Software Architecture Documentation

Comprehensive Template and Guide

Views and Beyond Approach for OrderFlow Commerce Platform

Based on SEI/CMU Documentation Standards

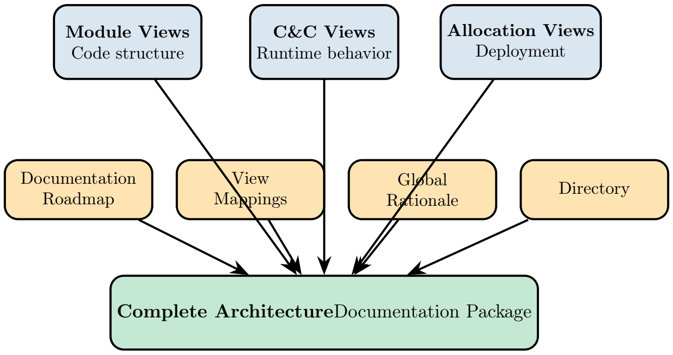
E-Commerce Platform Modernization

Architecture Documentation Series

December 9, 2025

Abstract

This document provides a comprehensive architecture documentation template following the Views and Beyond approach developed by the Software Engineering Institute at Carnegie Mellon University. The template establishes a systematic framework for documenting software architectures through multiple architectural views, each addressing specific stakeholder concerns. This guide serves dual purposes: as a reference template for creating architecture documentation and as a filled-in example demonstrating best practices for a modern e-commerce platform. The document covers the complete documentation structure including view templates, element catalogs, rationale documentation, requirements traceability, and governance processes essential for maintaining architecture documentation throughout the system lifecycle.



Contents

Document Control Information

Attribute	Value
Document ID	ARCH-DOC-001
Title	Architecture Description for OrderFlow Commerce Platform
Project	E-Commerce Platform Modernization
Version	2.1.0
Status	Approved **[APPROVED]**
Author / Owner	Architecture Team
Approver	Chief Architect
Creation Date	2023-06-15
Last Updated	December 9, 2025
Repository	`github.com/company/orderflow-architecture`
Branch/Tag	`main` / `v2.1.0`
Classification	Internal – Confidential
Review Cycle	Quarterly

Change History

Version	Date	Author	Summary of Changes
1.0.0	2023-06-15	J. Smith	Initial architecture documentation release
1.1.0	2023-09-20	A. Jones	Added Payment Service decomposition; updated C&C view
1.2.0	2023-12-10	J. Smith	Added Kubernetes deployment view; security patterns
2.0.0	2024-03-15	Architecture Team	Major revision: microservices decomposition; event-driven patterns
2.1.0	2024-06-01	B. Wilson	Added GraphQL gateway; updated requirements traceability

Change Request and Review Process

Architecture documentation changes follow a structured governance process to ensure quality and stakeholder alignment.

1. **Change Request:** Submit via GitHub Issue using the "Architecture Change" template. Include rationale, affected views, and stakeholder impact assessment.
2. **Triage:** Architecture Team reviews within 2 business days, assigns priority (Critical/High-/Medium/Low), and identifies reviewers.
3. **Development:** Author creates branch (`arch/description-of-change`), updates documentation, and ensures all cross-references remain valid.

4. **Review:** Pull request requires approval from:
   - At least one Architecture Team member
   - Affected subsystem technical lead
   - Chief Architect for structural changes
5. **Release:** Merged changes trigger documentation build. Major versions announced via architecture-announce mailing list.

Key Point

All architecture documentation changes must be traceable to either a requirements change, an Architecture Decision Record (ADR), or a documented technical debt item.

1 Documentation Roadmap

The Documentation Roadmap provides readers with an orientation to the architecture documentation, explaining its purpose, organization, and how different stakeholders should navigate it.

1.1 Scope and Summary

> **Definition**
>
> **Architecture Documentation** captures the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution (IEEE 1471/ISO 42010).

- **Purpose:** This document describes the software architecture of OrderFlow Commerce Platform, a modern e-commerce platform supporting B2C and B2B transactions. It serves as the authoritative reference for architectural decisions, component responsibilities, and system structure to enable effective communication among stakeholders, guide implementation, support analysis, and facilitate evolution.
- **Scope:** The documentation covers:
  - Core platform services (Order, Inventory, Payment, User, Catalog, Search)
  - Supporting infrastructure (API Gateway, Event Bus, Observability)
  - External integration points (Payment providers, Shipping carriers, ERP systems)
  - Deployment architecture (Kubernetes clusters, database tier, CDN)
- **Out of Scope:**
  - Detailed design within individual services (covered in service-level design docs)
  - Test strategies and plans (covered in QA documentation)
  - Operational runbooks (covered in SRE documentation)
  - Third-party system internals
- **Related Artifacts:**
  - Product Requirements Document (PRD-2024-001)
  - System Requirements Specification (SRS-ORDERFLOW-v3)
  - API Documentation (api.orderflow.example.com/docs)
  - ADR Repository (github.com/company/architecture-decisions)
  - Operations Guide (OPSGUIDE-ORDERFLOW-v2)

1.2 Organization of this Document

This document follows the Views and Beyond structure, organizing architecture information for efficient navigation and consumption.

