ArgMax Mini 설계 문서 - BE 박세현

1. 서론

본 설계 문서는 MLOps 플랫폼 **ArgMax Mini**의 **백엔드 시스템 설계**를 다룹니다. 이 시스템은 사용자가 데이터셋을 업로드하고, 모델을 학습시키며, 학습된 모델로 추론 요청을 수행하는 일련의 워크플로우를 지원합니다. 주요 기술적 과제는 대용량 데이터 처리, 장기 학습 작업의 안정적 처리, 그리고 Kubernetes 환경에서의 효율적인 GPU 자원 관리입니다. 문서는 요구사항 정의로 시작하여 시스템 아키텍처 및 도메인 설계 사항, 핵심 워크플로우별 상세 설계, 운영 정책 및 장애 대응 전략 순으로 구성됩니다.

---

2. 요구사항 정의

2.1 사용자 요구사항 (기능적)

[일반 사용자]

1. **데이터셋 관리 (CRUD):** 사용자는 데이터셋을 **업로드, 조회, 수정, 삭제**할 수 있어야 합니다.
2. **모델 관리 (CRUD):** 사용자는 모델 정보 및 **버전별 관리** 기능을 이용할 수 있어야 합니다.
3. **모델 학습 및 버전 생성:** 사용자의 요청에 따라 모델 학습을 시작하고, 학습 완료 시 새로운 **모델 버전**이 생성 및 저장되어야 합니다.
4. **모델 추론 요청:** 사용자는 특정 모델 버전을 선택하여 입력 데이터를 전달하고, 결과를 **API 형태로 응답**받을 수 있어야 합니다.

[내부 사용자]

1. **원본 데이터 접근:** 모델 개선 및 분석을 위해 고객이 업로드한 **원본 데이터셋**에 접근할 수 있어야 합니다.
2. **추론 로그 조회:** 모델 품질 및 성능 개선을 위해 고객의 **추론 요청/응답 로그**를 조회할 수 있어야 합니다.
3. **대용량 분석 쿼리 실행:** 위의 데이터를 대상으로 **분석/통계 쿼리**를 실행할 수 있어야 합니다.

2.2 기술 요구사항 (비기능적)

1. **대용량 데이터 처리 및 저장:** 최대 **2GB** 크기의 테이블 데이터를 **안정적/효율적으로** 처리하고 저장할 수 있어야 합니다.
2. **장기간 비동기 학습 파이프라인:** 학습이 **12~24시간** 소요되며 안정적으로 처리되어야 합니다.
3. **GPU 자원 관리 및 정책:** Kubernetes 환경에서 GPU 리소스를 효율적으로 할당/회수해야 하며, 장기간 점유(최소 12시간)에 대한 **관리 정책**이 수립되어야 합니다.
4. **유연성 및 안정성 있는 추론 서빙: 모델별로 상이한 입출력 구조**에 유연하게 대응해야 하며, 추론 요청 및 응답 데이터를 **안정적으로 기록**할 수 있는 환경을 갖춰야 합니다.

5. **격리된 내부 분석 환경:** 운영 시스템에 **부하를 주지 않고** 대용량 데이터에 대한 **통계/분석 쿼리 실행 환경** 및 **적절한 데이터 접근 제어**가 제공되어야 합니다.

