`ARCHITECTURE DESIGN SPECIFICATION (SAD) · ISO/IEC/IEEE 42010 & 4+1 VIEW STANDARD`

[아키텍처 설계 명세서] 엔터프라이즈 클라우드 네이티브 어플리케이션 아키텍처 설계서 (SAD)

문서 관리 항목	명세 내용	문서 관리 항목	명세 내용
문서 번호 (Doc ID)	`SAD-APP-2026-001`	**버전 (Version)**	`v1.0.0`
대상 시스템 / 범위	전사 백엔드 비즈니스 어플리케이션 표준	**문서 상태 (Status)**	`APPROVED (표준 규격 확정)`
설계 책임 (Architect)	엔터프라이즈 아키텍처(EA) 코어 그룹	**준거 표준 (Standard)**	`ISO/IEC/IEEE 42010 · 4+1 View · C4 Model`
런타임 기준 스택	`Java 21 LTS + Spring Boot 3.3.x`	**보안 등급**	`대외비 (Internal Engineering Standard)`

문서 제·개정 이력 (Revision History)

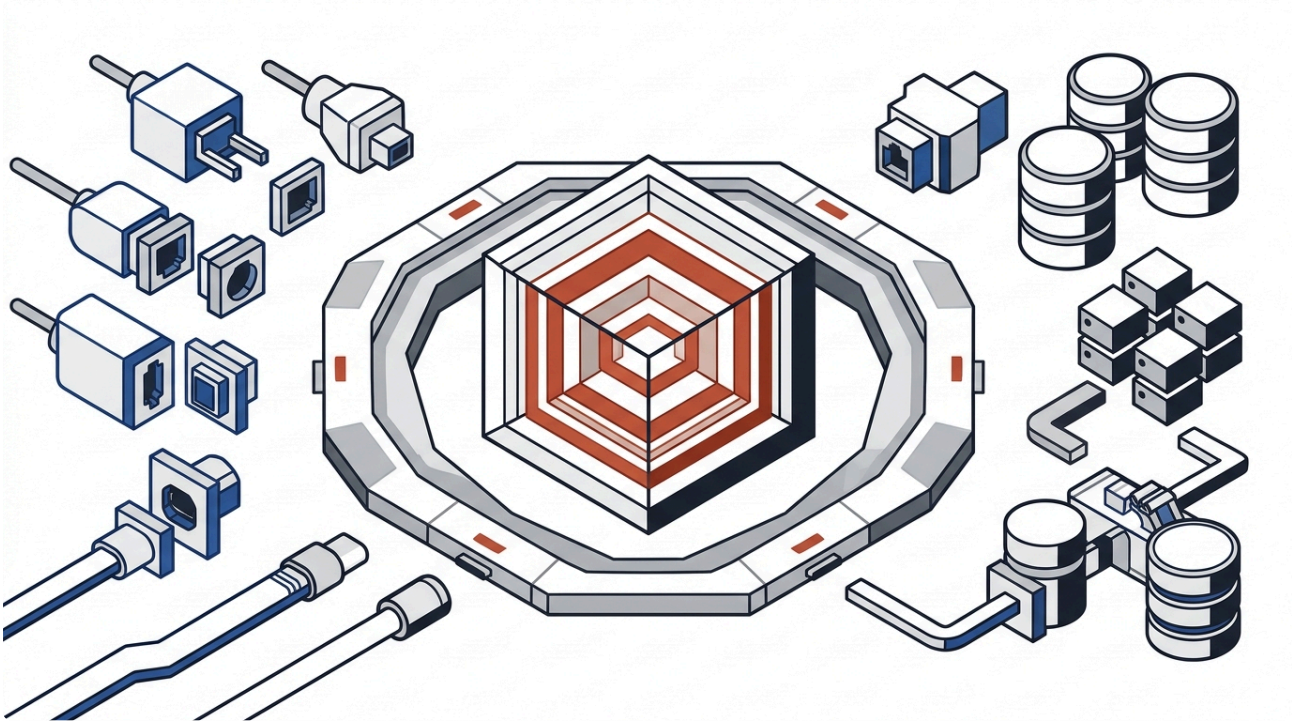
버전	일자	변경 구분	작성 / 승인자	주요 설계 제·개정 내역
`v0.5.0`	2026-10-15	초안 작성	수석 아키텍트 TF	헥사고날(Port & Adapter) 계층 구조 및 CQRS 데이터 뷰 초안 정의
`v0.9.0`	2026-10-22	검토 반영	플랫폼 엔지니어링 파트	Java 21 Virtual Threads, 다계층 캐시(L1/L2), Resilience4j 서킷브레이커 규격 추가
`v1.0.0`	2026-10-26	표준 승인	아키텍처 심의 위원회	4대 아키텍처 뷰(논리·프로세스·데이터·배포) 및 ArchUnit 적합성 검증 규칙 최종 확정

제1장. 설계 개요 및 아키텍처 원칙 (1. Architecture Overview & Principles)

> **[설계 핵심 정의 (Architecture Blueprint Summary)]**: 본 설계서는 전사 백엔드 어플리케이션의 구조적 일관성, 도메인 순수성, 고가용성을 보장하기 위해 **실용적 헥사고날 아키텍처(Pragmatic Hexagonal Architecture, Ports & Adapters)**와 **CQRS(명령·조회 책임 분리)**를 결합한 표준 어플리케이션 참조 아키텍처(Reference Architecture)를 명세함.

1.1 시스템 컨텍스트 및 도메인 코어 격리 개념 (System Context View)

- **시스템 경계 및 내·외부 분리**:
  - 어플리케이션 중심부의 **도메인 코어(Hexagon Core)**는 순수 비즈니스 규칙과 불변식(Invariant)만 보유하며, 외부 프로토콜(REST, GraphQL, gRPC, Kafka) 및 영속성 인프라(RDBMS, Redis, 외부 API)는 **포트(Port) 인터페이스와 어댑터(Adapter) 플러그인**을 통해서만 결합됨.