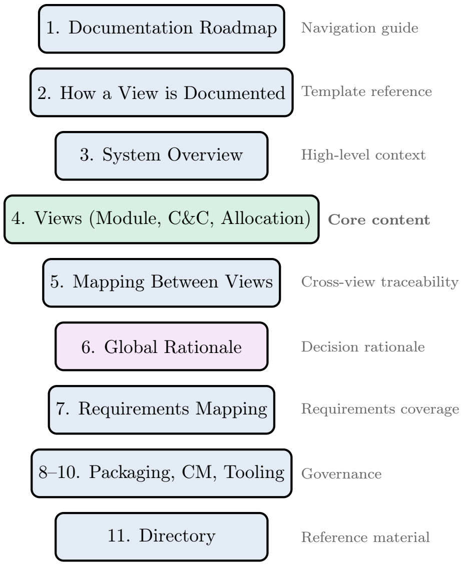


Figure 1: Document Structure Overview

- **Section 1: Documentation Roadmap** — Orientation to the document structure, stakeholder guidance, and view catalog.
- **Section 2: How a View is Documented** — Standard template used for all architectural views, ensuring consistency and completeness.
- **Section 3: System Overview** — High-level description of OrderFlow Commerce Platform including business context, capabilities, constraints, and stakeholders.
- **Section 4: Views** — The architectural views themselves:
  - Module views: Decomposition, Uses, Layered, Data Model
  - Component-and-Connector views: Service-Oriented, Event-Driven
  - Allocation views: Deployment, Work Assignment
- **Section 5: Mapping Between Views** — How elements correspond across views; rules for navigation and consistency.
- **Section 6: Global Rationale** — Cross-cutting architectural decisions, strategy, and trade-offs affecting multiple views.
- **Section 7: Requirements Mapping** — Traceability between requirements and architectural elements; coverage analysis.
- **Section 8: Packaging and Collaboration** — How documentation is packaged, published online, and maintained collaboratively.
- **Section 9: Configuration Management** — Versioning strategy, release process, and alignment with code releases.
- **Section 10: Presentation and Tooling** — Style guide, notation standards, and supporting tools.
- **Section 11: Directory** — Glossary, acronyms, and references for quick lookup.

1.3 View Overview

The following table summarizes all architectural views documented for OrderFlow Commerce Platform.

Table 2: Architectural Views Catalog

View Name	Style	Primary Concerns	Stakeholders
Decomposition View	Module	Code organization; team ownership; build structure	Developers; Tech Leads; Build Engineers
Uses View	Module	Dependencies; impact analysis; build order	Developers; Architects
Layered View	Module	Abstraction levels; portability; dependency rules	Architects; Developers
Data Model View	Module	Persistent data structures; data ownership	Data Engineers; DBAs; Developers
Service View	C&C (SOA)	Runtime services; APIs; synchronous communication	Developers; Operations; Integrators
Event-Driven View	C&C (Pub-Sub)	Asynchronous communication; event flows	Developers; Operations
Deployment View	Allocation	Infrastructure mapping; scaling; availability	Operations; SREs; Security
Work Assignment	Allocation	Team responsibilities; development coordination	Management; Tech Leads

1.4 Stakeholders and How They Use the Documentation

Different stakeholders have different information needs. This table guides each role to the most relevant sections.

Table 3: Stakeholder Navigation Guide

Stakeholder	Key Concerns	Recommended Sections
Executive / Sponsor	Business alignment; risk; investment decisions	System Overview; Global Rationale; Requirements Mapping
Product Manager	Feature mapping; capability coverage	System Overview; Requirements Mapping; Service View
Software Developer	Component responsibilities; interfaces; dependencies	Decomposition View; Uses View; Service View; Element Catalogs