---

3. 시스템 아키텍처 및 기술 스택

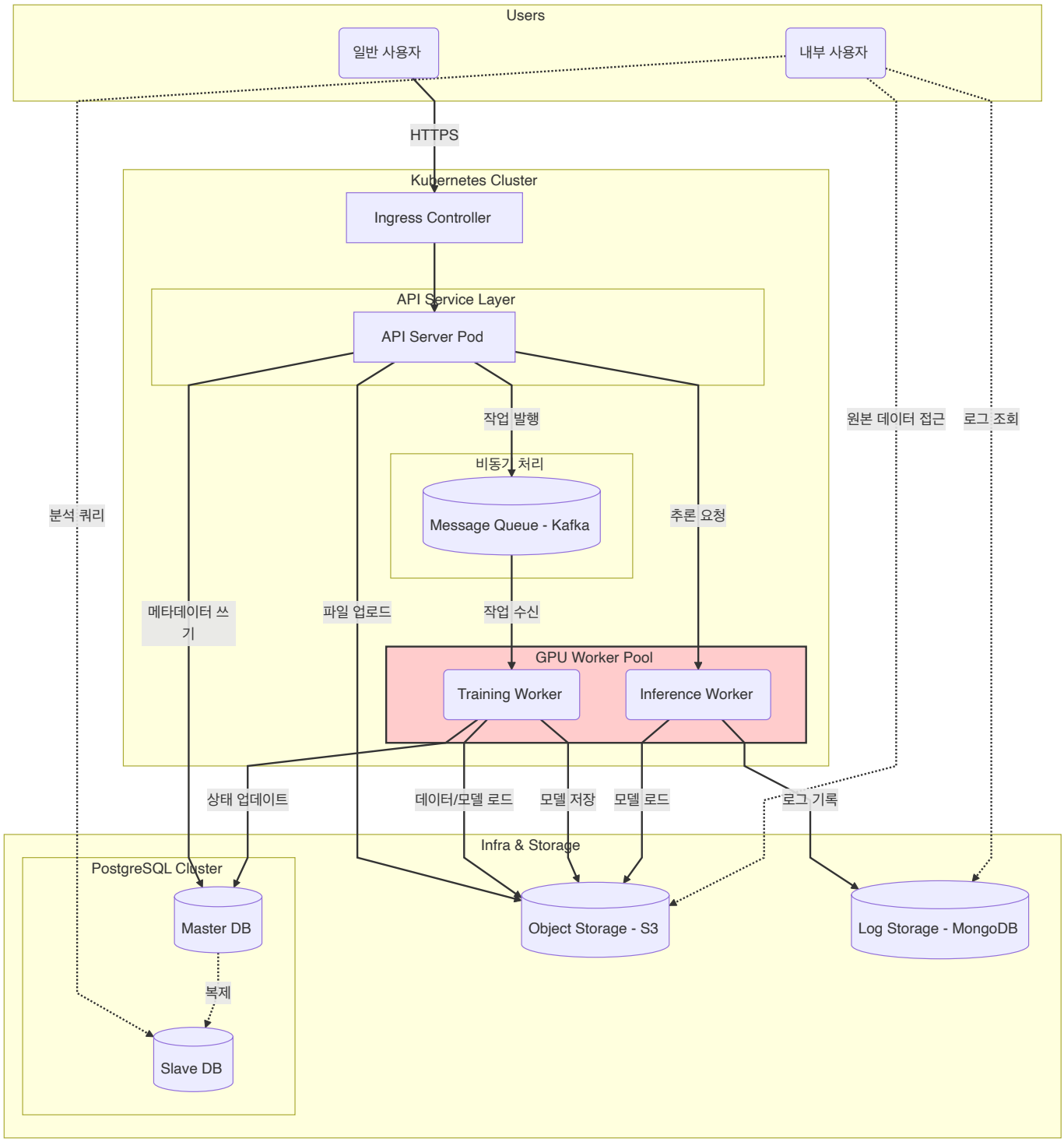
3.1 시스템 개요 및 아키텍처 패턴

시스템은 장기간 학습 처리와 실시간 추론 요청이라는 서로 다른 작업 특성을 고려해 최소한의 서비스 분리를 적용했습니다. 학습 처리(Training Worker)와 추론(Inference Worker)을 API 서버와 분리하여 운영 중 특정 부하가 전체 시스템에 파급되지 않도록 하는 것이 목적입니다.

내부 구조는 레이어드 아키텍처를 사용하되, 도메인 모델링은 복잡도를 낮추기 위해 DDD의 핵심 개념(엔티티 중심 모델링, 도메인 규칙 분리)만 선택적으로 적용합니다.

3.2 시스템 구성도 (Architecture Diagram)

시스템은 **Kubernetes 클러스터** 내의 API 서비스, **Kafka** 메시지 큐, 전용 GPU Worker 풀(Training/Inference)과 외부의 **S3**, **PostgreSQL 클러스터**, **MongoDB**로 구성됩니다.



