently deployable
5. **Contract-First** - Proto definitions drive service interfaces

3 Apps Directory - Microservices & Applications

The `apps/` directory contains 12 standalone applications, each serving a specific purpose in the case management ecosystem.

3.1 Core Services

3.1.1 case-api

Purpose: Central gRPC server exposing all case management operations.
Characteristics:

- Registers 15+ gRPC services (Cases, Timeline, Attachments, Analytics, Projects, etc.)
- Handles database migrations via golang-migrate
- Manages Elasticsearch index mappings
- Provides health checks on port 8080
- Supports multi-tenancy (case-management and on-call)

Key Files:

- `main.go` - Service initialization and module registration
- `etc/migrations/` - PostgreSQL schema migrations
- `config/k8s/` - Kubernetes deployment manifests

3.1.2 case-intake

Purpose: Entry point for case creation requests from external systems.
Characteristics:

- gRPC service accepting case intake requests
- Produces Kafka events for asynchronous processing
- Enforces rate limiting and validation
- Supports batch case creation

3.1.3 case-event-relay

Purpose: Implements transactional outbox pattern to relay database events to Kafka.
Characteristics:

- Reads domain events from PostgreSQL `domain_event` table
- Publishes events to Kafka topics (32 partitions)
- Active-passive deployment with distributed locking
- Handles 60+ event types (CaseCreated, StatusChanged, etc.)
- Automatic failover in 20-30 seconds

Components:

- LockController - Distributed lock management
- Reader - Database event queries
- Producer - Kafka message encoding
- Acknowledger - Event deletion after delivery
- MetricsGenerator - Backlog monitoring

3.1.4 case-event-handler

Purpose: Multi-flavor Kafka consumer processing domain events.
Supported Flavors (12):

1. **notification** - Sends notifications via Nexus/NP
2. **timeline** - Updates case activity timeline
3. **search-indexer** - Indexes cases in Elasticsearch
4. **jira** - Syncs cases with Jira issues
5. **snow-sync** - Syncs cases with ServiceNow tickets
6. **evp-indexer** - Indexes into Event Management platform
7. **paging** - Triggers PagerDuty alerts
8. **rule-evaluator** - Evaluates automation rules

9. **incident-escalation** - Escalates cases to incidents
10. **analytic** - Updates analytics aggregates
11. **maintenance-window** - Handles maintenance windows
12. **attachment** - Processes file attachments

Architecture:

- Each flavor runs as separate deployment
- Consumes from domain-events Kafka topic
- Supports replay and backfill operations
- Independent scaling per flavor

3.2 API & Integration Services

3.2.1 case-rapid-api

Purpose: HTTP REST API wrapper around gRPC case-api.
Characteristics:

- Auto-generated via Rapid platform
- JSON-API compatible HTTP endpoints
- Converts between JSON-API and gRPC protobuf
- Exposes analytics, cases, projects, synchronization APIs
- OpenAPI documentation generation

3.2.2 case-indexer

Purpose: Elasticsearch indexing service for case search.
Characteristics:

- Manages Elasticsearch index lifecycle
- Provides search query builders
- Supports filtering, sorting, and aggregations
- Index mapping management

3.2.3 case-third-party-reader

Purpose: Reads data from external ITSM systems.
Integrations:

- ServiceNow - Reads ticket data
- Jira - Reads issue data
- Bi-directional synchronization support
- Webhook receivers for external updates

3.2.4 case-cli

Purpose: Command-line interface for case operations.
Use Cases:

- Manual testing of notification flows
- Triggering case operations
- Debugging and troubleshooting
- Administrative tasks

3.3 Supporting Services

3.3.1 case-chat-interactions

Purpose: Handles chat platform integrations.
Platforms:

- Slack - Channel notifications and interactions
- Other chat platforms (extensible)

3.3.2 case-synthetics

Purpose: Synthetic testing for case operations.
Testing:

- End-to-end case lifecycle tests
- Performance monitoring
- Availability checks

3.3.3 caseworker

Purpose: Work platform integration service.
Integration:

- Work Platform tenant management
- Organization and team sync

3.4 Service Deployment Pattern

All apps follow standard Datadog service patterns:

- **Runtime:** Go 1.21+
- **Build:** Bazel build system
- **Deployment:** Kubernetes with Helm charts
- **Health:** Liveness/readiness probes on port 8080
- **Observability:** dd-trace (APM), StatsD (metrics), structured logging
- **Configuration:** Environment variables + INI files

4 Libs Directory - Reusable Libraries

The `libs/` directory contains 46 libraries organized by functional concern.