헥사고날(Port & Adapter) 클린 아키텍처 도메인 코어 및 내외부 어댑터 플러그인 구조 개념도

1.2 아키텍처 핵심 설계 원칙 명세 (Architecture Principles Specification)

원칙 ID	아키텍처 설계 원칙명	상세 설계 규격 및 강제 사항 (Design Rule Specification)	검증 방법 (Fitness Check)
`AP-01`	**의존성 역전 및 도메인 순수성 (DIP & Clean Domain)**	• 모든 컴파일 타임 소스코드 의존성은 **외부(`adapter`) → 내부(`application` → `domain`) 단방향**으로만 향함 • `domain.*` 패키지 내 JPA(`@Entity`), Spring(`@Component`), Jackson 어노테이션 침투를 전면 금지함	ArchUnit `ARCH-RULE-01`
`AP-02`	**명령(Command)과 조회(Query) 경로 분리 (CQRS)**	• 상태 변경(`Command`)은 UseCase Port 및 Domain Aggregate를 통해 불변식 검증 후 트랜잭션 커밋 수행 • 단순/복합 조회(`Query`)는 도메인 엔티티 변환 없이 QueryDSL/jOOQ Projection DTO로 직결 반환함	Code Review / QueryPort 분리
`AP-03`	**협소 트랜잭션 경계 및 외부 I/O 분리 (Narrow Tx)**	• DB 커넥션 점유 시간을 **평균 50ms 이내**로 유지하기 위해 `@Transactional` 내부에서 외부 HTTP/SMTP/Kafka 동기 호출을 엄격 금지함 (`Facade` + `TransactionTemplate`으로 분리)	ArchUnit `ARCH-RULE-03`
`AP-04`	**완전 무상태(Stateless) 및 실패 격리 설계 (Fault Isolation)**	• 파드 로컬 메모리/디스크에 클라이언트 세션 저장 금지 (100% 스케일아웃 호환) • 모든 외부 네트워크 호출은 `Resilience4j` 서킷브레이커·타임아웃·벌크헤드를 필수 적용함	K8s Chaos Test / Resilience4j
`AP-05`	**관측 가능성 내재화 (Observability by Default)**	• 모든 요청·이벤트·SQL·외부 호출은 `W3C TraceContext`(`traceId`, `spanId`)로 자동 연결되며, 로그·메트릭·트레이스 3대 신호를 표준 포맷으로 노출함	OpenTelemetry / Grafana Tempo

1.3 비기능 품질 속성 및 목표 SLA/SLO 명세 (Quality Attribute Specification)

요구사항 ID	품질 속성 분류	정량 목표 지표 (Target SLA / SLO)	아키텍처 실현 메커니즘 (Architectural Tactics)
`NFR-01`	**응답 지연 (Latency)**	• 조회(Query) API: **P95 ≤ 40ms, P99 ≤ 120ms** • 명령(Command) API: **P95 ≤ 80ms, P99 ≤ 200ms**	• Java 21 Virtual Threads 기반 논블로킹 I/O 처리 • L1(Caffeine) + L2(Redis) 다계층 캐시 및 Read Replica 라우팅
`NFR-02`	**처리량 (Throughput)**	• 단일 파드(2 vCPU, 4GB RAM) 기준 **≥ 1,200 TPS** • 클러스터 HPA 확장 시 **≥ 30,000 TPS** 선형 확장	• `spring.jpa.open-in-view=false` 및 `hikari.auto-commit=false` 적용 • CPU 60% 임계치 기반 K8s HPA(3~30 Pods) 자동 스케일아웃
`NFR-03`	**가용성 (Availability)**	• 연간 서비스 가용성 **≥ 99.95%** (연간 비계획 중단 ≤ 4.38시간) • 배포 중 에러 발생률 **0.00%**	• Multi-AZ 3개 가용영역 분산 배치 + PodDisruptionBudget(`minAvailable: 2`) • `PreStop(sleep 5)` + `Graceful Shutdown(30s)` 무중단 롤링/카나리 배포
`NFR-04`	**데이터 정합성 (Consistency)**	• 로컬 도메인 트랜잭션 ACID **100% 보장** • 도메인 간 이벤트 최종 일관성 도달 **P99 ≤ 1.5s**	• `Transactional Outbox` 테이블 원자적 커밋 + Debezium/Poller 릴레이 • 컨슈머 `idempotency_key` 기반 멱등성 보장
`NFR-05`	**단위 테스트 속도 (Testability)**	• 도메인/애플리케이션 계층 단위 테스트 커버리지 **≥ 85%** • 단위 테스트 500건 전체 실행 시간 **≤ 3.0초**	• 도메인 계층 외부 의존성 제거로 Spring Context 기동 없는 순수 JUnit 5 + Fake Port 메모리 테스트 수행

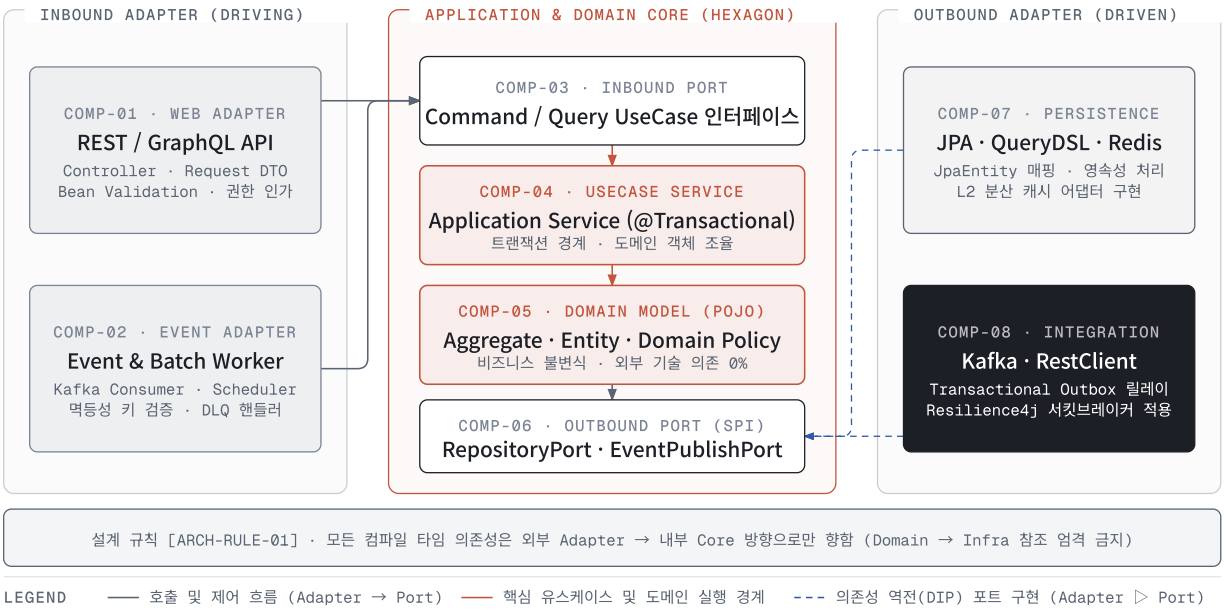
제2장. 논리 아키텍처 및 컴포넌트 설계 (2. Logical & Component View)

