4. Architecture Design & Evaluation

4.1. Candidate Designs per QA

4.1.1. Candidate Design List

QA	QAS	Candidate Design	Candidate Design Approach (CDA)
QA1: *Performance*	QAS-02 QAS-05	QA1_CD-01	QA1_CD-01_CDA-01: 멀티프로세스 + MQ + 캐시 + DB 복제 **QA1_CD-01_CDA-02**: GSLB + 다중 서버 + 캐시 + 다중 DB 복제 QA1_CD-01_CDA-03: LB + 다중 서버 + DB 샤딩
QA2: *Maintainability*	QAS-03	QA2_CD-01	QA2_CD-01_CDA-01: 단일 모니터링 서버 QA2_CD-01_CDA-02: 분산 모니터링 서버 **QA2_CD-01_CDA-03**: 에이전트 기반 모니터링
QA3: *Reliability*	QAS-06 QAS-07	QA3_CD-01	**QA3_CD-01_CDA-01**: 다중 서버 + DB Active-Active + 모니터링 QA3_CD-01_CDA-02: 다중 서버 + DB Active-Passive + 모니터링 QA3_CD-01_CDA-03: 다중 서버 + DB 샤딩 + 모니터링
QA4: *Usability*	QAS-04	QA4_CD-01	QA4_CD-01_CDA-01: MVP 패턴 + Undo **QA4_CD-01_CDA-02**: MVVM 패턴 + Undo QA4_CD-01_CDA-03: MVI 패턴 + Undo
QA5: *Security*	QAS-01	QA5_CD-01	QA5_CD-01_CDA-01: 망분리 + 암호화 + 접근 차단 + 로깅 QA5_CD-01_CDA-02: 허가 목록 + 암호화 + 접근 차단 + 로깅 **QA5_CD-01_CDA-03**: 분산 + 암호화 + 접근 차단 + 로깅

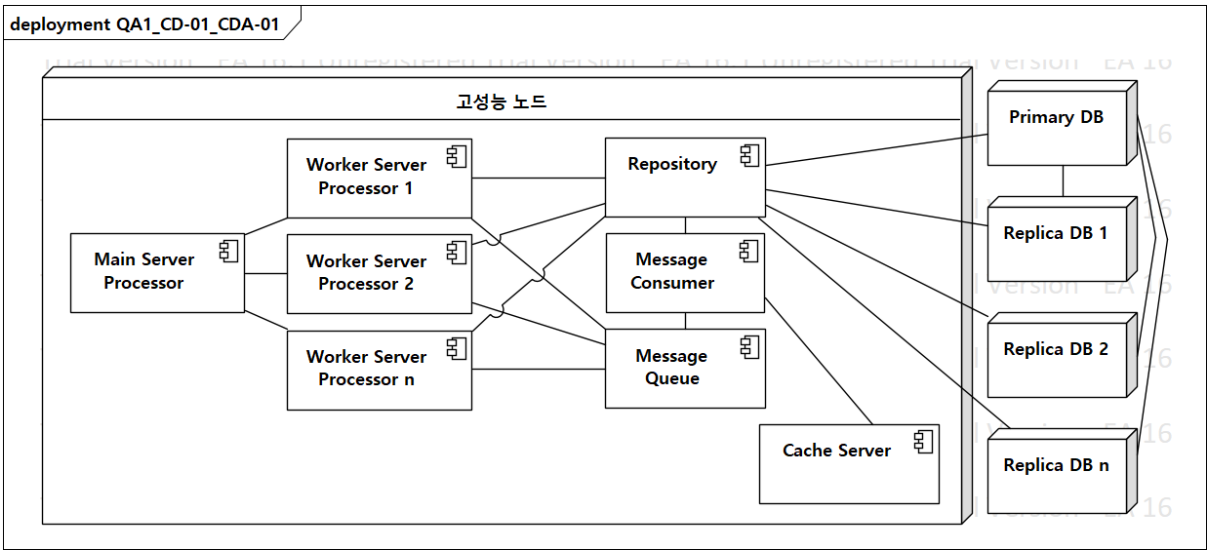
4.1.2. QA1: Performance

4.1.2.1. Design Goal

시스템은 대량의 단말기 접속을 받아 거래 내역을 수집하고 각각 마일리지 적립 등의 추가 작업을 해야 하며 이 데이터 기반의 이용 통계도 제공해야 한다. 계속해서 쏟아지는 거래 내역 데이터를 누락 없이 빠르게 처리하고 저장해야 하며 다양한 조건의 통계 조회 또한 일관성 있는 데이터 기반으로 전체 시스템 영향 없이 이뤄져야 한다. 단일 서버 안에서 데이터 처리 성능을 개선하거나 서버를 복제하거나 고성능 데이터 저장소를 사용할 수 있을 것이다.

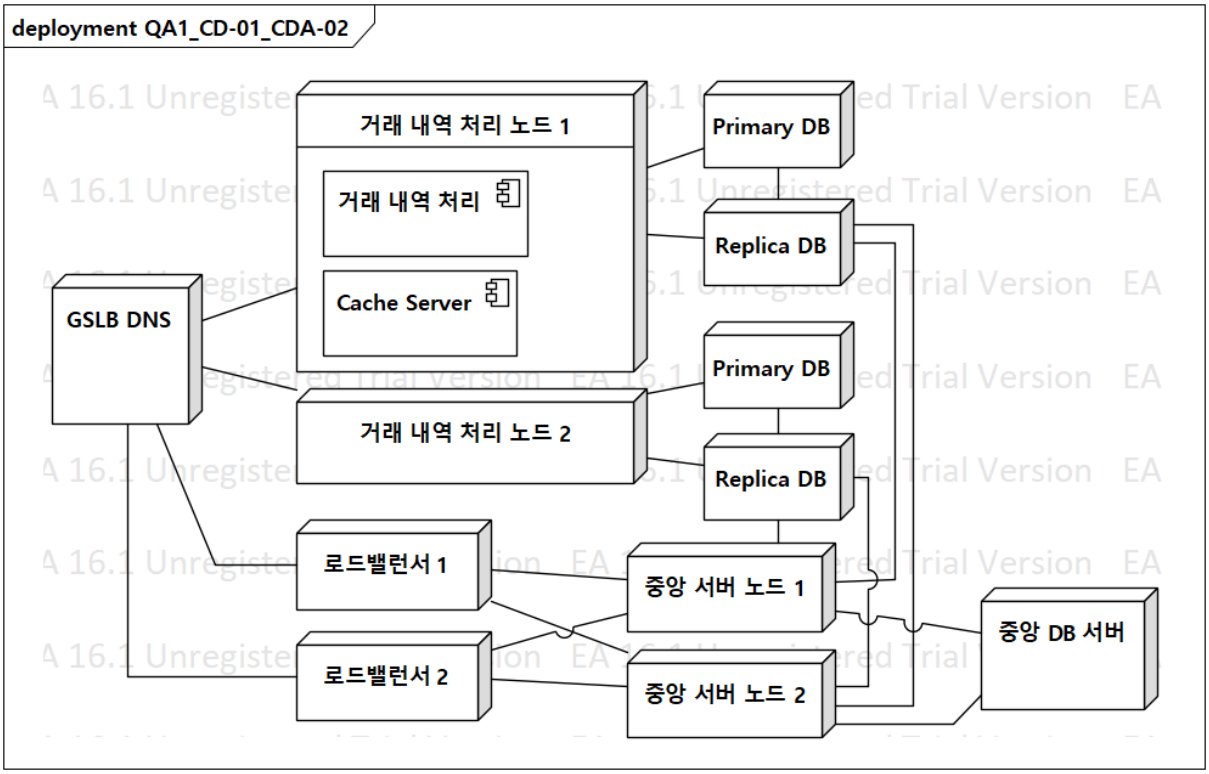
4.1.2.2. Candidate Design Approach List

4.1.2.3. CDA #1 Description: 멀티프로세스 + MQ + 캐시 + DB 복제



CDA ID	*QA1_CD-01_CDA-01*
Description	고성능 단일 노드에 먼저 처음 요청을 받는 메인 프로세스가 하나 있으며 워커 프로세스들로 이를 분산시킨다. 워커 프로세스에서 해당 요청을 실질적으로 처리한다. 여기서 거래 내역을 DB에 저장하고 바로 응답을 주게 되는데 마일리지 적립 등의 추가적인 처리가 이뤄질 수 있도록 메시지 큐(MQ)에 관련 정보를 담아 게시한다. 그 다음 메시지 큐의 거래 내역 토픽을 구독하고 있던 또 다른 프로세스에서 마일리지 적립 처리를 수행한다. 이때 마일리지를 적립할 계정이 존재하는지 및 마일리지 잔액 확인이 필요한데 매번 DB를 조회하는 대신 캐시 서버를 사용한다. 마일리지 갱신 또한 DB에 바로 쓰는 대신 캐시 서버에 써 놓고 DB에 한번에 동기화한다. DB는 Primary 및 n개의 Replica로 Replication 구성한다. 쓰기 작업은 Primary에 하고 추후 Replica로 복제되도록 하며 통계 조회시 읽기 작업은 Replica로 분산시킨다.
Pros	트래픽 처리 능력 향상. 추가 작업을 비동기적으로 처리하여 응답 시간 단축. 캐시 및 DB 복제 활용한 데이터 읽기 쓰기 성능 최적화.
Cons	관리 요소 많아져 운영 및 유지보수 복잡. 캐시 서버와 DB 간의 데이터 일관성 신경 써야 함. 메시지 큐, 캐시 서버 장애 발생시 시스템 전체 영향.