4.1 Infrastructure & Persistence

Library	Purpose
`pg`	PostgreSQL client, connection pooling, query builders, transaction management
`kafka`	Kafka producer/consumer abstractions, topic management, offset handling
`es`	Elasticsearch client, query DSL, index management
`grpc`	gRPC service setup, interceptors, middleware, error handling

4.2 Authentication & Authorization

Library	Purpose
`auth`	Granular RBAC authorization client (Ticino/Zoltron integration), permission checks
`obo`	On-behalf-of authentication for service-to-service calls, impersonation
`user`	User service integration with AAA domain, user lookup and caching
`internaluser`	Internal system user management for automated operations

4.3 External Integrations

Library	Purpose
`jira`	Jira API client, issue CRUD, webhook handling, field mapping
`servicenow`	ServiceNow REST API client, ticket sync, change management
`ddclient`	Datadog API clients (notebooks, incidents, monitors, dashboards)
`workplatform`	Work Platform tenant/org management, team sync
`nexus`	Cross-domain notification distribution, event bus
`eventcorrelation`	Event correlation with Incident Management domain
`np`	Notification Processor client for notification delivery

4.4 Domain Models & Utilities

Library	Purpose
`model`	Core domain models (OrgID, case metadata, timestamps)
`fields`	Case field definitions, custom attributes, field types
`validator`	Input validation for API requests, constraint checking
`protobuf`	Protocol buffer utilities, encoding/decoding helpers
`parser`	Query/filter expression parser (inspired by DD filter syntax)

Library	Purpose
`cerr`	Case management error types, error codes, wrapping
`codec`	Generic encoding/decoding utilities
`json`	JSON serialization/deserialization, custom marshalers
`jsonapimodel`	JSON-API specification compatibility layer

4.5 Feature Libraries

Library	Purpose
`notification`	Notification templating, delivery tracking, retry logic
`evaluator`	Rule evaluation engine, expression parsing, condition matching
`monitor`	Monitor integration, alert correlation
`trigger`	Automation trigger framework, event-condition-action
`experiments`	Feature flags, A/B testing, gradual rollouts
`casetype`	Case type definitions, validation rules, field schemas
`customattribute`	Custom attribute storage, indexing, querying

4.6 Observability & Utilities

Library	Purpose
`utils`	Configuration loading, logging setup, common utilities
`obsutil`	Observability utilities (structured logging, metric helpers)
`ddd`	Domain-driven design patterns (aggregates, repositories)
`org`	Organization management, org metadata
`team`	Team and group management, membership
`typeguard`	Type assertion helpers, safe casting
`retrier`	Retry logic with exponential backoff
`slices`	Slice manipulation utilities
`atlas`	Atlas worker integration for data processing pipelines
`forked`	Forked dependencies (config parsers, migration tools)

4.7 Library Design Principles

1. **Single Responsibility** - Each library has one clear purpose
2. **Dependency Injection** - Libraries accept interfaces, not concrete types
3. **Testability** - All libraries include comprehensive unit tests
4. **Documentation** - Godoc comments for all public APIs
5. **Versioning** - Internal versioning via Go modules

5 Modules Directory - Feature Modules

The `modules/` directory contains 22 modules implementing core business features. Each module typically includes:

- `server.go` - gRPC service implementation

- `service.go` - Business logic layer
- `repository.go` - Data access layer
- Domain models and value objects

5.1 Core Case Management Modules

Module	Purpose & Responsibilities
`casem`	Core case aggregate implementing the main Cases gRPC service. Handles case CRUD, queries, state management, lifecycle events.
`casetype`	Case type definitions, validation schemas, field requirements, workflow configurations.
`status`	Case status enumerations (Open, In Progress, Resolved, Closed), status transitions, validation rules.
`project`	Case organization into projects, project metadata, case-to-project associations.
`timeline`	Case activity timeline and audit log. Tracks all changes to a case with timestamps, authors, and change details.