2.1 논리 계층 및 헥사고날(Port & Adapter) 구조도

- **구조 명세**: 어플리케이션 내부는 `Inbound Adapter (Driving)` → `Application & Domain Core (Hexagon)` ← `Outbound Adapter (Driven)` 의 3대 영역·8개 표준 컴포넌트(`COMP-01` ~ `COMP-08`)로 구성됨.

VIEW 01 · LOGICAL & COMPONENT ARCHITECTURE (HEXAGONAL PORTS & ADAPTERS)

[논리 뷰] 헥사고날(Port & Adapter) 컴포넌트 및 의존성 구조도



View 1 논리 뷰 — 헥사고날(Port & Adapter) 컴포넌트 및 의존성 역전(DIP) 구조도

2.2 컴포넌트 카탈로그 및 책임 명세표 (Component Specification Catalog)

컴포넌트 ID	컴포넌트명	소속 패키지 계층	핵심 설계 책임 (Core Responsibilities)	제공/구현 인터페이스 및 의존성 제약
`COMP-01`	**Web Inbound Adapter**	`adapter.in.web`	• HTTP/REST/GraphQL 요청 수신 및 구문 검증 (`@Valid`) • Request DTO ↔ `UseCase` `Command/Query` 변환 및 응답 래핑	• 호출: `COMP-03 (Inbound Port)` • 금지: `Domain Model` 직접 조작 및 `COMP-07` 직접 호출
`COMP-02`	**Event Inbound Adapter**	`adapter.in.event`	• Kafka 토픽 메시지 소비, 스키마 역직렬화 (Avro/JSON) • 중복 수신 방지를 위한 **멱등성 키(`eventId`) 선검증** 및 DLT 위임	• 호출: `COMP-03 (Inbound Port)` • 설정: `AckMode.MANUAL_IMMEDIATE` 수동 오프셋 커밋
`COMP-03`	**Inbound Port (UseCase API)**	`application.port.in`	• 어플리케이션 코어가 외부에 노출하는 **비즈니스 유스케이스 계약 인터페이스** • 불변 입력 객체 (`Command` , `Query` Record)에서 자가 유효성 검증 수행	• 구현체: `COMP-04 (Application Service)` • 순수 Java 인터페이스 + Record로 정의
`COMP-04`	**UseCase Application Service**	`application.service`	• 유스케이스 워크플로우 조율 및 **트랜잭션 경계 (`@Transactional`)** 시작/종료 • `Outbound Port` 를 통해 도메인 모델 로드 → 도메인 메서드 실행 → 상태 저장	• 구현: `COMP-03` / 호출: `COMP-05` , `COMP-06` • 금지: 구체적 인프라 기술 (JPA/Feign/Kafka) 클래스 import
`COMP-05`	**Pure Domain Model**	`domain.model` / `domain.service`	• **Aggregate Root, Entity, Value Object, Domain Policy** 보유 • 핵심 비즈니스 상태 전이(`order.pay()` , `order.cancel()`) 및 불변식 강제	• 외부 라이브러리 의존성 **0%** (순수 Java POJO) • 도메인 이벤트(`DomainEvent` 생성 및 반환

컴포넌트 ID	컴포넌트명	소속 패키지 계층	핵심 설계 책임 (Core Responsibilities)	제공/구현 인터페이스 및 의존성 제약
`COMP-06`	**Outbound Port (SPI)**	`application.port.out`	• 어플리케이션 코어가 외부 인프라(DB, 캐시, 메시징, 외부 API)에 요구하는 **기능 명세 인터페이스(Service Provider Interface)**	• 세분화 원칙(ISP): `LoadOrderPort`, `SaveOrderPort`, `PublishOrderEventPort`로 역할별 분리 정의
`COMP-07`	**Persistence & Cache Adapter**	`adapter.out.persistence` `adapter.out.cache`	• `COMP-06` 영속성 포트를 구현하며 `Domain Model ↔ JpaEntity` 양방향 매핑 수행 • Spring Data JPA, QueryDSL 동적 쿼리 및 Redis 캐시 연산 캡슐화	• 구현: `COMP-06 (Outbound Port)` • 내부 전용: `JpaEntity`는 본 I 키지 외부로 노출 금지
`COMP-08`	**Messaging & External Adapter**	`adapter.out.messaging` `adapter.out.client`	• `Transactional Outbox` 저장/릴레이 및 Kafka Producer 발행 수행 • 외부 REST/gRPC 연동(`RestClient`) 및 `Resilience4j` 서킷 브레이커/Fallback 캡슐화	• 구현: `COMP-06 (Outbound Port)` • 외부 연동 DTO는 어댑터 내부에서 도메인 값 객체로 변환