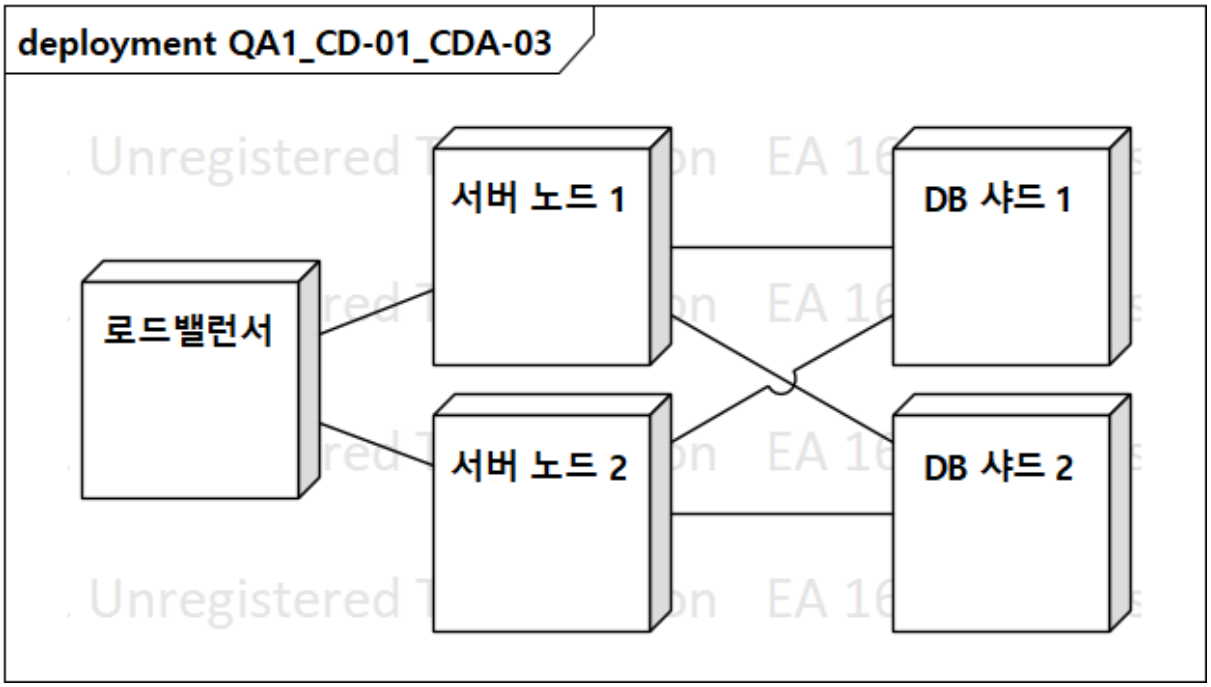
4.1.2.4. CDA #2 Description: GSLB + 다중 서버 + 캐시 + 다중 DB 복제



CDA ID	*QA1_CD-01_CDA-02*
Description	거래 내역 처리용 서버를 지리적으로 분산시켜 배치하고 GSLB(Global Server Load Balancing)를 적용하여 단말기의 요청을 위치, 서버 상태 등을 고려하여 빠르게 처리 가능한 쪽의 서버 IP로 연결되게 한다. 대량의 거래 내역이 들어올 때 우선 지리적으로 가까운 곳의 서버에서 요청을 처리할 수 있으며 가깝더라도 서버 부하가 심한 경우 좀 더 먼 곳에서 신속히 처리될 수 있게 한다. 각 서버는 받은 거래 내역에 대해 따로따로 마일리지 적립을 위한 처리를 한다. 소비자 계정이 존재하는지, 적립률 설정은 무엇인지 중앙 DB에서 조회해야 하는데 이때 각 서버 내 캐시를 두어 속도를 향상시킨다. 그리고 거래 내역은 각자 독립적(스키마 동일)으로 가지고 있는 DB에 저장하며 이 DB는 Primary-Replica 구성한다. 마일리지 적립은 캐시 서버에서 계정별 적립 금액을 누적하여 관리하다 주기적으로 중앙 서버에 Bulk로 보내서 DB에 갱신한다. 거래 내역의 통계 조회 요청은 우선 중앙 서버로 들어가지만 거래 내역은 지리적으로 분산되어 있는 각 DB들에 있으므로 병렬로 Replica에서 조회하여 응답한다. 그 외의 일반 서비스(카드 관리, 카드 사전 등록 등)들도 중앙 서버에서 처리하며 중앙 서버 또한 LB 뒤에 다중으로 구성한다. Candidate Design Evaluation 이후 수정사항: 중앙 서버 앞에 로드밸런서를 Active-Active 이중화하고 GSLB로 로드밸런서를 선택하게 하여 LB가 단일 장애 지점이 되는 걸 피한다. 각 서버는 미리 약속된 IP, Port, Protocol만 접속을 허용하여 공격 표면을 줄인다.
Pros	대량의 트래픽을 지리적으로 분산시켜 부하 분산 및 빠른 응답. 한 소비자가 지리적으로 멀지 않은 지점서 카드를 사용하니 캐시 효율성 높음. DB를 병렬로 운영하고 Primary-Replica 구성하여 통계 조회 부하 분산 및 속도 향상.

	한 서버가 다운되어도 다른 지역 서버로 연결되어 가용성 높음.
Cons	물리적으로 거리가 있는 여러 서버 관리 시간적, 비용적 부담. 거래 내역 처리용 서버, 중앙 서버 따로 구성 필요하여 복잡함. 분산된 캐시와 중앙 DB간의 데이터 일관성 신경 써야 함.

4.1.2.5. CDA #3 Description: LB + 다중 서버 + DB 샤딩



CDA ID	*QA1_CD-01_CDA-03*
Description	요청을 처리하는 물리 서버(노드)를 여러 대 운영하고 로드밸런서(LB)를 통해 트래픽을 분산하여 받는다. DB는 Shared-nothing architecture를 적용하여 샤딩(Sharding)한다. LB가 분배해준 거래 내역을 각 운영 서버에서 DB에 기록하는데 이때 해당 거래 내역의 카드 번호 및 카드 관리 시 카드 번호와 같은 고유 ID가 같으면 항상 동일한 샤드를 사용한다. 그러면 한 카드 거래 내역에 대해 마일리지 적립을 위한 카드 적립률 설정을 조회하기 위해서 전체 샤드에서 조회할 필요가 없다. 그렇게 한 샤드에서만 조회하여 얻은 정보로 마일리지 적립을 수행하는데 계정 정보 또한 고유 ID 등으로 샤드를 예측할 수 있게 해 두어 바로 DB에 쓸 수 있도록 한다. 통계는 한 소비자가 요청할 경우 등록된 카드별로 역시 한 샤드에서만 조회하면 되며 전체 통계가 필요한 제휴업체나 국토교통부 요청의 경우에는 전체 샤드에서 병렬로 조회할 수 있다.
Pros	로드밸런서로 트래픽을 분산시켜 처리 능력 향상. 여러 DB 서버를 병렬로 사용하여 부하 낮추고 전체 처리량 및 응답 속도 증가. 일부 운영 서버가 다운되어도 계속 서비스 가능.
Cons	여러 물리 서버를 관리해야 하는 복잡성.

	DB 샤딩 구현 및 최적화 복잡성.

4.1.2.6. Decision and Rationale

Performance		**Analysis**	**CDA #1** *멀티프로세스 + MQ + 캐시 + DB 복제*	**CDA #2 (Selected)** *GSLB + 다중 서버 + 캐시 + 다중 DB 복제*	**CDA #3** *LB + 다중 서버 + DB 샤딩*
ID	**Title**				
QAS-02	신속한 통계 조회	Pros	(+) *각 Replica로 읽기분산 시켜 빠른 처리* (+) *각 프로세스로 요청 분산 처리*	(++) *DB 병렬 처리 및 Replica 부하 분산으로 조회 시 빠름* (+) *각 서버로 트래픽 분산 처리*	(+) DB *병렬 처리로 전체 조회 시 빠름* (+) *각 서버로 트래픽 분산 처리*
		Cons	(-) *DB 복제 비용*	(-) *DB 복제 비용*	(-) *샤딩 구현 복잡성*
QAS-05	대량의 단말기 접속 처리	Pros	(+) *트래픽 분산 및 비동기 처리* (+) *캐시로 속도 향상*	(++) *지리적 부하 분산* (+) *캐시 효율성*	(++) *각 운영 서버 및 DB 샤드로 분산 처리*
		Cons	(-) *관리 요소 증가로 유지보수 복잡함* (-) *MQ, 캐시 장애 발생시 전체 영향*	(-) *여러 물리적 서버 관리 및 구성 부담* (-) *캐시와 중앙 DB 일관성 구현 복잡*	(-) *여러 물리적 서버 관리 및 구성 부담* (-) *샤딩 구현 복잡성*

Candidate Design:

QA	**QAS**	**CD**	**Description**
QA1: Performance	QAS-02 QAS-05	QA1_CD-01 *(GSLB + 다중 서버 + 캐시 + 다중 DB 복제)*	이 시스템은 대량의 단말기 접속을 받아 거래 내역을 지속적으로 수집 및 신속히 처리하고 추후 이 데이터를 기반으로 통계 조회도 제공해야 한다. GSLB를 활용해 거래 내역 처리 서버를 지리적으로 분산시키면 각 단말기에 가장 가까운 서버에서 요청을 처리할 수 있어 지연 시간을 줄일 수 있다. 여러 서버를 사용하므로 전체 트래픽 처리량 또한 높게 얻을 수 있다. 서버 상태도 고려하기 때문에 로드밸런싱 효과를 더 높이고 고가용성도 확보할 수 있다. 각 서버에는 캐시를 두어 중앙 DB에 대한 조회를 최소화한다. 거래 내역은 각자 가지고 있는 DB에만 저장하며 중앙 DB로의 쓰기 작업도 캐시에서 하다가 동기화하는 전략을 사용하여 성능을 확보한다. 소비자별 최근 거래 내역 분포가 지리적인 특징을 가지므로 캐시 효율성 또한 높다. 중앙 서버 또한 LB 뒤에 다중으로 구성하여 요청을

			분산 받고 모든 DB는 Primary-Replica 구성한다. 통계 조회는 분산된 각 DB(Replica)에서 병렬로 실행되어 처리량이 높다. CDA1은 프로세스 관리에 의존하여 지리적 분산의 이점을 얻을 수 없고 MQ의 관리가 복잡하며 병목 지점이 될 수 있다. CDA3은 지리적 분산의 이점을 얻을 수 없고 DB 샤딩 구현이 더 복잡하고 추후 확장이 어렵다. Candidate Design Evaluation 이후 수정사항: 중앙 서버 앞에 로드밸런서를 Active-Active 이중화하고 GSLB로 로드밸런서를 선택하게 하여 LB가 단일 장애 지점이 되는 걸 피한다. 각 서버는 미리 약속된 IP, Port, Protocol만 접속을 허용하여 공격 표면을 줄인다.