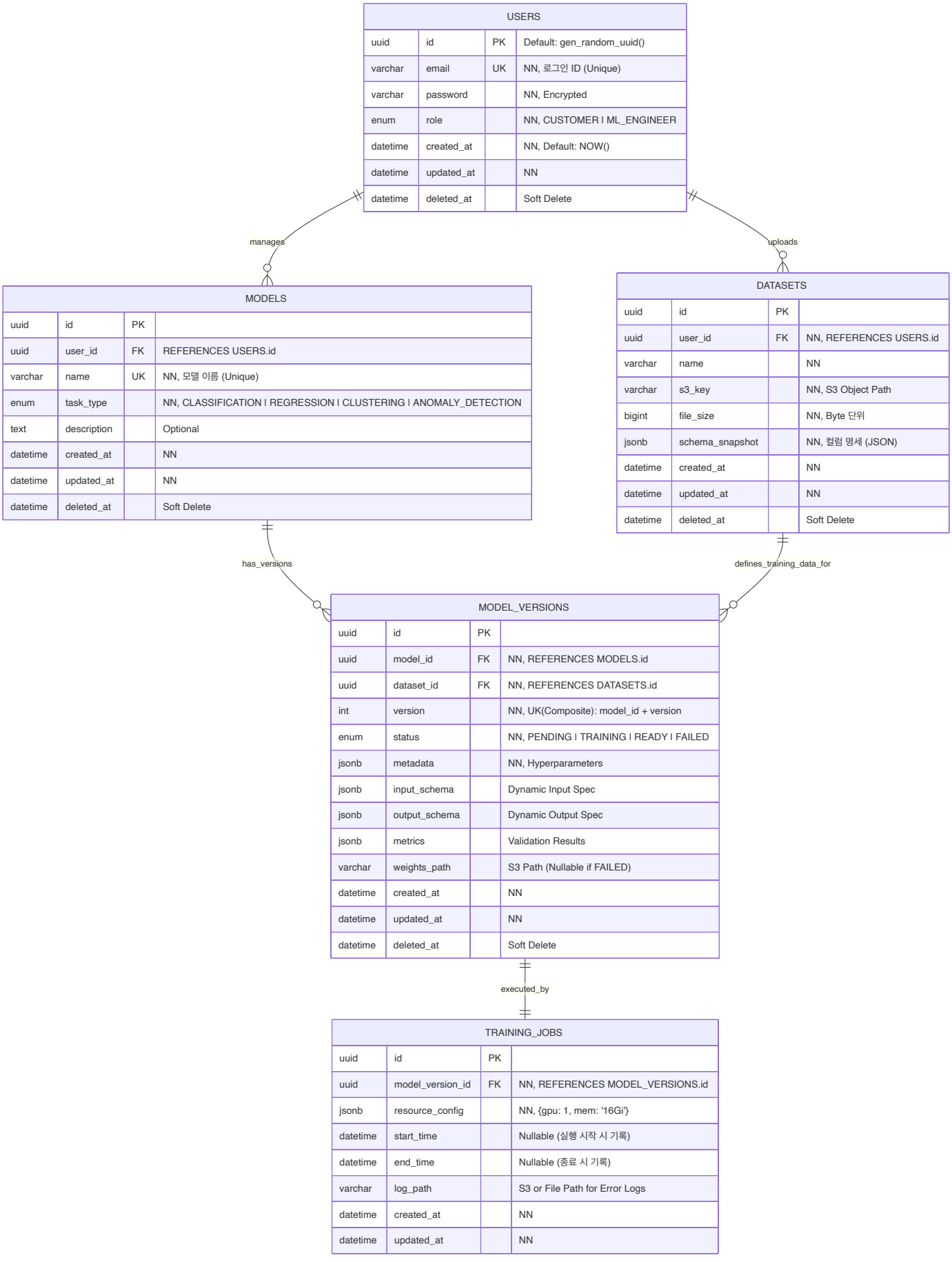
3.3 핵심 컴포넌트 및 기술 스택 선정

요구사항	주요 컴포넌트	선정된 기술 스택	선정 이유 (이슈 및 결정)
자원 관리/ 실행 환경	Worker Orchestration	**Kubernetes (K8s)**	요구사항. GPU 자원 격리(Taint/Toleration) 및 효율적인 할당/회수 (ResourceQuota) 기능 제공. Training/Inference Pod의 자동 확장에 필수적.
메타데이터 관리	RDB	**PostgreSQL**	모델 버저닝 및 상태 관리의 트랜잭션 일관성 보장. JSONB 데이터 타입 지원으로 모델별 유연한 입출력 스키마 처리에 적합.
대용량 파일 저장	Object Storage	**AWS S3**	데이터셋 및 학습 완료 모델 파일(최대 2GB)의 안정적인 저장 및 높은 가용성 확보.
장기간 비동기 학습	Message Queue	**Apache Kafka**	학습 요청이 즉시 처리되지 않아도 안정적으로 전달될 수 있도록 메시지 큐 역할 수행.
추론 로깅	NoSQL DB	**MongoDB**	비정형이고 대용량으로 발생하는 추론 로그를 빠르게 기록하고 유연하게 저장하기에 적합.
API 서비스	Backend Framework	**Spring Boot**	빠른 개발 속도와 함께 견고하고 확장 가능한 API 서버 구축.
API ↔ Inference 통신	RPC Framework	**gRPC**	추론 서버는 학습된 모델을 별도 Worker에서 로드·실행하므로, HTTP보다 직렬화 비용이 적은 gRPC를 선택.

3.4 도메인 모델 설계 (ERD)

시스템의 모든 비즈니스 로직은 다음 핵심 엔티티들의 관계를 기반으로 작동합니다. 특히 **PostgreSQL**의 **JSONB** 타입을 활용하여 스키마 유연성을 확보합니다.

- **[RDS]**
    - **User:** 사용자 인증 및 권한 관리를 위한 주체.
    - **Dataset:** 사용자가 업로드한 데이터셋의 메타데이터를 관리하며, 실제 파일은 **S3 주소** 형태로 저장합니다.
    - **Model:** 모델의 기본 정보(이름, 종류)를 관리합니다.
    - **ModelVersion:** 특정 **Model**에 대한 N개의 버전을 관리하며 (**1:N 관계**), 학습 상태, S3 주소, **JSONB** 기반의 스키마와 메트릭을 포함합니다.
    - **TRAINING_JOBS: ModelVersion** 생성에 사용된 **실행 환경 및 이력**을 기록합니다. **MODEL_VERSIONS**와 1:1 관계를 가지며, 장기간 학습 파이프라인의 상세 정보를 보관합니다.
- **[noSQL]**
    - **InferenceLog:** 추론 요청 시 발생한 비정형 입력 데이터, 추론 결과, 사용된 모델 버전 등 **대용량 로그**를 기록합니다.


USERS
uuid id PK Default: gen_random_uuid()
varchar email UK NN, 로그인 ID (Unique)
varchar password NN, Encrypted