Stakeholder	Key Concerns	Recommended Sections
Tech Lead	Team boundaries; integration points; technical decisions	All views; Global Rationale; Work Assignment View
Architect	Patterns; trade-offs; quality attribute support	All sections; emphasis on Rationale and Mappings
QA Engineer	Testability; component boundaries; scenarios	Service View; Behavior descriptions; Requirements Mapping
Operations / SRE	Deployment; monitoring; failure modes	Deployment View; Service View; Event-Driven View
Security Engineer	Security mechanisms; trust boundaries; data flows	Layered View; Service View; Deployment View; Global Rationale
DBA / Data Engineer	Data models; storage; data flows	Data Model View; Event-Driven View
New Team Member	System understanding; orientation	Documentation Roadmap; System Overview; then specific views
External Integrator	APIs; integration patterns; protocols	Service View; Context diagrams; Interface documentation

Best Practice

Reading Path for New Team Members:

1. Start with Section 3 (System Overview) for business context
2. Read the Decomposition View to understand code structure
3. Study the Service View to understand runtime behavior
4. Review Global Rationale to understand key decisions
5. Consult specific views as needed for assigned work

2 How a View is Documented

This section defines the standard template used for documenting each architectural view. Consistent structure enables efficient navigation and ensures completeness.

2.1 Standard View Template

Every architectural view follows this six-part structure:

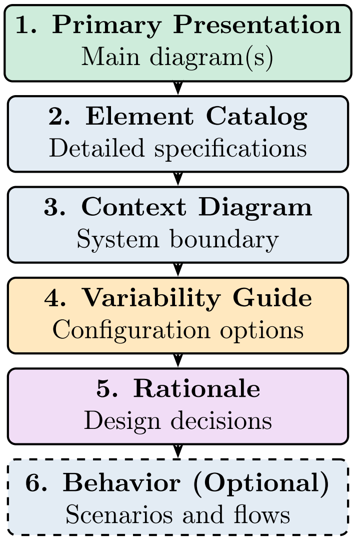


Figure 2: Standard View Template Structure

1. **Primary Presentation** — The main graphical or textual representation of the view. This is typically one or more diagrams showing the key elements and their relationships, accompanied by a notation key.
2. **Element Catalog** — Detailed documentation of each element and relation shown in the primary presentation. Includes element properties, interfaces, behaviors, and constraints.
3. **Context Diagram** — Shows the system or subsystem as a black box, depicting its boundary and interactions with external entities.
4. **Variability Guide** — Documents variation points: what can be configured, extended, or substituted, along with constraints on valid configurations.
5. **Rationale** — Explains why the design is the way it is, including decisions made, alternatives considered, and impact on quality attributes.
6. **Behavior of the Configuration** (Optional) — When needed, describes the behavior of the whole configuration rather than individual elements, typically through scenarios or sequence diagrams.

Warning

Avoid silently omitting sections. If a section does not apply to a particular view, explicitly state “Not applicable” or “None” with a brief explanation.

2.2 View Template Details

Template

Complete View Documentation Template

View Metadata

- **View Name:** Descriptive name (e.g., "Service-Oriented View")
- **View Style:** Category and specific style (e.g., "C&C / Service-Oriented Architecture")
- **Scope:** What portion of the system this view covers
- **Primary Stakeholders:** Roles this view primarily serves
- **Status:** Draft / Review / Approved
- **Last Updated:** Date of last modification

1. **Primary Presentation**
    - Main diagram(s) or textual representation
    - Notation key explaining all symbols, shapes, colors, and line styles
    - Brief highlights summarizing what the diagram shows
    - Multiple diagrams permitted when single diagram would be overcrowded

2. **Element Catalog**
    - **2.1 Elements and Properties:** For each element type and instance
    - **2.2 Relations and Properties:** For each relation type
    - **2.3 Element Interfaces:** APIs, contracts, protocols
    - **2.4 Element Behavior:** State machines, invariants, constraints

3. **Context Diagram**
    - System/subsystem shown as black box
    - External actors, systems, and interfaces
    - Brief narrative explaining key interactions

4. **Variability Guide**
    - Variation points (configuration, plugins, alternatives)
    - Binding times (compile, deploy, runtime)
    - Constraints on valid combinations
    - Instantiation instructions for known variants

5. **Rationale**
    - Key design decisions specific to this view
    - Alternatives considered and reasons for rejection
    - Impact on quality attributes
    - References to ADRs where applicable

6. **Behavior of the Configuration** (when applicable)
    - Key scenarios showing element interactions
    - Sequence diagrams or collaboration diagrams
    - Timing constraints or ordering requirements

2.3 Notation Standards

All views in this document follow these notation conventions:

Table 4: Standard Notation Elements

Element	**Symbol**	**Usage**
Module	Rectangle	Code unit (package, namespace, library)
Component	Rectangle with ports	Runtime unit (service, process)
Connector	Line with labels	Communication mechanism
External System	Dashed rectangle	System outside scope
Data Store	Cylinder	Database, file system, cache
Actor	Stick figure	Human user or external system
Boundary	Dashed border	System or subsystem boundary
Layer	Horizontal band	Abstraction level
Node	3D box	Infrastructure element
Synchronous call	Solid arrow	Request-response communication
Asynchronous message	Dashed arrow	Fire-and-forget or event
Data flow	Arrow with label	Data movement direction
Dependency	Dashed arrow	“Uses” or “depends on”
Containment	Nesting	Parent-child relationship

3 System Overview

This section provides a high-level understanding of OrderFlow Commerce Platform without committing to any particular view. It establishes shared context for all subsequent architectural descriptions.