5.2 Feature Enhancement Modules

Module	Purpose & Responsibilities
`link`	Case relationships and linking to external entities (incidents, monitors, dashboards, notebooks). Bi-directional link management.
`attachments`	File attachment storage, retrieval, and metadata management. Supports multiple file types, size limits, virus scanning.
`watcher`	Users watching/subscribing to case updates. Manages notification preferences, subscription lifecycle.
`notification`	Notification generation and sending based on case events. Template management, delivery tracking.
`notificationrule`	Automated notification rules triggered by case events. Rule engine, condition evaluation.
`automation`	Automation rules for case workflows. If-then-else logic, action execution, rule management.

5.3 Integration Modules

Module	Purpose & Responsibilities
`synchronisation`	Two-way synchronization with external ITSM systems (Jira, ServiceNow). Mapping management, conflict resolution.
`slack`	Slack integration for notifications, case creation from Slack, interactive components.
`trigger`	Workflow triggers for automation. Event matching, trigger evaluation, action dispatch.

Module	Purpose & Responsibilities
`notificationplatform`	Bridge between case notifications and external notification platforms.

5.4 Data & Analytics Modules

Module	Purpose & Responsibilities
`analytic`	Case analytics and reporting. Timeseries queries, scalar metrics, group-by dimensions (status, priority, assignee, etc.). Support for multiple group-by dimensions.
`aggregate`	Event aggregation for analytics. Pre-computed aggregates, incremental updates.
`view`	Saved case list views and filters. User-defined queries, sharing, default views.
`customattribute`	Custom field definitions and values per case. Dynamic schema, indexing, querying.

5.5 Configuration & Metadata Modules

Module	Purpose & Responsibilities
`settings`	Cached settings and configuration. User preferences, org-level settings, feature flags.
`projectfavorite`	User's favorite projects. Quick access, ordering.
`maintenancewindow`	Maintenance window scheduling. Notification suppression during maintenance.
`domainevent`	Domain event storage, lifecycle management, event versioning. Implements transactional outbox pattern.

5.6 Module Registration Pattern

All modules register their gRPC services with the `case-api` server:

```
// case-api/main.go
func main() {
    grpcServer := grpc.NewServer()

    // Register module services
    casempb.RegisterCasesServer(grpcServer,
        casem.NewServer(db, kafka))
    analyticpb.RegisterAnalyticsServer(grpcServer,
        analytic.NewServer(db))
    timelinepb.RegisterTimelineServer(grpcServer,
        timeline.NewServer(db))
    // ... 12+ more services

    grpcServer.Serve(listener)
}
```

6 Proto Directory - Protocol Buffer Definitions

The `proto/` directory contains 28 Protocol Buffer packages defining gRPC service interfaces.

6.1 Proto Organization

Each proto package follows a consistent structure:

```
proto/analyticpb/
          analytic.proto                # Service definition
          analytic.pb.go                # Generated Go code
          analytic_pb2.py               # Generated Python code
          analytic_grpc.pb.go           # Generated gRPC server/client
          analytic.pb.validate.go       # Generated validation
```