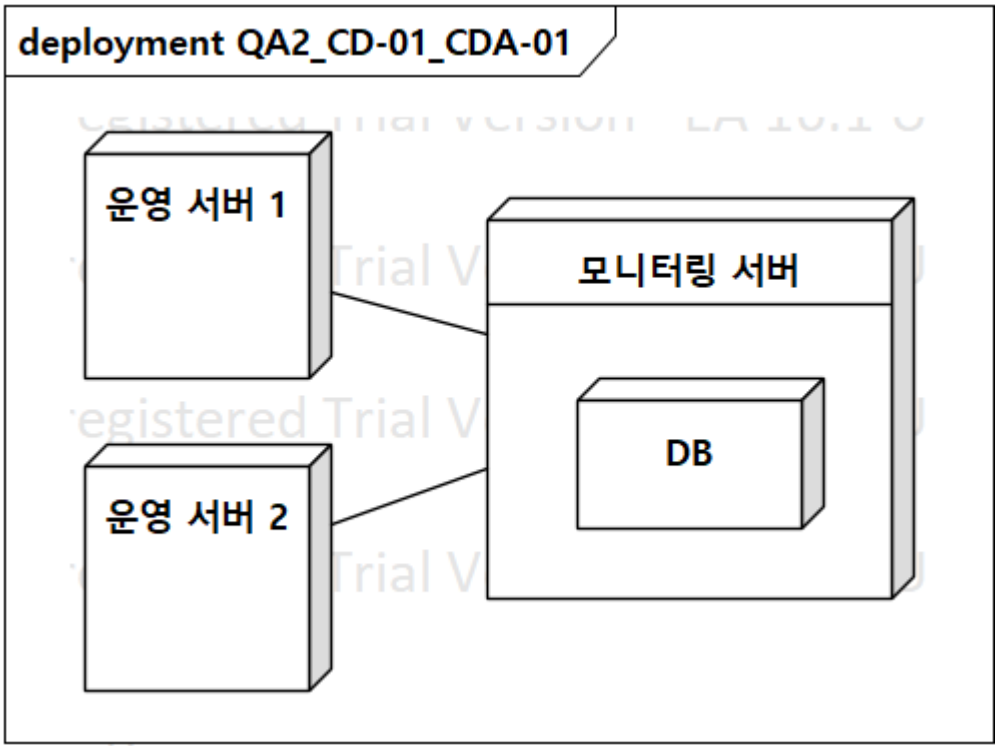
4.1.3. QA2: Maintainability

4.1.3.1. Design Goal

시스템은 동작 로그를 기록하고 네트워크 트래픽, CPU 사용률, 메모리 사용량 등의 서버 상태도 수집하여 오류 발생시 관리 시스템에서 조회할 수 있게 해준다. 개발 부서에서 문제 지점을 빠르게 파악하고 원인 분석할 수 있도록 도움으로써 시스템 가동 중단 시간을 최소화하고 전반적인 유지보수성을 향상시킬 수 있다. 어떻게 하면 시스템의 다른 기능에 영향이 없도록 데이터를 수집 및 처리하면서도 최대한 유용한 정보를 제공할 수 있을지 고려한다.

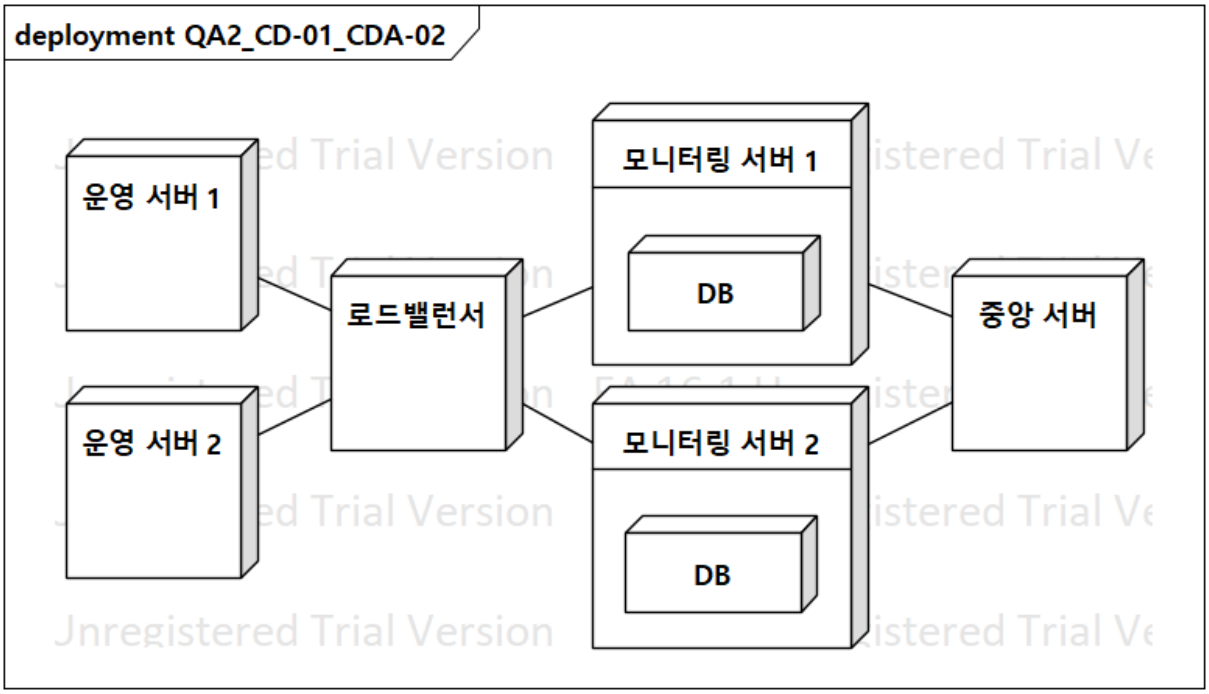
4.1.3.2. Candidate Design Approach List

4.1.3.3. CDA #1 Description: 단일 모니터링 서버



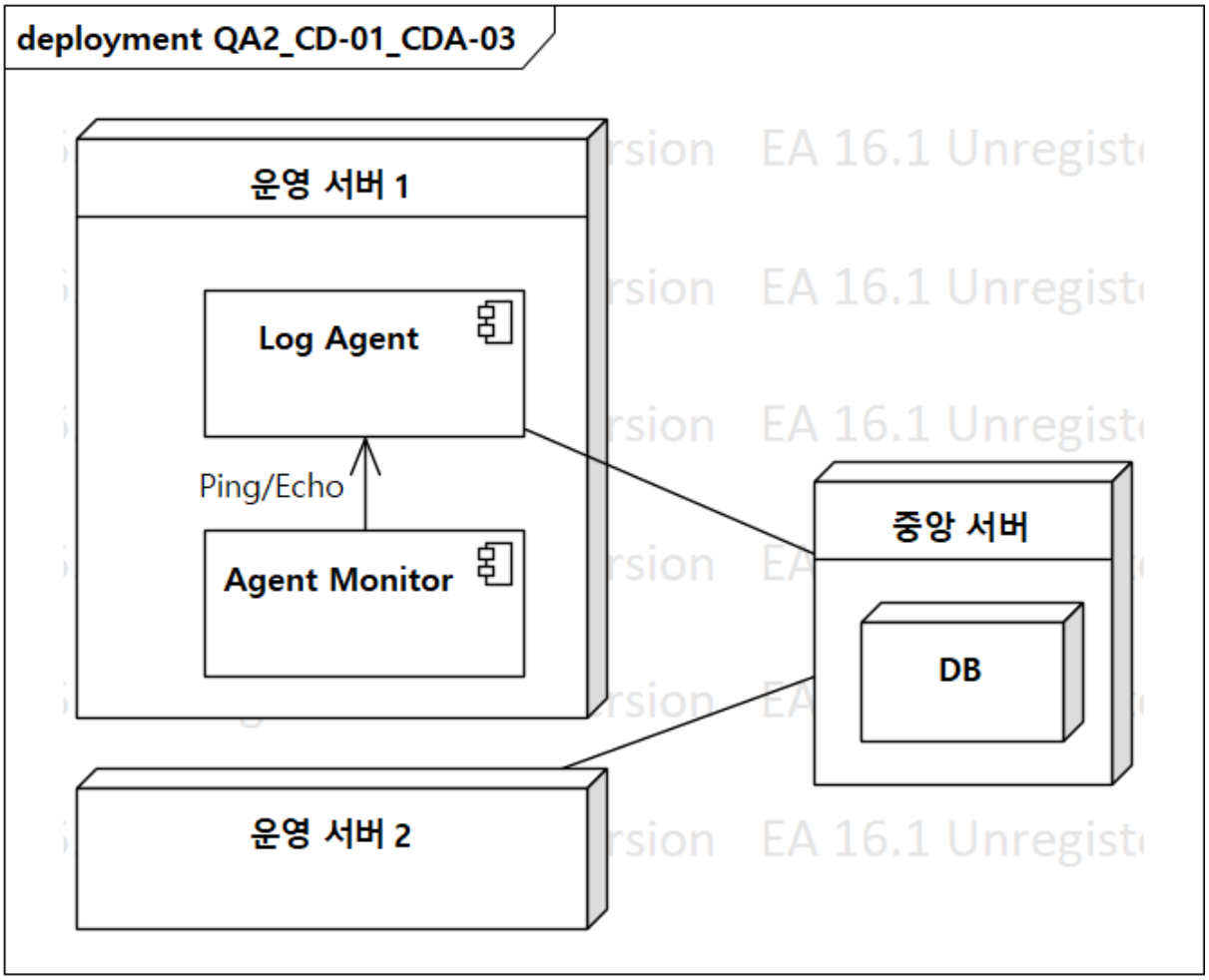
CDA ID	*QA2_CD-01_CDA-01*
Description	감시 대상 서버의 로그와 상태를 하나의 중앙 모니터링 서버로 전송하여 저장한다. 각 서버에서 프로세스, DB 등의 상태와 발생 로그를 수집하여 중앙 모니터링 서버로 계속 전송해주면 이걸 저장 및 인덱싱해둔다. 이후 관리 시스템에서 중앙 모니터링 서버에 접근하여 로그와 상태를 조회한다.
Pros	한 곳에서 통합된 데이터 조회. 중앙화 된 보안 및 접근 제어.
Cons	중앙 서버가 단일실패지점. 시스템이 커지면 데이터 수집 및 처리 병목. 각 서버에서 데이터 전송 구현.

4.1.3.4. CDA #2 Description: 분산 모니터링 서버



CDA ID	*QA2_CD-01_CDA-02*
Description	감시 대상 서버의 로그와 상태를 여러 모니터링 서버 중 하나에 분산시켜 보내 처리한다. 각 모니터링 서버는 데이터를 저장해 두고 나중에 중앙 서버로 보내 통합 인덱싱을 수행한다. 이후 관리 시스템에서 중앙 서버에서 조회하려고 하면 이 인덱스를 활용하여 필요한 데이터를 빠르게 찾고 각 분산된 서버에서 가져와 통합하여 보내준다. 감시 대상 서버의 배치 형태에 따라 모니터링 서버를 단순히 여러 대 두기만 하거나 아예 지리적으로 분산시키거나 할 수 있다.
Pros	데이터 수집 및 처리 부하 분산. 모니터링 서버 확장성 확보. 일부 모니터링 서버 장애가 발생해도 부분적으로 동작.
Cons	여러 모니터링 서버를 설정 및 관리해야 하므로 복잡성과 비용 높음. 데이터 일관성 신경 써야 함. 각 서버에서 데이터 전송 구현.