3.1 Problem Statement

OrderFlow Commerce Platform addresses the need for a modern, scalable e-commerce platform that supports both B2C (business-to-consumer) and B2B (business-to-business) commerce. The platform replaces a legacy monolithic system that has reached its scalability limits and cannot support the business growth targets of 10x transaction volume over five years.

Key Business Drivers:

- **Scalability:** Handle Black Friday traffic peaks of 100,000 concurrent users
- **Global Expansion:** Support multi-region deployment and multi-currency transactions
- **Time-to-Market:** Enable independent deployment of features by autonomous teams
- **Partner Integration:** Provide robust APIs for marketplace sellers and B2B customers
- **Operational Excellence:** Achieve 99.95% availability with sub-second response times

3.2 System Capabilities

OrderFlow Commerce Platform provides the following major capabilities:

Table 5: System Capabilities

Capability	**Description**
Product Catalog	Browse, search, and filter products with faceted navigation; support for configurable and bundled products
Shopping Cart	Persistent cart with real-time inventory checks; saved for later; gift options
Checkout	Multi-step checkout with address validation, tax calculation, shipping options, and payment processing
Order Management	Order lifecycle from placement through fulfillment; modifications, cancellations, and returns
Inventory Management	Real-time stock levels across warehouses; reservations; backorder management
User Management	Registration, authentication, profiles, preferences, and address book
Payment Processing	Multiple payment methods; PCI-compliant processing; refunds and disputes
Search	Full-text search with relevance ranking, filters, and personalization
Notifications	Transactional emails, SMS, and push notifications for order updates

Capability	Description
Analytics	Business intelligence, funnel analysis, and recommendation engine data
Administration	Back-office tools for catalog, inventory, orders, and customer service

3.3 Users and External Stakeholders

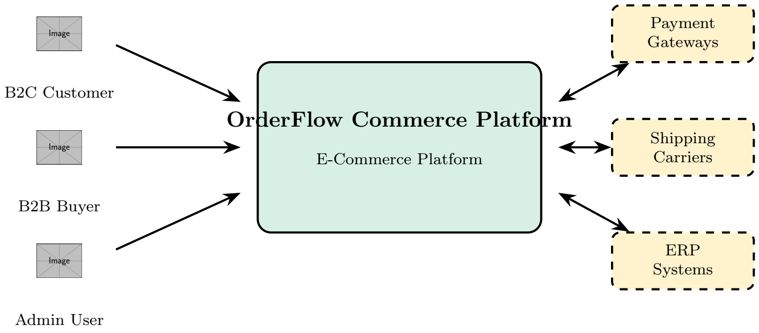


Figure 3: System Context Overview

User Types:

- **B2C Customers:** Individual consumers browsing and purchasing products via web and mobile
- **B2B Buyers:** Business purchasers with contract pricing, bulk ordering, and approval workflows
- **Marketplace Sellers:** Third-party merchants listing and fulfilling products
- **Customer Service:** Support agents handling inquiries, orders, and returns
- **Merchandisers:** Business users managing catalog, pricing, and promotions
- **Operations:** Warehouse staff managing inventory and fulfillment
- **Administrators:** IT staff managing system configuration and access

External Systems:

- Payment gateways (Stripe, PayPal, Adyen)
- Shipping carriers (FedEx, UPS, DHL)
- Tax calculation services (Avalara)
- ERP and financial systems (SAP, NetSuite)
- Marketing automation (Salesforce, HubSpot)
- Analytics platforms (Google Analytics, Segment)

3.4 Key Constraints and Context

Table 6: Architectural Constraints

Category	**Constraint**	**Impact on Architecture**
Technical	AWS cloud platform	All infrastructure designed for AWS services
Technical	Java/Kotlin primary languages	Service implementation technology stack
Technical	PostgreSQL for OLTP	Primary database; influences data model
Regulatory	PCI-DSS Level 1	Payment data isolation; encryption requirements
Regulatory	GDPR compliance	Data residency; consent management; right to deletion
Organizational	8 autonomous teams	Microservices aligned with team boundaries
Organizational	2-week sprint cycles	Incremental delivery; feature toggles
Schedule	November 2024 launch	MVP scope definition; phased rollout
Budget	$50K/month infrastructure	Cost-optimized architecture; auto-scaling
Integration	Legacy system coexistence	Strangler pattern; data synchronization

4 Views

This section contains the architectural views for OrderFlow Commerce Platform. Each view follows the standard template defined in Section 2.