enum role NN, CUSTOMER | ML_ENGINEER
datetime created_at NN, Default: NOW()
datetime updated_at NN
datetime deleted_at Soft Delete
manages
uploads
MODELS
uuid id PK
uuid user_id FK REFERENCES USERS.id
varchar name UK NN, 모델 이름 (Unique)
enum task_type NN, CLASSIFICATION | REGRESSION | CLUSTERING | ANOMALY_DETECTION
text description Optional
datetime created_at NN
datetime updated_at NN
datetime deleted_at Soft Delete
DATASETS
uuid id PK
uuid user_id FK NN, REFERENCES USERS.id
varchar name NN
varchar s3_key NN, S3 Object Path
bigint file_size NN, Byte 단위
jsonb schema_snapshot NN, 컬럼 명세 (JSON)
datetime created_at NN
datetime updated_at NN
datetime deleted_at Soft Delete
has_versions
defines_training_data_for
MODEL_VERSIONS
uuid id PK
uuid model_id FK NN, REFERENCES MODELS.id
uuid dataset_id FK NN, REFERENCES DATASETS.id
int version NN, UK(Composite): model_id + version
enum status NN, PENDING | TRAINING | READY | FAILED
jsonb metadata NN, Hyperparameters
jsonb input_schema Dynamic Input Spec
jsonb output_schema Dynamic Output Spec
jsonb metrics Validation Results
varchar weights_path S3 Path (Nullable if FAILED)
datetime created_at NN
datetime updated_at NN
datetime deleted_at Soft Delete
executed_by
TRAINING_JOBS
uuid id PK
uuid model_version_id FK NN, REFERENCES MODEL_VERSIONS.id
jsonb resource_config NN, {gpu: 1, mem: '16Gi'}
datetime start_time Nullable (실행 시작 시 기록)
datetime end_time Nullable (종료 시 기록)
varchar log_path S3 or File Path for Error Logs
datetime created_at NN
datetime updated_at NN