4.1.3.5. CDA #3 Description: 에이전트 기반 모니터링



CDA ID	*QA2_CD-01_CDA-03*
Description	각 감시 대상 서버에 별도 에이전트 프로그램을 설치하여 자체적으로 로그와 상태를 수집하고 처리한 후 중앙 서버로 전송한다. 에이전트는 로그 필터링을 수행하거나 전송 전 압축하는 등 추가적인 작업을 할 수 있다. 중앙 서버 장애로 전송하지 못한 데이터는 보관하여 나중에 다시 전송한다. 중앙 서버는 수집된 로그를 인덱싱하여 저장하고 관리 시스템에서 접속하여 로그와 상태를 조회한다. Candidate Design Evaluation 이후 수정사항: 에이전트는 서버 자원 현황을 모니터링하여 일정 부하 이하인 상태에서만 동작하도록 하고 부하가 오래 지속되는 경우 해당 내용만 따로 수집하여 전송한다. 에이전트 프로세스를 Ping/Echo로 모니터링하다가 장애 탐지되면 자동으로 에이전트 재실행하는 서비스 추가하여 가용성을 확보한다. 에이전트를 실행하는 유저 권한은 필요 최소한으로 설정하여 보안성을 확보한다.
Pros	데이터 처리 부하 분산. 로컬에서 데이터 전처리/필터링 등 세밀한 로깅 제어. 각 서버에 데이터 전송 구현 없이 에이전트만 설치.
Cons	각 서버에 에이전트 설치 및 관리 부담. 에이전트가 서버 리소스 사용.

4.1.3.6. Decision and Rationale

Maintainability		**Analysis**	**CDA #1** *단일 모니터링 서버*	**CDA #2** *분산 모니터링 서버*	**CDA #3 (Selected)** *에이전트 기반 모니터링*
ID	**Title**				
QAS-03	오류 발생시 신속한 분석	Pros	(++) *하나의 모니터링 서버만 관리* (+) *모든 로그가 한 곳에 통합됨*	(+) *데이터 처리 분산* (++) *가용성 및 확장성 높음*	(+) *데이터 처리 분산* (++) *세밀한 로깅 제어* (+) *에이전트 설치만 하면 구성 완료*
		Cons	(-) *중앙 서버가 단일 실패지점* (-) *데이터 처리 병목*	(-) *여러 서버 구성 복잡성 및 비용 증가* (-) *데이터 일관성 처리 필요*	(-) *에이전트 관리 부담* (-) *호스트 리소스 사용*

Candidate Design:

QA	**QAS**	**CD**	**Description**
QA2: Maintainability	QAS-03	QA2_CD-01 (*에이전트 기반 모니터링*)	이 시스템은 문제 발생시 원인 지점을 빠르게 파악할 수 있을 수준으로 유용한 로그를 제공해야 한다. 또한 이런 모니터링 시스템이 다른 서비스에 영향을 주지 않아야 한다. 각 서버에 로그 전송 기능을 따로 구현할 필요 없이 로깅 에이전트를 설치해두기만 하면 약속된 위치의 로그 파일을 처리하여 상태와 함께 중앙 서버로 전송한다. 에이전트 설정을 통해 세밀하게 로깅 동작(필터링, 전처리)을 변경할 수 있으며 이러한 계산은 분산된 각 서버에서 이뤄지므로 부하가 적다. CDA1은 시스템이 커져서 로그 처리에 병목이 발생할 때 서버 수직 확장이 어렵고 단일실패지점이 될 수 있다. CDA2는 여러 모니터링 서버를 관리해야 하는 비용 부담이 크고 분산된 데이터를 통합 조회하기 위해 복잡한 처리가 필요하다. Candidate Design Evaluation 이후 수정사항: 에이전트는 서버 자원 현황을 모니터링하여 일정 부하 이하인 상태에서만 동작하도록 하고 부하가 오래

			지속되는 경우 해당 내용만 따로 수집하여 전송한다. 에이전트 프로세스를 Ping/Echo로 모니터링하다가 장애 탐지되면 자동으로 에이전트 재실행하는 서비스 추가하여 가용성을 확보한다. 에이전트를 실행하는 유저 권한은 필요 최소한으로 설정하여 보안성을 확보한다.

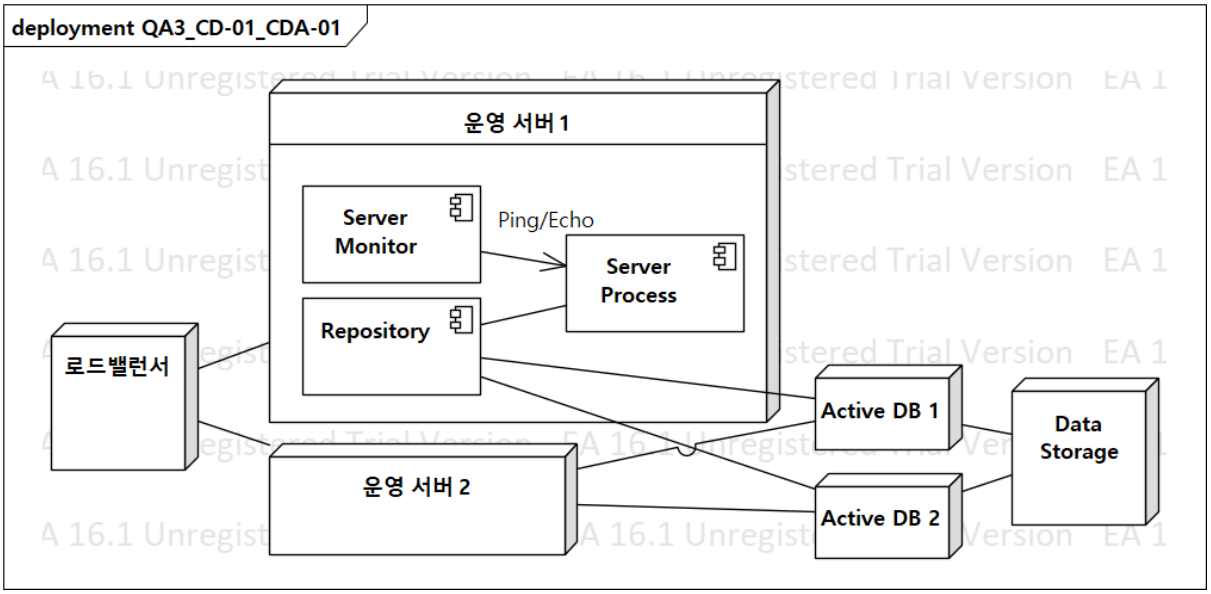
4.1.4. QA3: Reliability

4.1.4.1. Design Goal

시스템은 대량의 단말기로부터 들어오는 할인 정책 요청에 빠짐없이 안정적으로 응답해야 한다. 또한 카드 관리 서비스를 제공하는 서버 일부가 다운되었더라도 계속 해당 서비스를 제공할 수 있어야 하며 다운된 서버는 탐지하여 복구해야 한다. 두 사항을 만족시키기 위해서 서버와 DB를 분산하는 방법, 장애를 탐지하는 방법을 중점적으로 고려하여 높은 가용성을 가지는 디자인을 찾는다.

4.1.4.2. Candidate Design Approach List

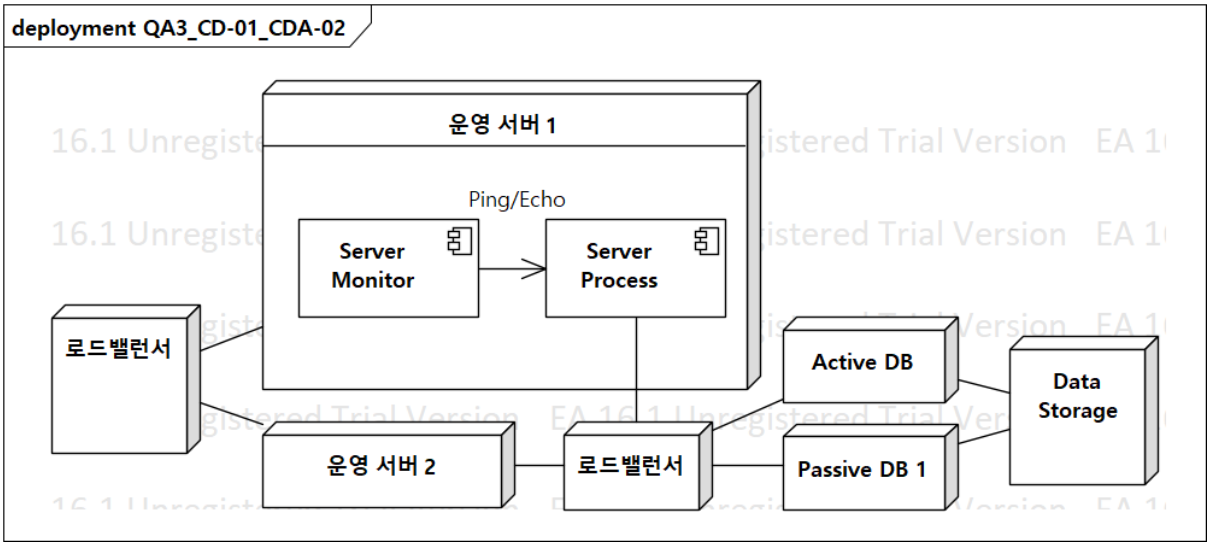
4.1.4.3. CDA #1 Description: 다중 서버 + DB Active-Active + 모니터링



CDA ID	QA3_CD-01_CDA-01

Description	로드밸런서 뒤에 운영 서버를 다중으로 배치하고 DB 또한 Active-Active Redundancy를 적용한다. 먼저 첫번째 로드밸런서에서 정상 운영 서버로 요청을 분산시키면 서버는 DB 접근을 위해 또 다른 로드밸런서로 접근한다. 이후 Active 상태의 DB 중 하나에서 쿼리가 실행되며 공용 데이터 저장소에 작업이 이뤄진다. 한 운영 서버에 장애가 발생하면 트래픽은 자동으로 정상 서버로 분산된다. 또한 서버 프로세스 수준의 문제일 경우엔 내부 Ping/Echo 모니터링 서비스에 의해 프로세스가 자동으로 재시작 된다. 서버 장비 수준의 문제라서 자동 복구가 불가능할 경우에는 Heartbeat 끊김이 별도 모니터링 서버에 의해 탐지되고 기록되어 외부에서 알 수 있도록 한다. 한 DB에 장애가 발생할 경우에는 자동으로 트래픽 분산 대상에서 제외된다. Candidate Design Evaluation 이후 수정사항: DB로의 요청 분산은 로드밸런서가 아니라 서버 어플리케이션 단에서 직접 쿼리를 분산하여 네트워크 통신 지연을 개선한다. 분산되면 추적이 어려운 쿼리들은 한 트랜잭션으로 묶는다. 이런 작업을 자동으로 해주는 컴포넌트를 구현하여 활용한다.
Pros	서버, DB 부하 분산. 서버 자동 복구. 서버, DB 확장 용이. 서버, DB Failover.
Cons	DB 동시 Active 비용. 저장소 병목현상 발생. 데이터 동시성 문제 발생.