2.3 표준 패키지 구조 및 모듈 경계 규격 (Package Hierarchy Specification)

```TEXT
com.company.{domain}
├── adapter
│   ├── in
│   │   ├── web                    # [COMP-01] *Controller, *Request, *Response, *WebMapper
│   │   ├── event                  # [COMP-02] *EventConsumer, *EventPayload
│   │   └── scheduler              # [COMP-02] *JobScheduler
│   └── out
│       ├── persistence            # [COMP-07] *PersistenceAdapter, *JpaEntity, *JpaRepositor
│       ├── cache                  # [COMP-07] *CacheAdapter, RedisKeySpec
│       ├── messaging              # [COMP-08] *OutboxAdapter, OutboxJpaEntity, *KafkaProduce
│       └── client                 # [COMP-08] *ExternalClientAdapter, *FeignClient / *RestCl
├── application
│   ├── port
│   │   ├── in                     # [COMP-03] *UseCase, *Query, command/*Command (Java Recor
│   │   └── out                    # [COMP-06] Load*Port, Save*Port, Record*OutboxPort, Reque
│   └── service                    # [COMP-04] *CommandService, *QueryService, *OrchestratorF
├── domain
│   ├── model                      # [COMP-05] AggregateRoot, Entity, ValueObject (순수 POJO)
│   ├── service                    # [COMP-05] *DomainPolicy, *DomainService
│   ├── event                      # [COMP-05] *DomainEvent
│   └── exception                  # [COMP-05] *DomainException, DomainErrorCode
└── common                         # [공통 커널] config/*, security/*, error/GlobalExceptionHan
```

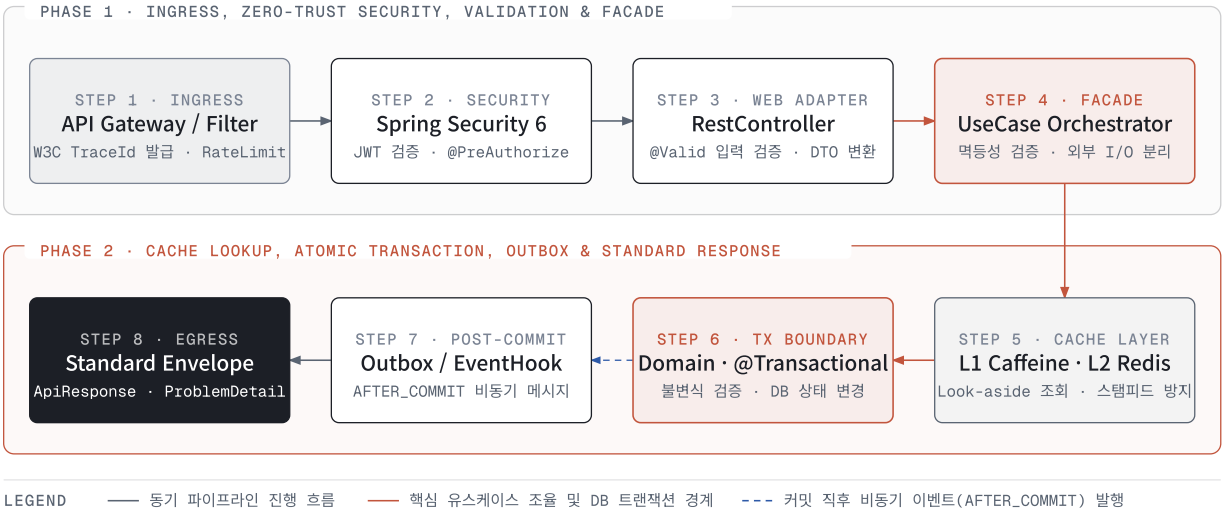
제3장. 프로세스 및 런타임 파이프라인 설계 (3. Process & Runtime Sequence View)

3.1 표준 요청 처리 8단계 파이프라인 다이어그램 (Process Pipeline View)

- **동적 실행 흐름**: 클라이언트 요청은 `STEP 1(Ingress)` 부터 `STEP 8(Egress)` 까지 표준화된 필터·인터셉터·어댑터 체인을 통과하며, 비즈니스 코어는 오직 `STEP 4 ~ STEP 6` 구간에서만 실행됨.

VIEW 02 · PROCESS & RUNTIME PIPELINE VIEW (8-STAGE SEQUENCE)

[프로세스 뷰] 표준 요청 처리 파이프라인 및 횡단 관심사 흐름



View 2 프로세스 뷰 — 표준 API 요청 처리 파이프라인 및 횡단 관심사(Cross-Cutting) 흐름도

3.2 단계별 런타임 처리 계약 명세표 (Pipeline Stage Contract Specification)

단계 ID	단계명 (Stage)	처리 컴포넌트	입력 계약 (Input Contract)	런타임 처리 규칙 및 타임아웃/트랜잭션 규격	출력 및 예외 계 Error)
`STEP 1`	**Ingress & Trace**	API Gateway / `TracingFilter`	HTTP Request + `traceparent` 헤더	• W3C `traceId` / `spanId` 생성 후 SLF4J MDC에 바인딩 • IP/Token 기준 Rate Limit(초당 200 req) 초과 시 즉시 차단	통과 시 `STEP` `HTTP 429` Requests
`STEP 2`	**Zero-Trust Auth**	`Spring Security 6 FilterChain`	`Authorization: Bearer <JWT>`	• OAuth2 JWK 공개키 서명 검증 및 만료 시각 확인 • `SecurityContext` 에 불변 `Principal(userId, roles)` 등록	통과 시 `STEP` `HTTP 401` ProblemDet
`STEP 3`	**DTO Validation**	`COMP-01` (`@RestController`)	JSON Body / Path / Query Param	• Jakarta `@Valid` 구문 검증 (Null, 범위, 정규식) • `WebMapper` 를 통해 불변 `Command` / `Query` Record로 변환	통과 시 `STEP` `HTTP 400` (InvalidPa
`STEP 4`	**UseCase Facade**	`COMP-04` (`Orchestrator`)	`Command` + `Idempotency-Key`	• 변경성 요청의 `Idempotency-Key` Redis 선조회(중복 요청 차단) • 외부 HTTP 호출이 필요한 경우 **DB 트랜잭션 진입 전/후에 분리 실행**	중복 요청 시 반환 / 정상 시
`STEP 5`	**Multi-Level Cache**	`COMP-07` (`CacheAdapter`)	Cache Key (`domain:v1:id`)	• 조회 경로 시 `L1(Caffeine)` → `L2(Redis)` 순차 Look-aside 조회 • Cache Miss 시에만 DB 조회 후 비동기 캐시 워밍(Warm-up)	Cache Hit 시 행 (응답 ≤ 5m
`STEP 6`	**Tx & Domain Core**	`COMP-04` + `COMP-05`	Loaded Domain Aggregate	• `@Transactional` 또는 `TransactionTemplate` 경계 진입 (점유 ≤ 50ms) • 도메인 엔티티 비즈니스 메서드 실행 및 불변식 검증 후 상태 영속화	정합성 위반 시 `DomainExc` 리 Tx 즉시 Ro
`STEP 7`	**Outbox & Event**	`COMP-08` (`OutboxAdapter`)	`DomainEvent` 목록	• 도메인 변경과 **동일 물리 트랜잭션 내 `outbox_event` 테이블에 INSERT** • 물리 커밋 직후 (`AFTER_COMMIT`) 비동기 릴레이가 Kafka 발행	커밋 실패 시 (께 롤백되어 유