4.1 Module Decomposition View

Module Decomposition View

View Metadata

- **View Style:** Module / Decomposition
- **Scope:** Entire OrderFlow Commerce Platform platform
- **Primary Stakeholders:** Developers, Tech Leads, Build Engineers
- **Status:** [APPROVED]

1. Primary Presentation

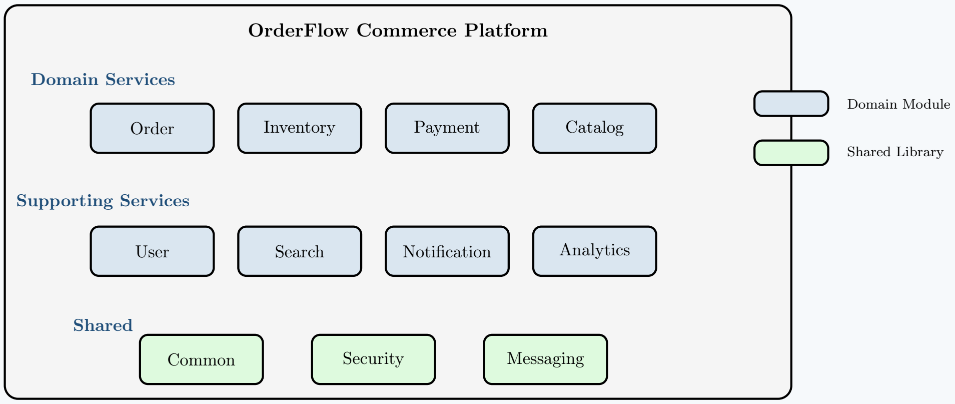


Figure 4: Module Decomposition – Top Level

Notation Key:

- Blue rectangles: Domain service modules (independently deployable)
- Green rectangles: Shared libraries (compile-time dependencies)
- Containment: Parent-child module relationship

2. Element Catalog

2.1 Module Elements

Module	Responsibility	Owner	Key Dependencies
Order	Order lifecycle management from cart to fulfillment	Order Team	Inventory, Payment, User, Common
Inventory	Stock levels, reservations, warehouse management	Inventory Team	Common, Messaging
Payment	Payment processing, refunds, PCI compliance	Payment Team	Security, Common
Catalog	Product information, categories, attributes	Catalog Team	Search, Common
User	Authentication, profiles, preferences	Identity Team	Security, Common
Search	Full-text search, indexing, relevance	Search Team	Catalog, Common
Notification	Email, SMS, push	Platform	Messaging, Common

4.2 Service-Oriented (C&C) View

Service-Oriented View

View Metadata

- **View Style:** Component-and-Connector / Service-Oriented Architecture
- **Scope:** Runtime services and synchronous communication
- **Primary Stakeholders:** Developers, Operations, Integrators
- **Status:** [APPROVED]

1. Primary Presentation

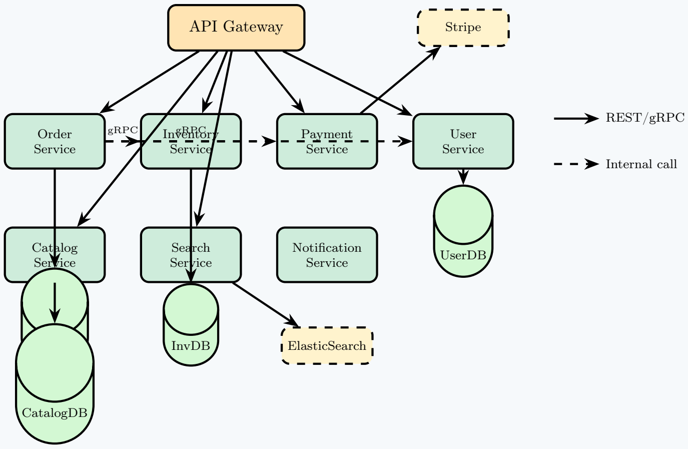


Figure 5: Service-Oriented View – Runtime Components

2. Element Catalog

2.1 Service Components

Service	API Style	Instances	SLA	Data Store
API Gateway	REST, GraphQL	3	99.99%	Redis (session)
Order Service	REST, gRPC	5	99.95%	PostgreSQL
Inventory Service	gRPC	3	99.95%	PostgreSQL
Payment Service	REST	3	99.99%	PostgreSQL (encrypted)
User Service	REST	3	99.95%	PostgreSQL
Catalog Service	REST	3	99.9%	PostgreSQL
Search Service	REST	2	99.9%	Elasticsearch
Notification Service	REST	2	99.5%	PostgreSQL