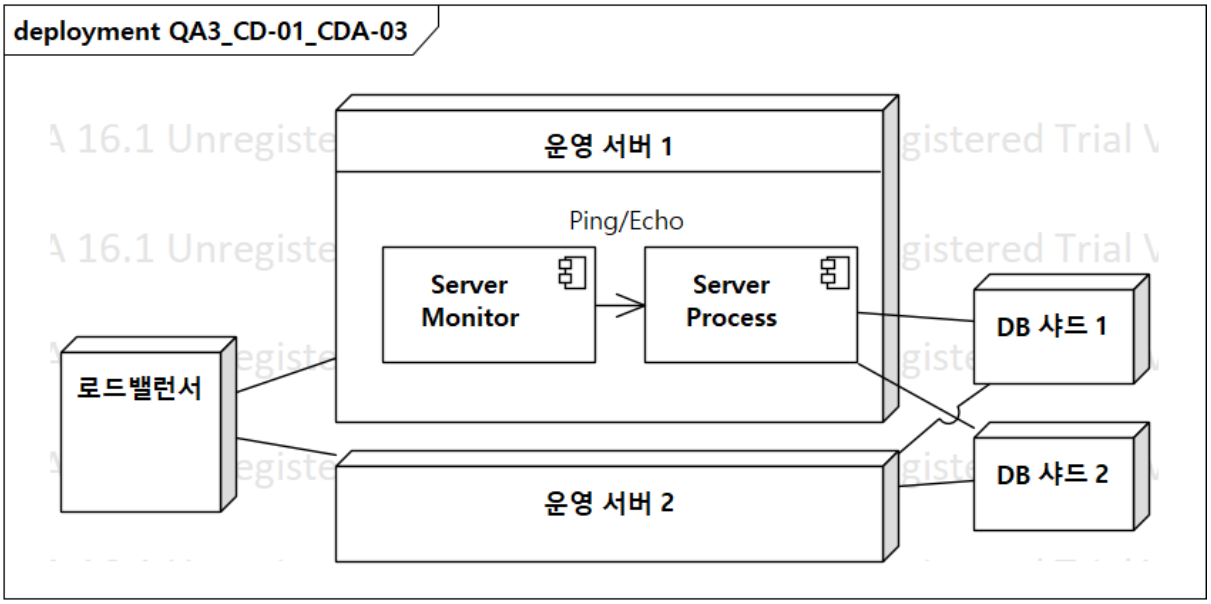
4.1.4.4. CDA #2 Description: 다중 서버 + DB Active-Passive + 모니터링



CDA ID	*QA3_CD-01_CDA-02*
Description	(이전 CDA1의 서버 다중화, 모니터링 부분은 동일하다.) DB는 Active-Passive(Hot) Redundancy를 적용한다. 한 Active DB에서 쿼리가 실행되어 공

	용 데이터 저장소에 작업이 이뤄지며 나머지는 실행 중이지만 아무것도 하지 않는다. Active DB에 장애가 발생할 경우 Passive DB가 Active로 전환되며 로드밸런서가 트래픽을 전환시킨다.
Pros	서버 부하 분산. 서버 자동 복구. 서버 확장 용이. 서버, DB Failover.
Cons	DB Hot Standby 비용. DB 부하 분산 불가.

4.1.4.5. CDA #3 Description: 다중 서버 + DB 샤딩 + 모니터링



CDA ID	*QA3_CD-01_CDA-03*
Description	(이전 CDA1의 서버 다중화, 모니터링 부분은 동일하다.) DB가 각 다중 서버용으로 따로 존재하는 형태로 샤딩을 적용하며 중복 데이터를 일부 허용한다. 한 운영 서버로 들어온 요청은 쓰기 작업의 경우엔 해당 서버용 DB 샤드에만 작업이 이뤄진다. 읽기 작업의 경우엔 다른 서버들의 DB에 각각 병렬로 수행하여 취합한다. 다만 고가용성이 필요한 일부 데이터의 경우엔 각 DB들에 중복 저장되도록 약속하여 전체 DB를 조회할 필요가 없도록 한다. 예를 들어 한 운영 서버에서 할인 정책 배포가 수행되면 다른 운영 서버 DB에도 할인 정책이 중복되어 저장되며 추후 단말기가 할인 정책을 조회하려고 하면 로드밸런싱 된 서버의 DB에서만 조회하면 된다. 한 운영 서버의 DB에서 장애가 발생하면 같이 있던 운영 서버 프로세스는 오류 로그를 남기고 Reconfiguration을 통해 타 운영 서버의 DB를 사용한다.
Pros	서버, DB 부하 분산. 서버 자동 복구. 서버, DB Failover.

	DB 로드밸런서 불필요.
Cons	데이터 일관성 등 샤딩 구현 복잡함. 전체 조회시 중복 데이터 처리 필요. 서버, DB 확장 어려움.

4.1.4.6. Decision and Rationale

Reliability		**Analysis**	**CDA #1 (Selected)** *다중 서버 + DB Active-Active + 모니터링*	**CDA #2** *다중 서버 + DB Active-Passive + 모니터링*	**CDA #3** *다중 서버 + DB 샤딩 + 모니터링*
ID	**Title**				
QAS-06	안정적인 할인 정책 배포	Pros	(++) *서버, DB 부하 분산 및 Failover* (+) *서버, DB 확장성*	(+) *서버 부하 분산* (+) *서버, DB Failover* (+) *서버 확장성*	(++) *서버, DB 부하 분산 및 Failover*
		Cons	(-) *데이터 동시성 문제*	(-) *DB 부하*	(-) *서버, DB 확장 어려움* (-) *데이터 일관성 문제*
QAS-07	언제나 가능한 카드 관리	Pros	(++) *서버, DB 부하 분산 및 Failover* (+) *다운된 서버 복구*	(+) *서버 부하 분산* (+) *서버, DB Failover* (+) *다운된 서버 복구*	(++) *서버, DB 부하 분산 및 Failover* (+) *다운된 서버 복구*
		Cons	(-) *데이터 동시성 문제*	(-) *DB 부하*	(-) *데이터 일관성 문제*

Candidate Design:

QA	**QAS**	**CD**	**Description**
QA3: Reliability	QAS-06 QAS-07	QA3_CD-01 *(다중 서버 + DB Active-Active + 모니터링)*	시스템은 대량의 단말기 요청에 안정적으로 응답해야 한다. 또한 일부 카드 관리 서비스에 장애가 발생하여도 계속 해당 서비스를 제공해야 하며 다운된 서비스는 탐지하여 복구해야 한다. 서버를 다중화 하고 DB는 Active-Active 구성한 후 각각 로드밸런서로 부하를 분산시키며 일부가 죽더라도 중단 없이 계속해서 서비스를 제공할 수 있다. 또한 장애 상황을 탐지하여 가능한 경우 자동 복구를 시도하고 그렇지 않을 경우 외부에 알릴 수 있도록 로그를 남긴다. 장애 상황은 크게 프로세스 수준과 장비 수준으로 나누어서 전자의 경우엔 내부

			Ping/Echo 모니터링 서비스에 의해 자동 복구가 시도되며 후자의 경우엔 외부 Heartbeat 모니터링 서버에 의해 오류 로그가 남는다. 이 디자인은 또한 서버, DB 추가 확장이 비교적 용이하며 추가 가용성 확보가 필요할 때 도움이 된다. CDA2는 DB의 부하 분산이 이뤄지지 않고 확장이 비교적 어렵다. CDA3은 서버, DB의 확장이 비교적 어렵다. Candidate Design Evaluation 이후 수정사항: DB로의 요청 분산은 로드밸런서가 아니라 서버 어플리케이션 단에서 직접 쿼리를 분산하여 네트워크 통신 지연을 개선한다. 분산되면 추적이 어려운 쿼리들은 한 트랜잭션으로 묶는다. 이런 작업을 자동으로 해주는 컴포넌트를 구현하여 활용한다.

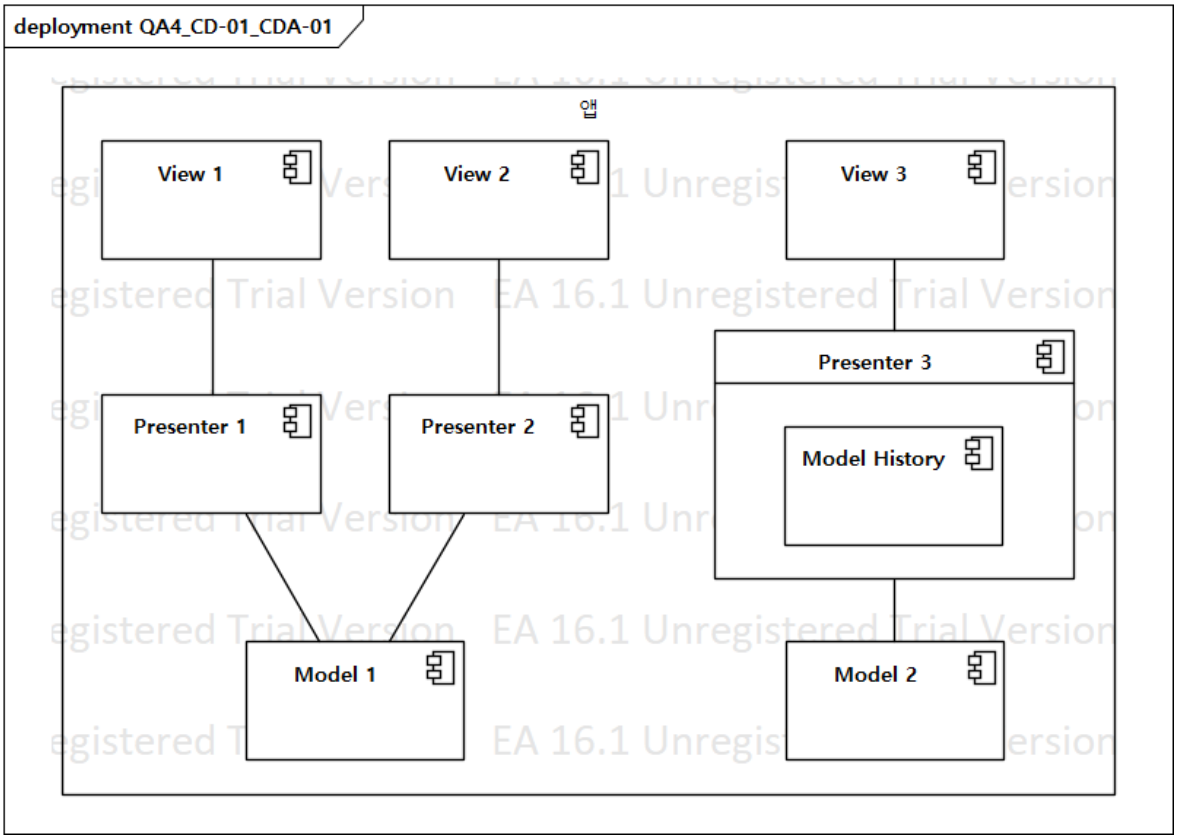
4.1.5. QA4: Usability

4.1.5.1. Design Goal

앱은 기본적으로 사용하기 쉬워야 한다. 구체적으로는 소비자가 별도 매뉴얼을 공부하지 않아도 사용할 수 있어야 한다. 그러기 위한 방법 중 하나로 잘못된 입력값이나 조작에 대해 사용자 오류를 방지하고 즉각적이고 이해하기 쉬운 피드백을 제공한다. 소비자는 돌이킬 수 없는 오류에 대한 걱정 없이 점진적으로 올바른 앱 사용법을 익힐 수 있다. 여러 View에서 다양한 사용자 오류를 방지하고 피드백을 표시하도록 개발하기 효과적인 설계를 고려한다.