단계 ID	단계명 (Stage)	처리 컴포넌트	입력 계약 (Input Contract)	런타임 처리 규칙 및 타임아웃/트랜잭션 규격	출력 및 예외 처 Error)
`STEP 8`	**Standard Egress**	`ApiResponse` / `ExceptionHandler`	`UseCase Result` 또는 `Exception`	• 정상 응답: `ApiResponse<T>(success, data, traceId, timestamp)` 래핑 • 예외 발생: `GlobalExceptionHandler`가 **RFC 9457** `ProblemDetail`로 변환	`HTTP 200/2` `HTTP 4xx/5` applicatio

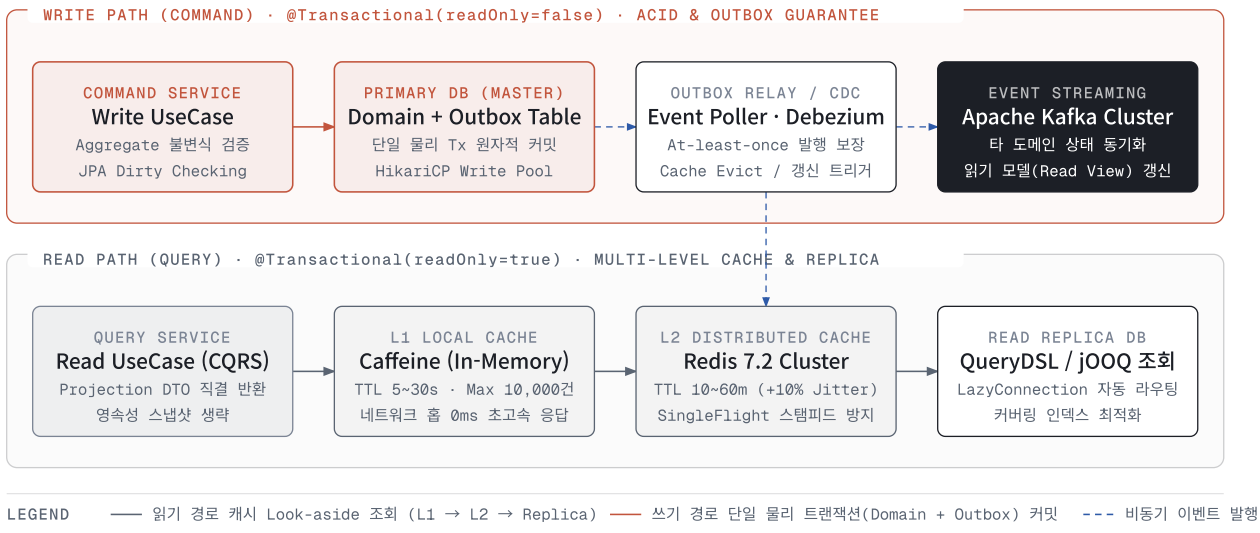
제4장. 데이터 및 영속성 아키텍처 설계 (4. Data & Persistence View)

4.1 CQRS 데이터 경로 분리, 다계층 캐시(L1/L2) 및 Outbox 아키텍처 다이어그램

- **설계 명세**: 쓰기 경로(`Command`)와 읽기 경로(`Query`)의 데이터베이스 커넥션 및 모델을 물리적·논리적으로 분리하여, 대량 조회 트래픽이 핵심 트랜잭션 처리 성능을 침해하지 않도록 설계함.

VIEW 03 · DATA, PERSISTENCE & MULTI-LEVEL CACHE ARCHITECTURE (CQRS)

[데이터 뷰] CQRS Read/Write 분리 및 다계층 캐시·Outbox 구조도



View 3 데이터 뷰 — CQRS Read/Write 분리, 다계층 캐시(L1/L2) 및 Transactional Outbox 데이터 구조도

4.2 데이터 저장소 및 다계층 캐시 정책 명세표 (Data & Cache Specification)

구분 ID	저장소 계층	적용 기술	대상 데이터 및 키 규격	TTL / 정합성 / 라우팅 설계 규격	장애 및 만료 (Stamped 응 설계
`DATA-01`	**Primary RDBMS (Write Master)**	Aurora PostgreSQL 16 / MySQL 8.0	핵심 도메인 상태 테이블 + `outbox_event` 테이블	• `@Transactional(readOnly=false)` 진입 시 `LazyConnectionDataSourceProxy` 가 Master Pool로 라우팅 • `hikari.auto-commit=false` , `open-in-view=false` 강제	• Multi-AZ 예비 복제본 (Failover ≤ 초) • 데드락 방지 위해 복합 PK 인덱스 정렬 통일
`DATA-02`	**Read Replica DB (Read Slave)**	Aurora Read Replica (최소 2노드)	조회 전용 QueryDSL / jOOQ Projection 쿼리	• `@Transactional(readOnly=true)` 선언 시 Read Pool로 자동 라우팅 • JPA 영속성 컨텍스트 스냅샷 생략으로 메모리 35% 절감	• 복제 지연 (Replicatio Lag > 100r 감지 시 중요 만 Primary Fallback 라
`DATA-03`	**L1 Local Cache (In-Memory)**	Caffeine 3.1	공통 코드, 카테고리, 권한 정책 (`{svc}:l1:{domain}:{id}`)	• 파드 힙 메모리 내 저장 (`maximumSize=10,000` , `expireAfterWrite=5~30s`) • 네트워크 I/O 비용 0ms 보장	• 수정 발생 Kafka 브로 트 토픽(`ca evict-top` 을 수신하여 드 L1 즉시 두
`DATA-04`	**L2 Distributed Cache**	Redis 7.2 Cluster (3 Master / 3 Replica)	상품 상세, 유저 프로필, 집계 뷰, 멱등성 키 (`{svc}:l2:{ver}:{key}`)	• 기본 TTL `10~60분` + **랜덤 Jitter(±10%)** 부여로 동시 만료 폭주 (Cache Avalanche) 차단 • 직렬화: `Kryo` 또는 `Jackson Afterburner`	• 핫 키 만료 일 스레드만 조회하는 **SingleFlig 산 뮤텍스 락** 용 • Redis 장애 서킷브레이커 시 Open되어 Read Repli 우회