2.3 Service Interfaces

Each service exposes:

- REST API documented via OpenAPI 3.0 specification
- gRPC interfaces defined in Protocol Buffer files (for internal communication)

4.3 Deployment View

Deployment View

View Metadata

- **View Style:** Allocation / Deployment
- **Scope:** Production infrastructure
- **Primary Stakeholders:** Operations, SREs, Security
- **Status:** [APPROVED]

1. Primary Presentation

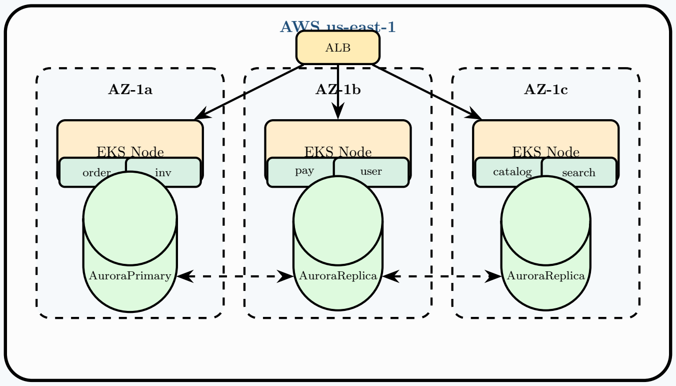


Figure 6: Deployment View – Multi-AZ Production

2. Element Catalog

Element	Type	Quantity	Specifications
EKS Cluster	Kubernetes	1	v1.28, 3 AZs, managed control plane
Worker Nodes	EC2	6-12	m6i.xlarge, auto-scaling group
Aurora PostgreSQL	RDS	1 cluster	r6g.xlarge, 3 instances, Multi-AZ
ElastiCache Redis	Managed	1 cluster	r6g.large, 3 nodes, cluster mode
ALB	Load Balancer	2	Internet-facing + internal
S3	Object Storage	3 buckets	Assets, backups, logs
CloudFront	CDN	1 dist	Global edge locations

5. Rationale

Key Decisions:

- **Multi-AZ Deployment:** Survives single AZ failure; 99.95% availability target (ADR-015)
- **EKS over ECS:** Kubernetes portability; team expertise; ecosystem (ADR-003)

4.4 View Packets

For large systems, view packets provide focused slices for specific subsystems or stakeholder groups.

Table 10: View Packets Registry

Packet ID	Scope	Contained Views	Stakeholders
VP-ORDER	Order processing subsystem	Order module decomposition; Order service C&C; Order data model	Order Team
VP-PAYMENT	Payment processing	Payment module; Payment service; PCI deployment	Payment Team; Security
VP-SEARCH	Search infrastructure	Search module; Search service; Elasticsearch deployment	Search Team
VP-SECURITY	Security architecture	Security module; Auth flows; Security deployment zones	Security Team

5 Mapping Between Views

This section explains how elements in different views correspond, enabling navigation across views and ensuring consistency.

5.1 Mapping Rules

Best Practice

Element Naming Convention:

- Same name across views implies same conceptual element
- Module “Order” maps to service “Order Service” maps to deployment “order-service-pod”
- When names differ, explicit mapping is provided in tables below

5.2 Module to Service Mapping

Table 11: Module to Runtime Service Mapping

Module	**Service Component**	**Relationship**	**Notes**
Order	Order Service	implements	1:1 mapping
Inventory	Inventory Service	implements	1:1 mapping
Payment	Payment Service	implements	1:1 mapping
Catalog	Catalog Service	implements	1:1 mapping
User	User Service	implements	1:1 mapping
Search	Search Service	implements	1:1 mapping
Notification	Notification Service	implements	1:1 mapping
Analytics	Analytics Service	implements	1:1 mapping
Common	(embedded)	compiled-into	Library, not runtime
Security	(embedded)	compiled-into	Library, not runtime
Messaging	(embedded)	compiled-into	Library, not runtime

5.3 Service to Deployment Mapping

Table 12: Service to Infrastructure Mapping

Service	**K8s Deployment**	**Database**	**Cache**
Order Service	order-deployment	order-db (Aurora)	order-cache (Redis)
Inventory Service	inventory-deployment	inventory-db (Aurora)	inventory-cache (Redis)
Payment Service	payment-deployment	payment-db (Aurora)	–
User Service	user-deployment	user-db (Aurora)	session-cache (Redis)
Catalog Service	catalog-deployment	catalog-db (Aurora)	catalog-cache (Redis)
Search Service	search-deployment	–	–
API Gateway	gateway-deployment	–	rate-limit-cache (Redis)

6 Global Rationale

This section captures cross-cutting architectural decisions and system-wide trade-offs that affect multiple views.