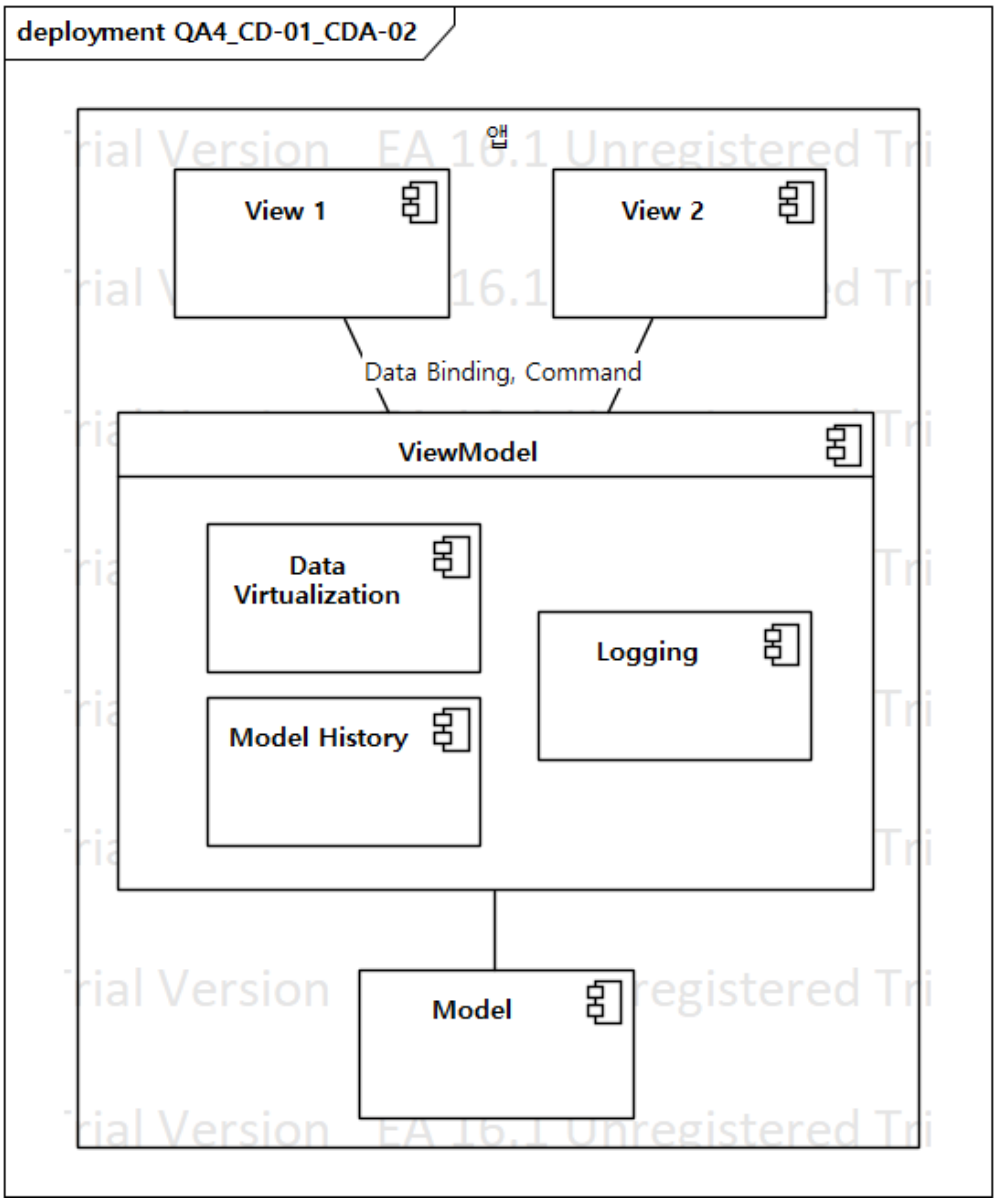
4.1.5.2. Candidate Design Approach List

4.1.5.3. CDA #1 Description: MVP 패턴 + Undo



CDA ID	*QA4_CD-01_CDA-01*
Description	MVP 패턴은 Model, View, Presenter로 구성되며 Presenter가 Model, View 사이를 중재하여 Model과 View간 종속성을 제거한다. View에서 잘못된 입력값이나 조작이 발생하면 Presenter에서 Model에 반영하기 전에 검사하여 오류를 방지하고 다시 View로 피드백을 제공할 수 있다. 또한 Model을 변경하기 전에 변경점을 기록해두거나 백업하여 Undo 수행시 다시 상태를 복원할 수 있다.
Pros	View, Model간 의존성이 없음. 상태 관리가 직관적이고 테스트 용이.
Cons	Presenter에 비즈니스 로직, UI 로직이 다 들어가 비대해짐. Presenter, View간 의존성이 강함.

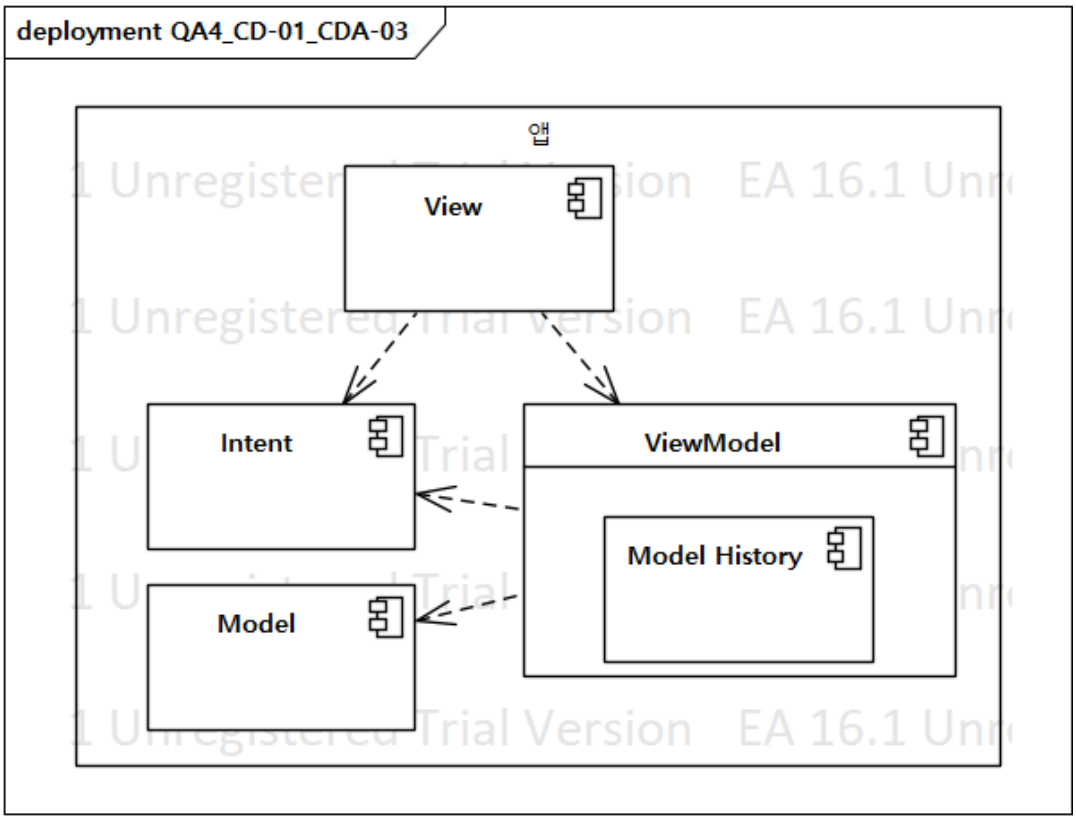
4.1.5.4. CDA #2 Description: MVVM 패턴 + Undo



CDA ID	*QA4_CD-01_CDA-02*
Description	MVVM 패턴은 Model, View, ViewModel로 구성되며 ViewModel이 Model, View 사이에서 데이터 바인딩과 커맨드 패턴을 통해 상호작용을 처리한다. 같은 ViewModel을 사용하는 여러 View가 있을 수 있고 ViewModel은 View에 대해 몰라도 된다. View에서 잘못된 입력값이나 조작이 발생하면 ViewModel에서 Model에 반영하기 전에 오류를 방지하고 피드백이 View에 즉시 표시되게 할 수 있다. 또한 변경점을 기록해두거나 백업하여 Undo 수행시 다시 상태를 복원할 수 있다. Candidate Design Evaluation 이후 수정사항: 통계 조회 화면 등 뷰에 연결되는 데이터가 많은 경우에는 데이터 가상화(화면에 보이는 부분만 바인딩) 기술을 사용하여 성능을 최적화한다. 데이터 바인딩 부분의 디버깅을 돕기 위해서 데이터 읽기/쓰기 로그가 남도록 한다.
Pros	View, Model간 의존성이 없고 ViewModel도 View에 의존성이 없음. 데이터 바인딩을 통해 간결하고 신속하게 View 갱신.

	ViewModel 하나에 여러 View 활용 가능.
Cons	ViewModel이 비대해질 수 있음. 데이터 바인딩 디버깅이 어려움.