6.2 Main Service Definitions

Proto Package	Service & Methods
`proto/` (Cases)	CreateCase, GetCase, UpdateCase, DeleteCase, QueryCases, ArchiveCase, UnarchiveCase
`analyticpb`	GetTimeseries (with group-by), GetScalar, GetSuggestions, GetGroupByValues
`timelinepb`	GetTimeline, AddTimelineEntry, UpdateEntry, DeleteEntry
`attachmentpb`	UploadAttachment, GetAttachment, DeleteAttachment, ListAttachments
`projectpb`	CreateProject, GetProject, UpdateProject, DeleteProject, ListProjects
`statuspb`	GetStatuses, GetStatusTransitions, UpdateStatus
`casetypepb`	GetCaseTypes, CreateCaseType, UpdateCaseType
`linkpb`	CreateLink, GetLinks, DeleteLink
`watcherpb`	AddWatcher, RemoveWatcher, GetWatchers
`viewpb`	SaveView, GetView, DeleteView, ListViews
`customattributepb`	DefineAttribute, GetAttributes, SetAttributeValue
`notificationrulepb`	CreateRule, UpdateRule, DeleteRule, EvaluateRule
`automationpb`	CreateAutomation, UpdateAutomation, ExecuteAutomation
`synchronisationpb`	CreateSync, GetSyncStatus, UpdateMapping
`domainpb`	Internal domain event definitions (not exposed externally)
`intakepb`	IntakeCase, BatchIntake

6.3 Proto Design Patterns

6.3.1 Request/Response Pattern

All RPC methods follow consistent naming:

```
service Analytics {
    rpc GetTimeseries(GetTimeseriesRequest)
        returns (GetTimeseriesResponse);
    rpc GetScalar(GetScalarRequest)
        returns (GetScalarResponse);
}
```

6.3.2 Common Message Types

Shared types across services:

- `Timestamp` - RFC3339 timestamps
- `OrgID` - Organization identifiers
- `UserID` - User identifiers
- `CaseID` - Case public identifiers
- `Pagination` - Cursor-based pagination

6.3.3 Validation Rules

Proto files include validation annotations:

```
message CreateCaseRequest {
    string title = 1 [(validate.rules).string = {
        min_len: 1,
        max_len: 256
    }];
    string priority = 2 [(validate.rules).string = {
        in: ["P1", "P2", "P3", "P4", "P5"]
    }];
}
```

7 Architecture Patterns & Event Flow

7.1 Event-Driven Architecture

The case management domain follows an event-driven architecture with clear separation between synchronous and asynchronous operations.

7.1.1 Complete Event Flow

```
  External Apps   (Web UI, API clients, Integrations)

          HTTP/gRPC


     case-rapid-api   (HTTP → gRPC translation)

              gRPC


       case-intake    (Validation, rate limiting)

              gRPC


        case-api      (Core business logic)
      (15+ services)
```

```
              SQL + Domain Events


      PostgreSQL Database

       cases table
       domain_event table






    case-event-relay  (Transactional outbox)
      (Distributed
       lock owner)

              Kafka Produce


       Kafka
       Topic: domain-events
       Partitions: 32





        ...
     Handler    Handler    Handler    Handler
    notificationtimeline  search-idx  jira





    Nexus/NP  Timeline    ES        Jira
                Table     Index     API
```

7.2 Transactional Outbox Pattern

The domain implements the transactional outbox pattern to ensure reliable event delivery:

7.2.1 Pattern Benefits

1. **Atomicity** - Events written in same transaction as business data
2. **Reliability** - No event loss on service failures
3. **Ordering** - Events processed in write order
4. **At-Least-Once** - Events delivered at least once to consumers
5. **No Dual Writes** - Avoids distributed transaction complexity

7.2.2 Implementation Steps

1. **Write Phase**
   - case-api writes business data to `cases` table
   - case-api writes domain event to `domain_event` table
   - Both operations in single PostgreSQL transaction
   - Transaction commits atomically
2. **Relay Phase**
   - case-event-relay acquires distributed lock
   - Reads unread events (`read_on IS NULL`)
   - Encodes events as protobuf with version prefix
   - Publishes to Kafka (async with ack callback)
3. **Acknowledgment Phase**
   - Kafka confirms delivery to relay
   - Relay batch-deletes acknowledged events
   - Metrics published for end-to-end latency
4. **Consumption Phase**
   - case-event-handler flavors consume from Kafka
   - Each flavor processes events independently
   - Consumer commits offset after successful processing