6.1 Architecture Strategy

OrderFlow Commerce Platform follows a **microservices architecture** with **event-driven integration**, designed to support:

- Independent deployment by autonomous teams
- Horizontal scaling of individual services
- Technology diversity where beneficial
- Resilience through isolation and graceful degradation

Architectural Styles Applied:

- **Microservices:** Domain-driven service decomposition
- **Event-Driven:** Asynchronous communication via Kafka
- **API Gateway:** Centralized entry point and cross-cutting concerns
- **CQRS:** Separate read models for search and reporting
- **Database per Service:** Data isolation and autonomy

6.2 Key Cross-Cutting Decisions

ADR-001: Aurora PostgreSQL for Primary Storage

Context: Need reliable, scalable relational storage for transactional data.
Decision: Use Amazon Aurora PostgreSQL for all primary data stores.
Consequences:

- (+) Auto-scaling storage; fast failover; managed backups
- (+) PostgreSQL compatibility; team expertise
- (-) AWS lock-in for database tier
- (-) Higher cost than self-managed PostgreSQL

ADR-002: Event-Driven Architecture with Kafka

Context: Services need to communicate without tight coupling; audit trail required.
Decision: Use Apache Kafka (Amazon MSK) for asynchronous event-driven communication.
Consequences:

- (+) Loose coupling between services
- (+) Event replay for recovery and debugging
- (+) Natural audit log of all state changes
- (-) Eventual consistency complexity
- (-) Operational overhead for Kafka cluster

ADR-003: Kubernetes (EKS) for Container Orchestration

Context: Need container orchestration for microservices deployment.
Decision: Use Amazon EKS for Kubernetes-based container orchestration.
Consequences:

- (+) Industry-standard orchestration; rich ecosystem
- (+) Portability to other clouds if needed
- (+) Strong team Kubernetes expertise
- (-) Complexity compared to simpler options (ECS)
- (-) Steeper learning curve for operations

6.3 Impact on Quality Attributes

Table 13: Architecture Support for Quality Attributes

Quality	Target	Architectural Support
Performance	<500ms p95	CDN caching; Redis caching; gRPC internal calls; read replicas
Availability	99.95%	Multi-AZ deployment; auto-scaling; circuit breakers; health checks
Scalability	100K concurrent	Horizontal pod scaling; database read replicas; Kafka partitioning
Security	PCI-DSS L1	Network isolation; encryption at rest/transit; secret management
Modifiability	1-day deploy	Microservices isolation; feature flags; backward-compatible APIs
Testability	80% coverage	Service isolation; contract testing; test containers

6.4 Open Questions

Table 14: Open Architectural Questions

ID	Question	Impact	Target Date
OQ-1	Multi-region deployment strategy?	Affects all views; latency; data residency	Q4 2024
OQ-2	GraphQL federation approach?	Gateway architecture; team coordination	Q3 2024
OQ-3	Machine learning platform integration?	New services; data pipelines	Q1 2025

7 Requirements Mapping

This section demonstrates traceability between requirements and architectural elements.

7.1 Mapping Strategy

Requirements are mapped to architecture using a centralized traceability matrix maintained in this document and synchronized with the requirements management tool (Jira). Each requirement is traced to:

- Architectural views that address it
- Specific elements responsible for satisfying it
- Validation approach (how satisfaction is verified)

7.2 Requirements Traceability Matrix

Table 15: Requirements Traceability Matrix

Req ID	Summary	Type	Addressed By	Validation
FR-001	User registration and login	Func	User Service; Security module	Integration test
FR-002	Product search with filters	Func	Search Service; Elasticsearch	E2E test
FR-003	Shopping cart management	Func	Order Service; Redis cache	Integration test
FR-004	Order placement	Func	Order Service; Payment Service	E2E test
FR-005	Payment processing	Func	Payment Service; Stripe integration	Contract test
QA-001	99.95% availability	Quality	Multi-AZ deployment; auto-scaling	SLA monitoring
QA-002	<500ms response (p95)	Quality	CDN; caching; gRPC	Load testing
QA-003	100K concurrent users	Quality	Horizontal scaling; Kubernetes	Load testing
QA-004	PCI-DSS compliance	Quality	Payment isolation; encryption	Audit
CON-001	AWS cloud platform	Constraint	All deployment elements	Architecture review
CON-002	GDPR data residency	Constraint	EU region deployment	Compliance audit