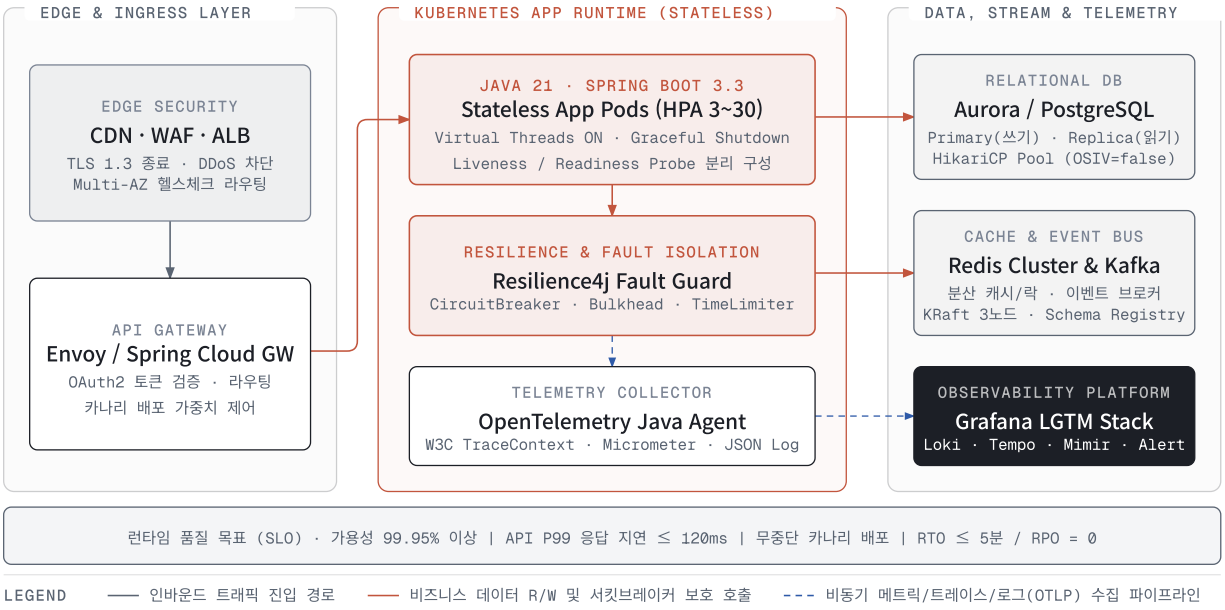
제5장. 물리 배포 및 회복탄력성·관측성 설계 (5. Deployment, Resilience & Observability View)

5.1 클라우드 네이티브 런타임 및 배포 토폴로지 다이어그램 (Deployment Topology View)

- **인프라 및 런타임 명세**: 외부 트래픽은 `CDN/WAF/ALB` 와 `API Gateway` 를 거쳐 Kubernetes 클러스터 내 **Java 21 Virtual Threads 기반 Stateless App Pods**로 전달되며, 모든 파드는 `OpenTelemetry Java Agent` 를 통해 메트릭·트레이스·로그를 `Grafana LGTM` 스택으로 실시간 전송함.

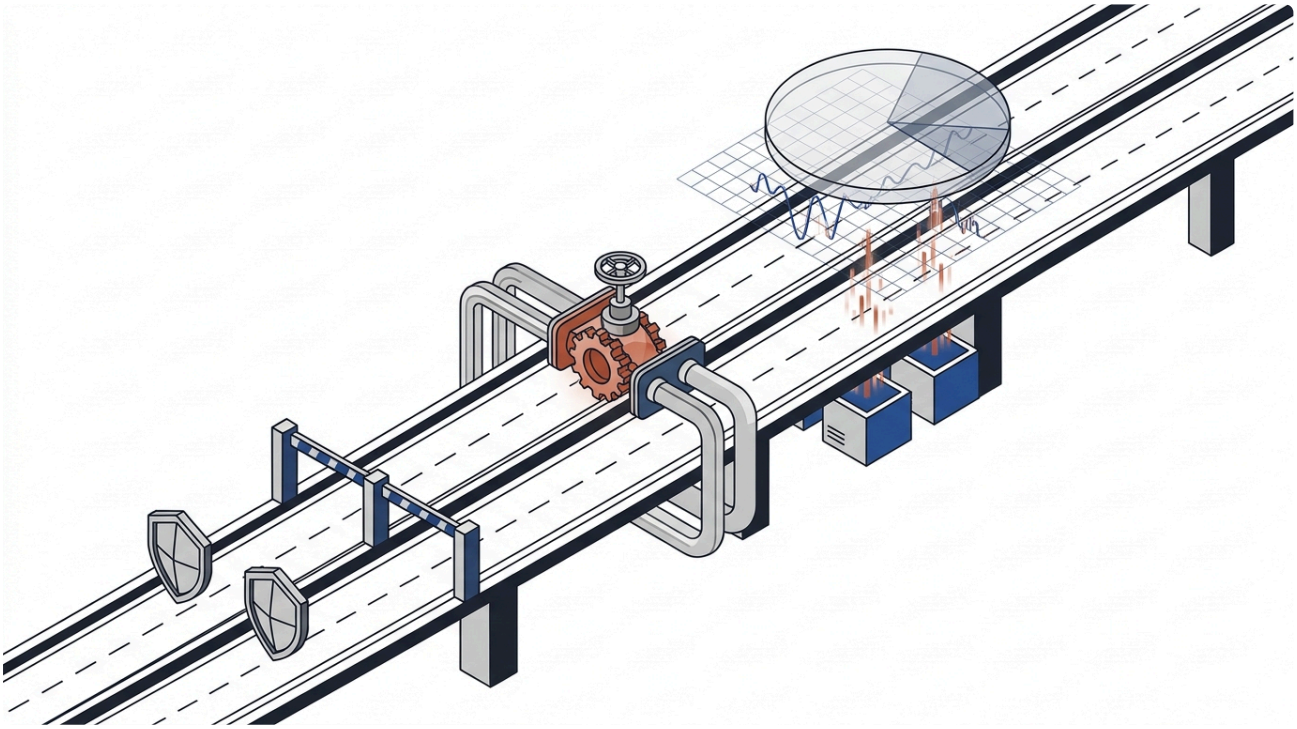
VIEW 04 · DEPLOYMENT, RESILIENCE & OBSERVABILITY TOPOLOGY