3.5 설계 상의 고려사항 및 Trade-off

단일 API 서버 vs 완전한 MSA

본 시스템은 완전한 마이크로서비스 아키텍처(MSA)가 아닌, 단일 API 서버에서 메타데이터 관리 및 요청 처리를 담당하는 구조를 채택했습니다. Training Worker와 Inference Worker는 분리했지만, 이들은 독립적인 API를 제공하는 서비스라기보다는 작업 전용 워커에 가깝습니다. 완전한 MSA(예: Dataset Service, Model Service, Training Service, Inference Service 분리)는 다음과 같은 이유로 채택하지 않았습니다:

- **운영 복잡도**: 서비스 간 통신 오버헤드, 분산 트랜잭션 처리, 서비스 디스커버리 등 추가 인프라 및 관리 부담이 발생합니다.
- **개발 속도**: 초기 개발 및 프로토타입 단계에서는 모놀리식 구조가 빠른 반복 개발에 유리합니다.
- **데이터 일관성**: 모델, 버전, 데이터셋 간의 강한 트랜잭션 일관성이 필요한 경우, 단일 DB 트랜잭션으로 처리하는 것이 간결합니다.

대신, 향후 확장 시 도메인별로 서비스를 분리할 수 있도록 내부 모듈을 도메인 기준으로 구성하고, 인터페이스를 명확히 정의했습니다.

Kafka 선택

작업 지연을 허용해야 하는 학습 특성 때문에 메시지 큐가 적합했습니다. RabbitMQ도 충분히 가능했으나, 장기 Job 처리에서는 Kafka의 Log 기반 구조가 안정적이고, 메시지 영속성 및 재처리가 용이하다는 점에서 선택했습니다.

JSONB 스키마 사용

모델별로 입출력 스키마가 다르기 때문에 RDB 정규화를 강제하면 유연성이 떨어집니다. 대신 PostgreSQL의 JSONB를 활용해 구조 확장성을 확보했습니다. 단점으로는 정교한 쿼리 최적화가 제한적이며, 필요 시 인덱스 전략이 추가로 필요할 수 있습니다.