4.1.5.5. CDA #3 Description: MVI 패턴 + Undo



CDA ID	*QA4_CD-01_CDA-03*
Description	MVI 패턴은 Model, View, Intent로 구성되며 View에서 발생한 Intent가 Model을 업데이트하면 그것이 다시 View에 반영되는 단방향 데이터 흐름을 가진다. 이때 Model은 불변성을 가져서 상태 변화는 새로운 Model이 생성됨을 의미한다. View에서 잘못된 입력값이나 조작에 해당하는 Intent가 발생하면 Model에 반영하지 않고 대신 피드백 정보를 Model에 반영하여 View에 표시되도록 할 수 있다. Intent를 기록하거나 불변 Model을 백업해두면 Undo 수행 시 다시 상태를 복원할 수 있다.
Pros	데이터 흐름과 상태 변화 추적이 쉬움. 불변 상태를 이용하여 동시성 문제 적음.
Cons	비교적 구현 양이 많고 복잡함. 간단한 View 변경도 무조건 Intent를 생성해서 이뤄져야 함.

4.1.5.6. Decision and Rationale

Usability		**Analysis**	**CDA #1** *MVP 패턴 + Undo*	**CDA #2 (selected)** *MVVM 패턴 + Undo*	**CDA #3** *MVI 패턴 + Undo*
ID	**Title**				
QAS-04	즉각적인 입력 오류 피드백	Pros	(+) *View-Model 의존성 없음* (+) *데이터 흐름이 명확함*	(+) *View-Model 및 ViewModel->View 의존성 없음* (+) *간결하고 신속한 View 갱신* (+) *ViewModel 하나에 여러 View 활용*	(+) *View-Model 의존성 없음* (+) *데이터 흐름과 상태 변화가 명확함* (+) 동시성 문제 적음
		Cons	(-) *Presenter가 비대해지면 개발 어려움*	(-) *ViewModel이 비대해지면 개발 어려움* (-) *데이터 흐름 디버깅이 어려움*	(-) *구현 양이 많고 복잡함* (-) *간단한 UI 표시도 Intent를 사용해야 함*

Candidate Design:

QA	**QAS**	**CD**	**Description**
QA4: Usability	QAS-04	QA4_CD-01 *(MVVM 패턴 + Undo)*	소비자가 앱 사용 중 돌이킬 수 없는 오류에 대해 걱정하지 않고 사용하기 쉽도록 다양한 View에서 사용자 오류를 방지하고 적절한 피드백을 제공해야 한다. View, Model 사이에 ViewModel이 위치하여 둘의 의존성을 제거해주고 View를 독립적으로 개발할 수 있게 해준다. ViewModel은 중간에서 소비자의 입력값 오류나 오조작을 Model에 반영하기 전 적절히 처리할 수 있고 View에도 신속히 피드백을 띄울 수 있다. View의 갱신은 ViewModel과의 데이터 바인딩을 통해 이뤄지므로 직접적인 View 조작 없이도 실시간으로 변경이 반영된다. 따라서 즉각적으로 사용자에게 피드백을 제공하기 더 쉽다. 그리고 여러 View에서 동일한 ViewModel을 사용할 수 있어서 이런 사용자 오류 방지 및 피드백 코드를 동일하게 재활용하여 개발 효율성을 높이고 일관적인 사용자 경험을 제공할 수 있다. CDA1은 Presenter와 View의 의존성이 강하고 Presenter가 비대해질 수 있으며 직접적인 View 조작이 번거롭다. CDA3은 모든 사용자 입력을 Intent로 변환해야 하고 그에 따른 상태 관리 구현이 복잡하고 과하다.

			Candidate Design Evaluation 이후 수정사항: 통계 조회 화면 등 뷰에 연결되는 데이터가 많은 경우에는 데이터 가상화(화면에 보이는 부분만 바인딩) 기술을 사용하여 성능을 최적화한다. 데이터 바인딩 부분의 디버깅을 돕기 위해서 데이터 읽기/쓰기 로그가 남도록 한다.

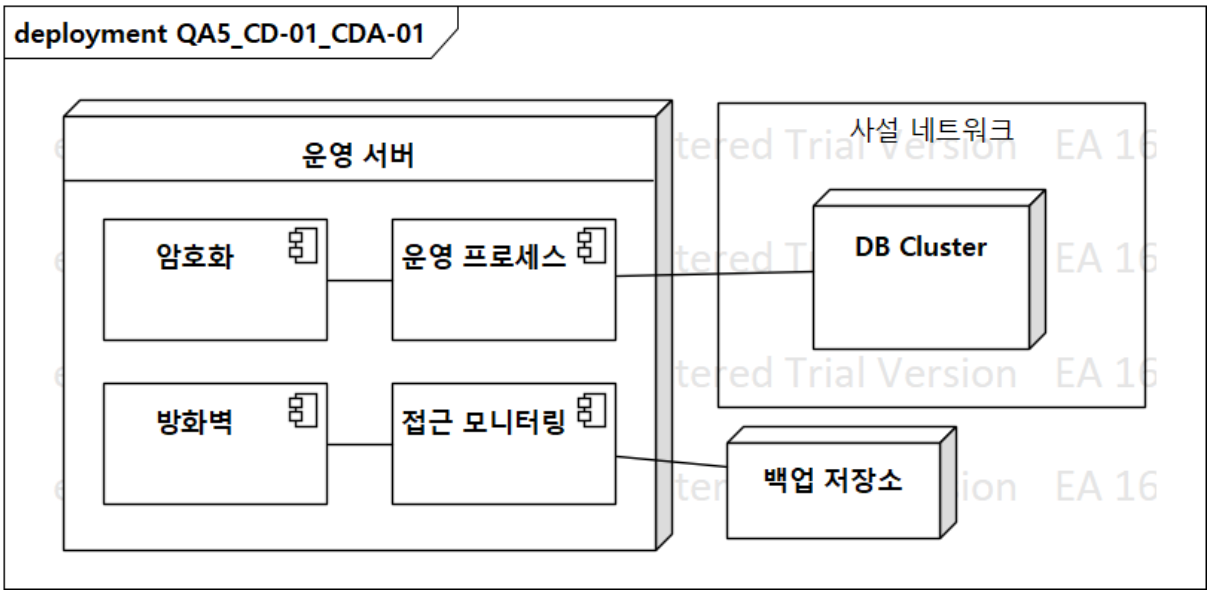
4.1.6. QA5: Security

4.1.6.1. Design Goal

시스템은 소비자 계정 정보(예: 이메일, 비밀번호)와 거래 내역(예: 결제 카드, 가맹점, 사용 금액)을 관리하는데 이러한 데이터들이 그대로 DB에 담긴 상태로 유출될 시 사회에 피해를 입힐 수 있고 규제기관의 철퇴를 맞을 수 있기에 주의해야 한다. 비정상적인 DB 접근은 최대한 차단되어야 하며 유출이 발생하여도 식별 가능한 개인 정보는 포함되어 있지 않아야 한다. 또한 유출 경위와 범위를 파악하여 사후 조치할 수 있도록 기록을 항상 남겨야 한다. 이러한 부분들을 만족하는 디자인을 찾는다.

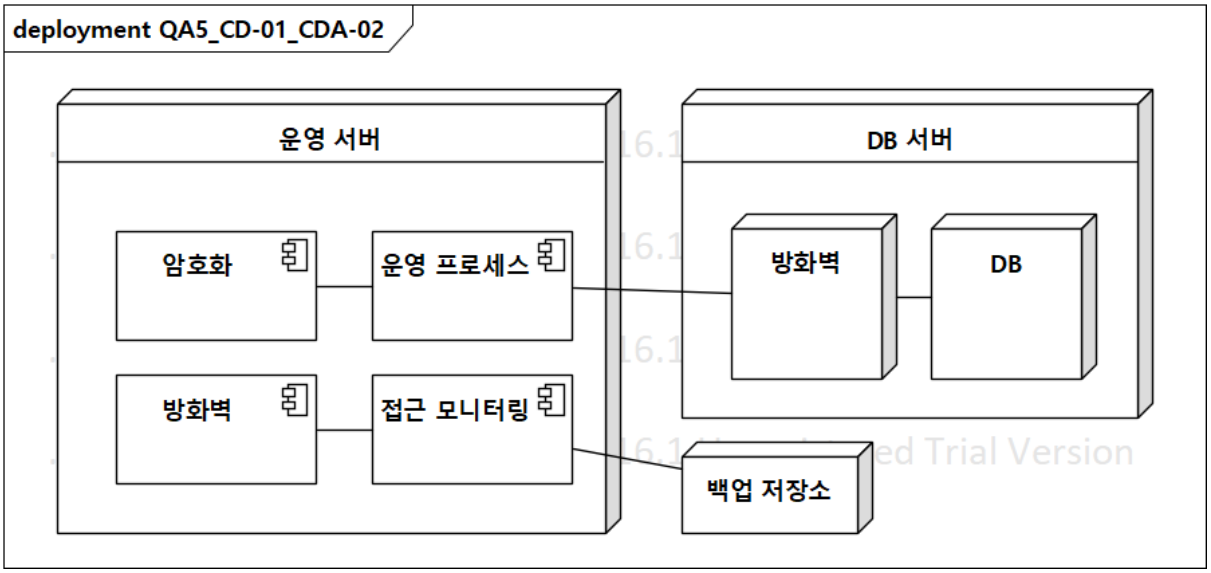
4.1.6.2. Candidate Design Approach List

4.1.6.3. CDA #1 Description: 망분리 + 암호화 + 접근 차단 + 로깅



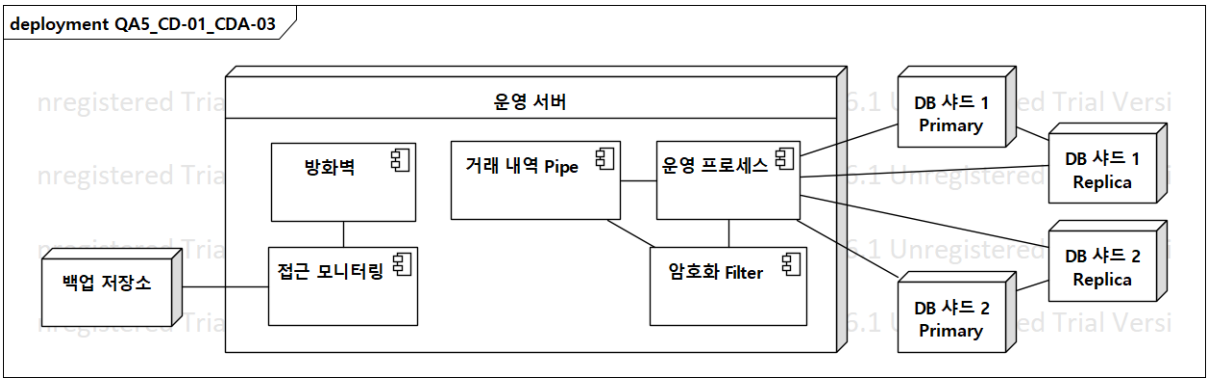
CDA ID	*QA5_CD-01_CDA-01*
Description	DB 서버를 별도 네트워크에 분리 배치하여 공개 인터넷으로 접근할 수 없게 Limit Exposure를 적용한다. 데이터는 암호화하여 저장 및 전송하며 지속적으로 실패하는 서버 접근 시도는 자동으로 차단한다. 모든 접근 기록과 요청은 로그에 남기고 주기적으로 다른 저장소에 백업하여 사후 분석할 수 있도록 한다.
Pros	공격 대상을 찾고 접근하는 것부터 어렵게 함. 데이터가 유출되더라도 암호화되어 식별할 수 없음. 무차별 대입 공격 자동 차단. 로그를 통해 사후 분석 지원.
Cons	망분리 시스템 운영, 유지보수 복잡성. 망분리로 통신 지연 발생.