8 Packaging, Online Documentation, and Collaboration

8.1 Packaging Scheme

Architecture documentation is packaged for different audiences:

Table 16: Documentation Packages

Package	Contents	Audience
Full Architecture Doc	Complete document (this)	Architects; Tech Leads
Executive Summary	System Overview; Key Decisions (5 pages)	Executives; Sponsors
Developer Guide	Decomposition; Service views; Interfaces	Developers
Operations Guide	Deployment view; Runbooks	SREs; Operations
Security Package	Security architecture; Threat model	Security Team; Auditors
Onboarding Package	Overview; Key views; Glossary	New team members

8.2 Online Documentation

- **Wiki Root:** `wiki.company.com/architecture/orderflow`
- **One Page per View:** Each view has dedicated wiki page with embedded diagrams
- **Diagram Source:** Stored in Git alongside documentation; exported to wiki
- **Search:** Full-text search enabled across all architecture pages
- **Notifications:** Subscribers notified of significant updates

9 Configuration Management and Release Strategy

9.1 Configuration Management

- **Repository:** `github.com/company/orderflow-architecture`
- **Branch Strategy:** `main` for approved content; feature branches for changes
- **Tagging:** Semantic versioning aligned with major system releases
- **Code Alignment:** Architecture doc version noted in system release notes

9.2 Release Strategy

- **Major Releases:** Aligned with system major versions; full review cycle
- **Minor Updates:** Monthly for accumulated changes; Tech Lead approval
- **Patches:** As needed for corrections; peer review sufficient
- **Communication:** Release notes to `architecture-announce@company.com`

10 Presentation and Tooling

10.1 Presentation and Style Guide

- **Document Format:** LaTeX source; PDF output; wiki mirror
- **Diagrams:** TikZ for embedded; draw.io/Lucidchart for complex diagrams
- **Fonts:** Latin Modern (LaTeX default); consistent sizing
- **Colors:** Consistent palette defined in document preamble
- **Terminology:** As defined in Glossary; consistent throughout

10.2 Tooling

Table 17: Architecture Documentation Tools

Purpose	Tool	Usage
Document authoring	LaTeX, VS Code	Primary document creation
Diagrams	TikZ, draw.io, Lucidchart	Architecture diagrams
Version control	Git, GitHub	Source management; reviews
Wiki	Confluence	Online documentation
Modeling	Structurizr	C4 model diagrams
API docs	OpenAPI, Swagger UI	Interface documentation
ADRs	Markdown, adr-tools	Decision records

11 Directory

11.1 Glossary

Table 18: Glossary of Terms

Term	Definition
Architecture View	A representation of a system from the perspective of a related set of concerns
Component	A runtime entity that provides or consumes services through interfaces
Connector	A runtime pathway for interaction between components
Element	A fundamental piece of architecture (module, component, or node)
Module	A code unit that implements a coherent set of responsibilities
Primary Presentation	The main graphical or textual representation of a view
Quality Attribute	A measurable property of a system (performance, availability, etc.)
Stakeholder	An individual or organization with interests in the system
View Packet	A focused subset of views for a specific subsystem or audience
Viewpoint	A specification of conventions for constructing and using a view

11.2 Acronyms

Table 19: Acronyms

Acronym	Meaning
ADR	Architecture Decision Record
ALB	Application Load Balancer
API	Application Programming Interface
C&C	Component-and-Connector
CDN	Content Delivery Network
CQRS	Command Query Responsibility Segregation
EKS	Elastic Kubernetes Service

gRPC	Google Remote Procedure Call
MSK	Managed Streaming for Apache Kafka
PCI-DSS	Payment Card Industry Data Security Standard
RDS	Relational Database Service
SLA	Service Level Agreement
SRE	Site Reliability Engineering

11.3 References

1. Clements, P., et al. (2010). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison-Wesley.
2. Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley.
3. ISO/IEC/IEEE 42010:2011. *Systems and software engineering — Architecture description.*
4. Keeling, M. (2017). *Design It! From Programmer to Software Architect.* Pragmatic Bookshelf.
5. Brown, S. (2018). *The C4 Model for Visualising Software Architecture.* Leanpub.
6. Nygard, M. (2007). *Release It! Design and Deploy Production-Ready Software.* Pragmatic Bookshelf.
7. Newman, S. (2021). *Building Microservices* (2nd ed.). O'Reilly Media.
8. Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Addison-Wesley.