7.3 Multi-Tenancy Architecture

The domain supports two tenants with separate Kafka topics:

Tenant	Kafka Topic	Use Case
case-management	domain-events	Case Management product domain events
on-call	domain-events-oncall	On-Call product domain events

7.4 Distributed Lock Pattern

The case-event-relay uses distributed locking to implement active-passive deployment:

- **Lock Storage:** PostgreSQL `event_relay_lock` table
- **Heartbeat:** 10-second interval
- **Timeout:** 20 seconds (2x heartbeat period)
- **Failover:** Automatic within 20-30 seconds
- **Split-Brain Prevention:** Grace period prevents multiple owners

8 Build System & Development Workflow

8.1 Bazel Build System

The entire case management domain uses Bazel for building, testing, and dependency management.

8.1.1 Key Commands

```
# Always use 'bzl' wrapper (enforced by hooks)
bzl build //domains/case_management/apps/case-api:case-api
bzl test //domains/case_management/modules/analytic/...
bzl run //:gazelle -- domains/case_management/
```

8.1.2 Target Conventions

Gazelle generates Bazel targets by convention:

- Library target: `//foo/bar/baz:go_default_library`
- Test target: `//foo/bar/baz:go_default_test`
- Integration test: `//foo/bar/baz:go_default_integration_test`

8.1.3 Test Logs

Failed test logs are located at:
`bazel-testlogs/<path>/<target>/test.log`
Example: `bazel-testlogs/domains/case_management/modules/analytic/analytic_test/test.log`

8.2 Development Workflow

8.2.1 Local Development

1. Run Tilt for local Kubernetes development:
   ```
   tilt up
   ```
2. Build and test locally:
   ```
   bzl build //domains/case_management/apps/case-api:case-api
   bzl test //domains/case_management/modules/analytic/...
   ```
3. Update BUILD files after adding dependencies:
   ```
   bzl run //:gazelle -- domains/case_management/
   ```

8.2.2 Deployment Workflow

1. Run tests: `bzl test //domains/case_management/...`
2. Commit changes: `git commit -m "feat: description"`
3. Push branch: `git push origin feature-branch`
4. Create PR: `gh pr create`
5. Deploy to staging: `integrate case-api -st -sg`

8.3 Testing Strategy

8.3.1 Test Types

- **Unit Tests** - Test individual functions and methods
- **Integration Tests** - Test database and Kafka interactions
- **End-to-End Tests** - Test complete workflows via case-synthetics

8.3.2 Common Test Patterns

1. Table-driven tests for multiple scenarios
2. Test fixtures in `testdata/` directories
3. Mock clients for external dependencies
4. Testcontainers for PostgreSQL and Kafka

9 Configuration & Deployment

9.1 Kubernetes Deployment

All services deploy to Kubernetes using Helm charts in `config/k8s/`.

9.1.1 Multi-Tenant Configuration

```
config/k8s/values/tenants/
          case-management/
                case-api.yaml
                case-intake.yaml
                case-event-relay.yaml
                case-event-handler.yaml
          on-call/
              case-api.yaml
              case-event-handler.yaml
```

9.1.2 Datacenter Configuration

```
config/fabric-remotes/values/datacenter/
          us1.yaml
          eu1.yaml
          ap1.yaml
          gov1.yaml  (FIPS-compliant)
```

9.2 Environment Variables

Common environment variables across services:

- `DD_ENV` - Environment (staging, prod, gov)
- `DD_SERVICE` - Service name
- `DD_VERSION` - Service version

- `DATABASE_URL` - PostgreSQL connection string
- `KAFKA_BROKERS` - Kafka broker addresses
- `ELASTICSEARCH_URL` - Elasticsearch endpoint

10 Observability & Monitoring

10.1 Metrics

All services emit metrics to Datadog:

- Request rates and latencies (via dd-trace)
- Error rates and types
- Database query performance
- Kafka consumer lag
- Event processing latency