[배포 뷰] 클라우드 네이티브 런타임 토폴로지 및 관측성 설계도



View 4 배포 뷰 — 클라우드 네이티브 K8s 런타임 토폴로지 및 고가용성·관측성 설계도

5.2 회복탄력성(Resilience), 보안 및 관측성(Observability) 상세 설계



어플리케이션 런타임 회복탄력성(서킷브레이커), Zero-Trust 보안 및 실시간 관측성 파이프라인 개념도

- **회복탄력성·보안·관측성 파라미터 명세표 (Runtime Engineering Specification)**

규격 ID	설계 영역	적용 컴포넌트	표준 설정 파라미터 및 임계치 명세 (Parameter Specification)	동작 및 장애 격리 규칙
`RES-01`	**Circuit Breaker (서킷브레이커)**	Resilience4j `CircuitBreaker`	• `slidingWindowType: COUNT_BASED`, `slidingWindowSize: 100` • `failureRateThreshold: 50%`, `slowCallDurationThreshold: 2000ms` • `waitDurationInOpenState: 15s`, `permittedNumberOfCallsInHalfOpenState: 10`	• 외부 연동 실패율 50% 도달 시 **즉시 서킷 Open (0ms Fail-Fast)** • Fallback 메서드(캐시 기본값 또는 대기열 적재)로 장애 전파 차단
`RES-02`	**Bulkhead & Timeout (격리벽/타임아웃)**	Resilience4j `Bulkhead` / `RestClient`	• `ConnectTimeout: 1000ms`, `ReadTimeout: 3000ms` (무한 대기 금지) • `SemaphoreBulkhead`: 외부 연동 대상별 최대 동시 호출 수 `maxConcurrentCalls: 50` 제한	• 특정 외부 파트너사 API 지연 시에도 해당 Bulkhead 슬롯(50개)만 소진되고 전체 Virtual Thread 및 DB 커넥션 풀은 100% 보호됨
`RES-03`	**Graceful Shutdown & K8s Probe**	Spring Boot Actuator + K8s	• `server.shutdown: graceful`, `spring.lifecycle.timeout-per-shutdown-phase: 30s` • `livenessProbe`: `/actuator/health/liveness` (내부 데드락 감지) • `readinessProbe`: `/actuator/health/readiness` (트래픽 수신 가능 여부)	• 주의: `livenessProbe`에 외부 DB/Redis 상태를 포함하면 일시적 DB 순단 시 전 파드가 동시 재시작(Cascading Restart)되므로 **외부 의존성 헬스는 `readinessProbe`에만 제한 연결**
`RES-04`	**통합 관측성 (Logs / Metrics / Traces)**	OpenTelemetry + Logback JSON	• **Trace**: W3C `traceparent` 100% 전파 (정상 요청 10% 샘플링, 에러/슬로우 요청 **100% Tail-based 샘플링**) • **Log**: `traceId`, `spanId`, `userId`, `clientIp` 자동 주입 및 주민번호/카드번호 정규식 마스킹	• 알람 기준: 5분간 5xx 에러율 > 1.0%, HikariCP Pending Threads > 5, API P99 > 300ms 발생 시 PagerDuty/Slack 즉시 발보

제6장. 표준 인터페이스 및 코드 템플릿 명세 (6. Standard Interface & Code Specification)

6.1 계층별 표준 코드 블루프린트 (Reference Implementation Templates)

- **[Template 1] 순수 도메인 어그리거트 모델 (`COMP-05` — 외부 기술 의존성 0% POJO)**

JAVA

```
package com.company.order.domain.model;

public class Order {
    private final OrderId id;
    private final BuyerId buyerId;
    private Money totalAmount;
    private OrderStatus status;
    private final List<OrderDomainEvent> domainEvents = new ArrayList<>();

    public static Order place(OrderId id, BuyerId buyerId, List<OrderLine> lines) {
        if (lines == null || lines.isEmpty()) {
            throw new OrderDomainException(OrderErrorCode.EMPTY_ORDER_LINES);
        }
        Money total = lines.stream().map(OrderLine::calculateAmount).reduce(Money.ZERO, Mo
        Order order = new Order(id, buyerId, total, OrderStatus.PLACED);
        order.domainEvents.add(new OrderPlacedEvent(id.value(), buyerId.value(), total.amo
        return order;
    }

    public void confirmPayment(String pgTransactionId) {
        if (this.status != OrderStatus.PLACED) {
            throw new OrderDomainException(OrderErrorCode.INVALID_ORDER_STATE_TRANSITION);
        }
        this.status = OrderStatus.PAID;
        this.domainEvents.add(new OrderPaidEvent(this.id.value(), pgTransactionId));
    }
}
```

- **[Template 2] 유스케이스 애플리케이션 서비스 (`COMP-04` — 포트 조합 및 트랜잭션 경계)**

JAVA

```
package com.company.order.application.service;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class PlaceOrderCommandService implements PlaceOrderUseCase {

    private final LoadOrderPort loadOrderPort;
    private final SaveOrderPort saveOrderPort;
    private final RecordOutboxPort recordOutboxPort;

    @Override
    @Transactional(rollbackFor = Exception.class)
    public OrderResult placeOrder(PlaceOrderCommand command) {
        // 1. 멱등성 검증 (중복 주문 요청 방어)
        if (loadOrderPort.existsByIdempotencyKey(command.idempotencyKey())) {
            return loadOrderPort.loadResultByIdempotencyKey(command.idempotencyKey());
        }
        // 2. 순수 도메인 어그리거트 생성 및 비즈니스 규칙 실행
        Order order = Order.place(OrderId.newId(), new BuyerId(command.buyerId()), command

        // 3. Outbound Port를 통한 상태 영속화 및 Transactional Outbox 원자적 기록
        saveOrderPort.save(order, command.idempotencyKey());
        recordOutboxPort.recordEvents(order.pullDomainEvents());

        return OrderResult.from(order);
    }
}
```