4.1.6.4. CDA #2 Description: 허가 목록 + 암호화 + 접근 차단 + 로깅



CDA ID	*QA5_CD-01_CDA-02*
Description	DB 서버에 접근할 수 있는 IP, 포트, 프로토콜을 최소한으로 방화벽에 설정하여 미리 약속된 곳에서만 접근할 수 있게 Limit access를 적용한다. 데이터는 암호화하여 저장 및 전송하며 허가된 대상이라도 지속적으로 실패하는 접근 시도는 자동으로 차단한다. 모든 접근 기록과 요청은 로그에 남기고 주기적으로 다른 저장소에 백업하여 사후 분석할 수 있도록 한다.
Pros	사전 허가된 대상 외엔 접근 기본 차단. 데이터가 유출되더라도 암호화되어 식별할 수 없음. 무차별 대입 공격 자동 차단. 로그를 통해 사후 분석 지원.
Cons	허가 목록을 지속적으로 갱신해야 하는 부담. 허가 목록이 잘못 갱신될 경우 서비스 장애 발생.

4.1.6.5. CDA #3 Description: 분산 + 암호화 + 접근 차단 + 로깅



CDA ID	*QA5_CD-01_CDA-03*
Description	Separate entities를 적용하여 DB를 샤딩하고 민감한 데이터를 분산 저장함으로써 전체 데이터가 유출될 위험을 줄인다. 데이터는 암호화하여 저장 및 전송하며 각 샤드 서버별로 지속적으로 실패하는 접근 시도는 자동으로 차단한다. 각각의 접근 기록과 요청은 로그에 남기고 주기적으로 다른 저장소에 백업하여 사후 분석할 수 있도록 한다. Candidate Design Evaluation 이후 수정사항: 계속 들어오는 거래 내역의 경우 암호화 작업으로 성능이 하락하는 것을 방지하기 위해 Pipe&Filter 구조를 적용하여 비동기적으로 처리한다 한 DB 샤드가 단일 실패 지점이 되지 않도록 각 샤드도 Primary-Replica 구성하고 Primary 장애 발생 시 Replica가 승격되도록 한다.
Pros	분산되어 찾기 어렵게 함. 전체 데이터가 한번에 유출될 위험을 줄이고 대응 시간을 범. 데이터가 유출되더라도 암호화되어 식별할 수 없음. 무차별 대입 공격 자동 차단. 로그를 통해 사후 분석 지원.
Cons	샤딩 구현 및 유지보수 복잡성.

4.1.6.6. Decision and Rationale

Security		**Analysis**	**CDA #1** *망분리 + 암호화 + 접근 차단 + 로깅*	**CDA #2** *허가 목록 + 암호화 + 접근 차단 + 로깅*	**CDA #3 (Selected)** *분산 + 암호화 + 접근 차단 + 로깅*
ID	**Title**				
QAS-01	개인정보 유출 방지	Pros	(+) *DB 서버를 찾기 어렵게 함* (+) *데이터 암호화* (+) *무차별 대입 차단* (+) *사후분석 지원*	(+) *허가된 대상 외 기본 차단* (+) *데이터 암호화* (+) *무차별 대입 차단* (+) *사후분석 지원*	(++) *분산시켜 찾기 어렵게 하고 전체 데이터 유출 위험 낮춤* (+) *데이터 암호화* (+) *무차별 대입 차단* (+) *사후분석 지원*

		Cons	(-) *네트워크 운영, 유지보수 복잡함* (-) *통신 지연 발생*	(-) *허가 목록 지속적 갱신 부담* (-) *허가 목록 잘못 설정시 장애 발생*	(-) *샤딩 구현 및 유지보수 복잡함*

Candidate Design:

QA	**QAS**	**CD**	**Description**
QA5: Security	QAS-01	QA5_CD-01 *(분산 + 암호화 + 접근 차단 + 로깅)*	이 시스템은 소비자의 개인정보를 활용하므로 해당 데이터가 유출되지 않도록 보안 조치를 적용하고 유출되어도 식별 가능한 개인정보가 없도록 해야 한다. 또한 사후 분석도 지원해야 한다. 지속적으로 실패하는 DB 서버 접근은 자동으로 탐지하여 차단하도록 방화벽을 설정한다. 무차별 대입 공격을 방지할 수 있다. 개인을 직간접적으로 식별할 수 있는 데이터를 저장하거나 송수신할 때 항상 암호화한다. DB가 유출되더라도 식별 가능한 개인정보는 없을 것이며 피해를 최소한으로 줄일 수 있다. 모든 접근 기록과 요청은 로그에 기록해야 하며 변조를 방지하기 위해 주기적으로 별도 저장소에 백업한다. 사후 분석을 통해 유출 경로를 파악하여 조치할 수 있고 유출 범위를 알아내 피해 정도를 확인할 수 있다. Separate entities를 적용하여 DB를 샤딩하고 데이터를 분산 저장한다. 분산된 샤드 서버들을 찾기 어렵게 하고 각각을 따로 공략해야 하게 만듦으로써 유출 범위를 제한하고 공격 속도를 늦춘다. CDA1은 망분리 구성과 운영이 어렵고 통신 지연이 발생할 수 있다. CDA2는 허가 목록을 지속적으로 갱신해야 하는 부담이 있고 잘못 갱신될 경우 서비스 장애가 발생한다. Candidate Design Evaluation 이후 수정사항: 계속 들어오는 거래 내역의 경우 암호화 작업으로 성능이 하락하는 것을 방지하기 위해 Pipe&Filter 구조를 적용하여 비동기적으로 처리한다 한 DB 샤드가 단일 실패 지점이 되지 않도록 각 샤드도 Primary-Replica 구성하고 Primary 장애 발생 시 Replica가 승격되도록 한다.

4.2. Candidate Designs Evaluation for all QAs

QA	QAS	Analysis	Candidate Design (CD) #1 QA1_CD-01 (GSLB + 다중 서버 + 캐시 + 다중 DB 복제)	Candidate Design (CD) #2 QA2_CD-01 (에이전트 기반 모니터링)	Candidate Design (CD) #3 QA3_CD-01 *(다중 서버 + DB Active-Active + 모니터링)*	Candidate Design (CD) #4 QA4_CD-01 (MVVM 패턴 + Undo)	Candidate Design (CD) #5 QA5_CD-01 (분산 + 암호화 + 접근 차단 + 로깅)
QA1 Performance	QAS-02 QAS-05	Pros	(+) 다중 서버/DB로 대량 트래픽 처리 (+) 지리적으로 가까운 서버에서 빠른 응답 (+) 캐시 활용하여 속도 향상	(+) 로그 전처리를 각 서버의 에이전트가 처리하므로 중앙 서버 부하를 줄임 (+) 로그 전송 전 필터링이나 압축할 수 있어 전송 효율 높음	(+) 서버/DB 트래픽 분산시켜 처리 효율 향상	NA	(+) DB 샤딩을 통해 부하 분산, 병렬 처리
		Cons	NA	(-) 에이전트가 각 서버의 자원을 사용 → 서버의 자원 현황을 모니터링하여 일정 부하 이하 상태에서만 동작하도록 함	(-) 서버 앞, DB 앞에 로드밸런서가 각각 있어 네트워크 속도 저하 발생 → DB 앞의 로드밸런서를 빼고 서버 어플리케이션 단에서 직접 쿼리 분산	(-) 거대한 뷰의 경우 데이터 바인딩의 성능 문제 발생 → 통계 조회 화면의 경우 데이터가 많아 성능 문제 발생할 수 있으므로 데이터 가상화 기술 사용	(-) 암호화 작업이 추가되어 처리 지연 → 회원가입 같은 경우 많지 않아 별도 작업은 필요 없으나 대량 거래 내역의 경우 Pipe&Filter 적용
QA2 Maintainability	QAS-03	Pros	(+) 장애 발생시 문제 발생한 서버로 분석 범위 좁힐 수 있음 (+) 역할별로 서버가 나뉘어 책임 분리	(+) 세밀하게 로깅을 제어하여 로그 분석 시 더 효율적이도록 지원 (+) 로깅을 위해 에이전트만 설치하면 되어 간편	(+) 한 DB 서버를 수정 중에도 다른 DB 서버 계속 동작 (+) 복구 불가능한 문제는 모니터링 서버가 탐지하여 로깅	(+) View, Model 의존성 제거하여 독립적으로 수정 가능	(+) 접근 기록, 요청을 로깅하고 백업하여 추후 분석 가능
		Cons	NA	NA	(-) 한 작업에 대한 여러 쿼리가 다른 DB 서버에서 실행되면 추적 어려움. → 트랜잭션 활용	(-) 데이터 바인딩 중 문제 발생 시 디버깅이 어려움 → 앱 동작 로컬 로깅	(-) DB 관련 기능 구현 시 샤드 선택을 항상 고려해야 함 → 내부적으로 일관성 있는 샤드 선택이 구현된 DB 컴포넌트 구현 및 사용
QA3 Reliability	QAS-06 QAS-07	Pros	(+) 한 서버 다운되어도 Failover 가능 (+) 부하 분산으로 높은 트래픽도 누락 없이 안정적으로 처리	(+) 일시적 장애로 로그를 전송하지 못하여도 에이전트가 적절한 방법으로 추후 재전송 (+) 각 서버별로 에이전트가 있으므로 한 서버가 죽어도 다른 서버의 로그 수집은 영향 없음	(+) 서버/DB 다중화 및 Failover 적용 (+) 서버 장애 시 자동 복구 시도	NA	(+) 한 샤드의 장애가 전체로 퍼지지 않음