10.2 Distributed Tracing

Full distributed tracing from HTTP request through event consumption:

1. HTTP request arrives at case-rapid-api (trace created)
2. gRPC call to case-api (span created)
3. Database write (span created)
4. Domain event written (span created)
5. Event relayed to Kafka (span created, trace context in headers)
6. Event consumed by handler (span created, parent trace ID preserved)
7. External API call (span created)

10.3 Dashboards

Key dashboards for case management domain:

- Service health and golden signals
- Event relay backlog and latency
- Consumer lag by flavor
- Database performance
- Integration health (Jira, ServiceNow, Elasticsearch)

11 Current Development Work

Based on git status, active development is on branch:
`agustin.fusaro/CASEM-2958-support-multiple-group-bys-backup`

11.1 Modified Files

1. `apps/apis/case-rapid-api/http/handler_analytic.go`
    - HTTP handler for analytics endpoints
    - Adding support for multiple group-by dimensions
2. `modules/analytic/server.go`
    - gRPC service implementation
    - Core analytics query logic
3. `modules/analytic/groupby/suggestions.go`
    - Group-by dimension suggestions
    - Value autocomplete for group-by fields
4. `modules/analytic/BUILD.bazel`
5. `modules/analytic/groupby/BUILD.bazel`
    - Bazel build configuration updates

11.2 Feature: Multiple Group-By Support

The feature (CASEM-2958) adds support for multiple group-by dimensions in analytics queries:

Before:

```
GET /analytics?group_by=status
Returns: [{status: "open", count: 10}, ...]
```

After:

```
GET /analytics?group_by=status,priority
Returns: [
  {status: "open", priority: "P1", count: 5},
  {status: "open", priority: "P2", count: 3},
  ...
]
```

12 Best Practices & Guidelines

12.1 Code Organization

1. **Apps** - Only for standalone deployable services
2. **Libs** - For code shared across multiple apps/modules
3. **Modules** - For feature-specific business logic with gRPC interface
4. **Proto** - Contract-first development, version all breaking changes

12.2 Dependency Rules

- Apps may depend on libs and modules
- Modules may depend on libs and other modules
- Libs may depend on other libs (avoid circular dependencies)
- Proto packages should have no dependencies

12.3 Testing Guidelines

1. Write unit tests for all business logic
2. Use integration tests for database and Kafka interactions
3. Mock external dependencies (Jira, ServiceNow, Elasticsearch)
4. Run tests before committing: `bzl test //path/...`

12.4 Error Handling

- Use `cerr` package for domain-specific errors
- Include error codes for client handling
- Log errors with context (trace ID, user ID, case ID)
- Return gRPC status codes appropriately

13 Conclusion

The Case Management domain is a well-architected, event-driven system with clear separation of concerns. The directory structure reflects intentional design decisions:

- **apps/** - Independently deployable microservices
- **libs/** - Reusable cross-cutting functionality
- **modules/** - Feature-oriented business logic
- **proto/** - Contract-first gRPC interfaces
- **config/** - Multi-tenant, multi-datacenter deployment

The architecture prioritizes:

1. **Reliability** - Transactional outbox pattern prevents event loss
2. **Scalability** - Event-driven design enables horizontal scaling
3. **Maintainability** - Clear boundaries between components
4. **Observability** - Comprehensive metrics, tracing, and logging
5. **Extensibility** - Pluggable event handlers and integrations

This structure has enabled the Case Management team to rapidly develop new features, integrate with external systems, and scale to handle production workloads across multiple datacenters worldwide.

13.1 Key Takeaways

1. The domain follows microservices architecture with 12 apps
2. 46 libraries provide reusable functionality
3. 22 modules implement business features
4. 28 proto packages define gRPC contracts
5. Event-driven design with Kafka enables asynchronous processing
6. Transactional outbox pattern ensures reliable event delivery
7. Multi-tenant support for case-management and on-call products

This document reflects the case management domain structure as of January 2026. For the latest details, refer to the source code at:
`domains/case_management/`