- **[Template 3] 공통 응답(`ApiResponse<T>`) 및 RFC 9457 `ProblemDetail` 에러 규격 (`COMP-01`)**

JAVA

```
package com.company.order.common.response;

public record ApiResponse<T>(
    boolean success,
    T data,
    String traceId,
    Instant timestamp
) {
    public static <T> ApiResponse<T> ok(T data) {
        return new ApiResponse<>(true, data, MDC.get("traceId"), Instant.now());
    }
}
```

제7장. 아키텍처 적합성 검증 및 기술 스택 명세 (7. Architecture Fitness & Tech Stack BOM)

7.1 ArchUnit 자동화 아키텍처 적합성 검증 규칙 (Architecture Fitness Functions)

모든 프로젝트는 아래 5대 아키텍처 규칙(`ARCH-RULE-01` ~ `ARCH-RULE-05`)을 JUnit 테스트 단계에서 자동 실행하며, 단 1건이라도 위반 시 CI 빌드가 실패(Fail) 처리됨.

규칙 ID	아키텍처 검증 규칙명	검증 대상 패키지	위반 판정 조건 (Violation Condition)	강제 수준
`ARCH-RULE-01`	도메인 계층 순수성 보장	`..domain..`	`org.springframework..` , `jakarta.persistence..` , `com.fasterxml.jackson..` , `..adapter..` 참조 시	**Blocker (CI 차단)**
`ARCH-RULE-02`	헥사고날 단방향 계층 의존성	전 계층 (`adapter` , `application` , `domain`)	`application` 이 `adapter` 를 참조하거나, `adapter.in` 이 `adapter.out` 을 직접 참조 시	**Blocker (CI 차단)**
`ARCH-RULE-03`	트랜잭션 내 외부 I/O 호출 금지	`..application.service..`	`@Transactional` 이 선언된 메서드에서 `..adapter.out.client..` (외부 HTTP/gRPC) 호출 시	**Blocker (CI 차단)**
`ARCH-RULE-04`	JPA 엔티티 외부 유출 금지	`..adapter.out.persistence..`	`@Entity` 가 붙은 클래스가 `persistence` 패키지 외부 (`application` , `web` 등)에서 참조될 시	**Blocker (CI 차단)**
`ARCH-RULE-05`	컨트롤러 서비스 구현체 직접 주입 금지	`..adapter.in.web..`	Controller가 `application.port.in` 인터페이스가 아닌 `application.service` 구체 클래스를 필드로 주입받을 시	**Critical (PR 차단)**

JAVA

```java
package com.company.architecture;

@AnalyzeClasses(packages = "com.company", importOptions = ImportOption.DoNotIncludeTests.clas
public class HexagonalArchitectureFitnessTest {

    @ArchTest
    static final ArchRule ARCH_RULE_01_domain_must_be_pure_pojo =
        noClasses().that().resideInAPackage("..domain..")
            .should().dependOnClassesThat()
            .resideInAnyPackage("org.springframework..", "jakarta.persistence..", "com.faster

    @ArchTest
    static final ArchRule ARCH_RULE_02_hexagonal_layer_isolation =
        layeredArchitecture().consideringOnlyDependenciesInLayers()
            .layer("InboundAdapter").definedBy("..adapter.in..")
            .layer("OutboundAdapter").definedBy("..adapter.out..")
            .layer("Application").definedBy("..application..")
            .layer("Domain").definedBy("..domain..")
            .whereLayer("InboundAdapter").mayNotBeAccessedByAnyLayer()
            .whereLayer("OutboundAdapter").mayNotBeAccessedByAnyLayer()
            .whereLayer("Application").mayOnlyBeAccessedByLayers("InboundAdapter")
            .whereLayer("Domain").mayOnlyBeAccessedByLayers("Application", "OutboundAdapter")
}
```

7.2 전사 표준 기술 스택 명세서 (Standard Technology BOM)

기술 분류	표준 채택 기술 (Artifact / Engine)	표준 버전 (BOM)	아키텍처 역할 및 필수 설정 규격
Language & JVM	Eclipse Temurin OpenJDK	**Java 21 LTS**	Virtual Threads(`JEP 444`), ZGC Generational(`-XX:+UseZGC -XX:+ZGenerational`)
Core Framework	Spring Boot / Spring Framework	**3.3.x / 6.1.x**	`spring.threads.virtual.enabled=true` , `ProblemDetail(RFC 9457)`
Persistence (Command/Query)	Hibernate ORM + QueryDSL / jOOQ	**6.5.x / 5.1.x**	`open-in-view=false` , `provider_disables_autocommit=true` , `batch_size=100`
Connection Pool & DB Migration	HikariCP + Flyway	**5.1.x / 10.x**	`hikari.auto-commit=false` , `ddl-auto=validate` , 스키마 버전 형상 관리
Cache & Distributed Lock	Caffeine (L1) + Redisson / Lettuce (L2)	**3.1.x / 3.30.x**	Look-aside 다계층 캐싱, Pub/Sub 기반 로컬 캐시 무효화, 분산 락
Messaging & Event Bus	Spring for Apache Kafka (KRaft)	**3.2.x (Kafka 3.7)**	`Transactional Outbox` 릴레이, `acks=all` , `enable.idempotence=true`
Resilience & HTTP Client	Resilience4j + Spring `RestClient`	**2.2.x**	CircuitBreaker, SemaphoreBulkhead, TimeLimiter, Exponential Backoff Retry
Observability & Governance	OpenTelemetry Agent + Micrometer + ArchUnit	**2.4.x / 1.13.x / 1.3.x**	W3C TraceContext 전파, Prometheus 메트릭, 빌드 시점 아키텍처 규칙 강제