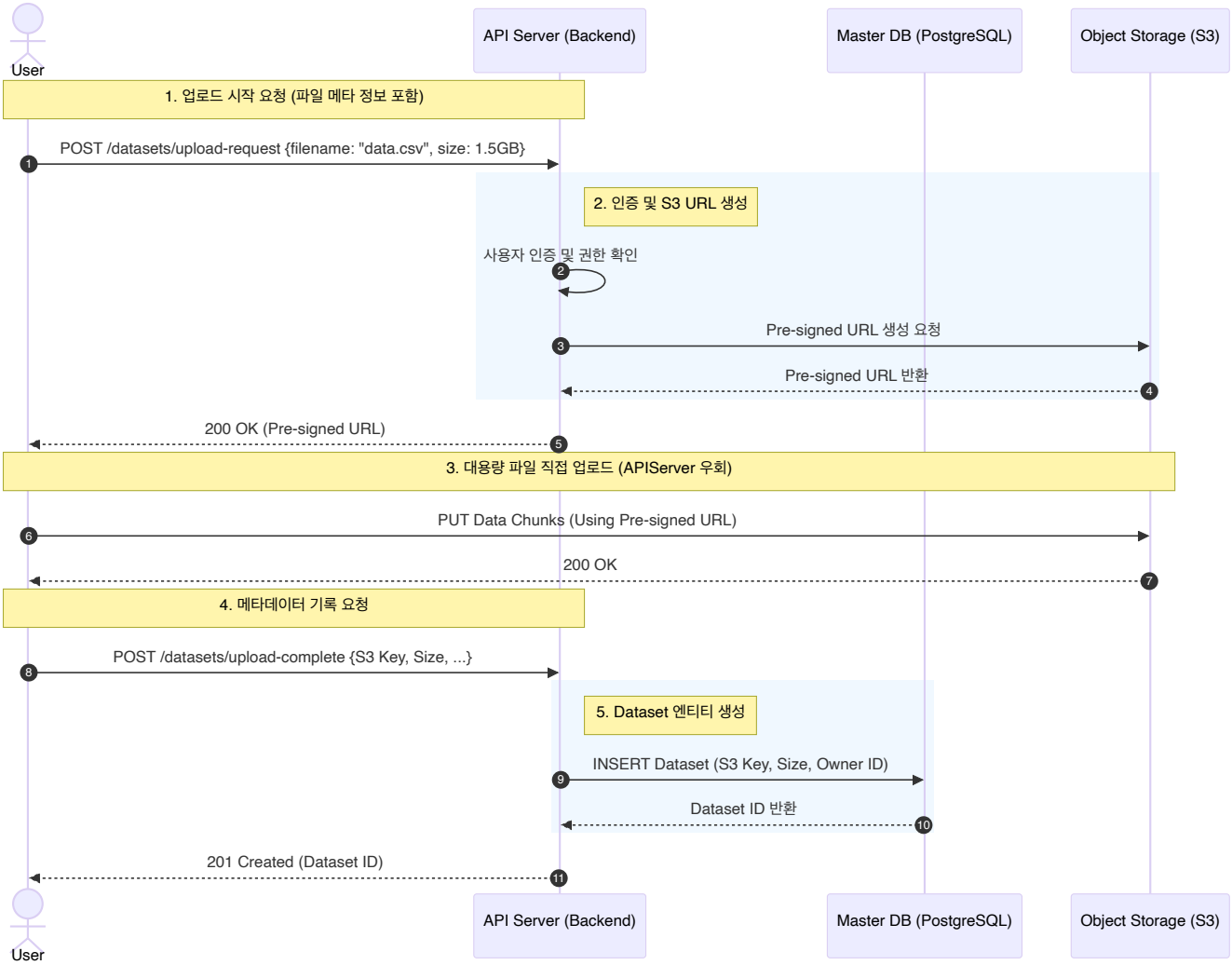
---

4. 핵심 기능별 상세 설계

4.1 데이터셋 업로드 및 관리

대용량 파일(최대 2GB) 처리 요구사항을 반영하여, **APIServer**를 거치지 않고 사용자 클라이언트가 **S3**에 직접 데이터를 전송하는 방식을 사용해 부하를 분산하고 효율성을 높입니다.

1. **업로드 요청 및 인증: APIServer**가 사용자로부터 요청을 받으면, 인증 후 S3에 파일을 업로드할 수 있는 **Pre-signed URL**을 생성합니다.
2. **데이터 직접 전송:** 사용자 클라이언트는 **Pre-signed URL**을 사용하여 대용량 파일(최대 2GB)을 **S3**에 **멀티파트 업로드 (Multipart Upload)** 방식으로 직접 전송합니다.
3. **메타데이터 기록:** 업로드 완료 후, 클라이언트 요청을 받은 APIServer는 해당 S3 주소와 파일 사이즈를 포함하여 **PostgreSQL**에 `Dataset` **엔티티** 레코드를 생성합니다.
4. **대용량 처리 준비:** 이후 Worker가 대용량 파일 처리 시 **OOM**을 방지하기 위해 **스트리밍(Chunk)** 방식으로 S3 데이터를 읽어 처리할 준비를 완료합니다.



4.2 모델 학습 파이프라인 (장기간 비동기)

모델 학습은 장기간 소요되는 작업이므로, **Apache Kafka**를 통해 API 서버의 부하를 막고 Worker를 분리하는 비동기 방식으로 처리됩니다.

1. **요청 수신 및 버전 생성: APIServer**가 요청을 수신하면, **PostgreSQL**에 `ModelVersion` **status를 PENDING 상태**로 생성하여 Job ID를 확보합니다.
2. **비동기 Job 발행:** 학습에 필요한 메타데이터(ModelVersion ID, 데이터셋 S3 주소)를 **Kafka Topic**에 메시지로 구성하여 발행합니다.
3. **Worker Pool 처리:** 미리 배포된 **Training Worker Pod들(K8s Deployment)**이 **Kafka에서 메시지를 Pull 방식으로 소비**하여 학습을 수행합니다. Worker는 **S3**에서 데이터셋을 **스트리밍 방식**으로 읽어 GPU 학습을 진행합니다.
4. **결과 저장 및 상태 업데이트:** 학습 완료 후, 완성된 모델 파일을 **S3**에 저장하고, **PostgreSQL**의 해당 `ModelVersion` 상태를 **READY**로, 모델 S3 주소 및 성능 지표를 업데이트합니다.

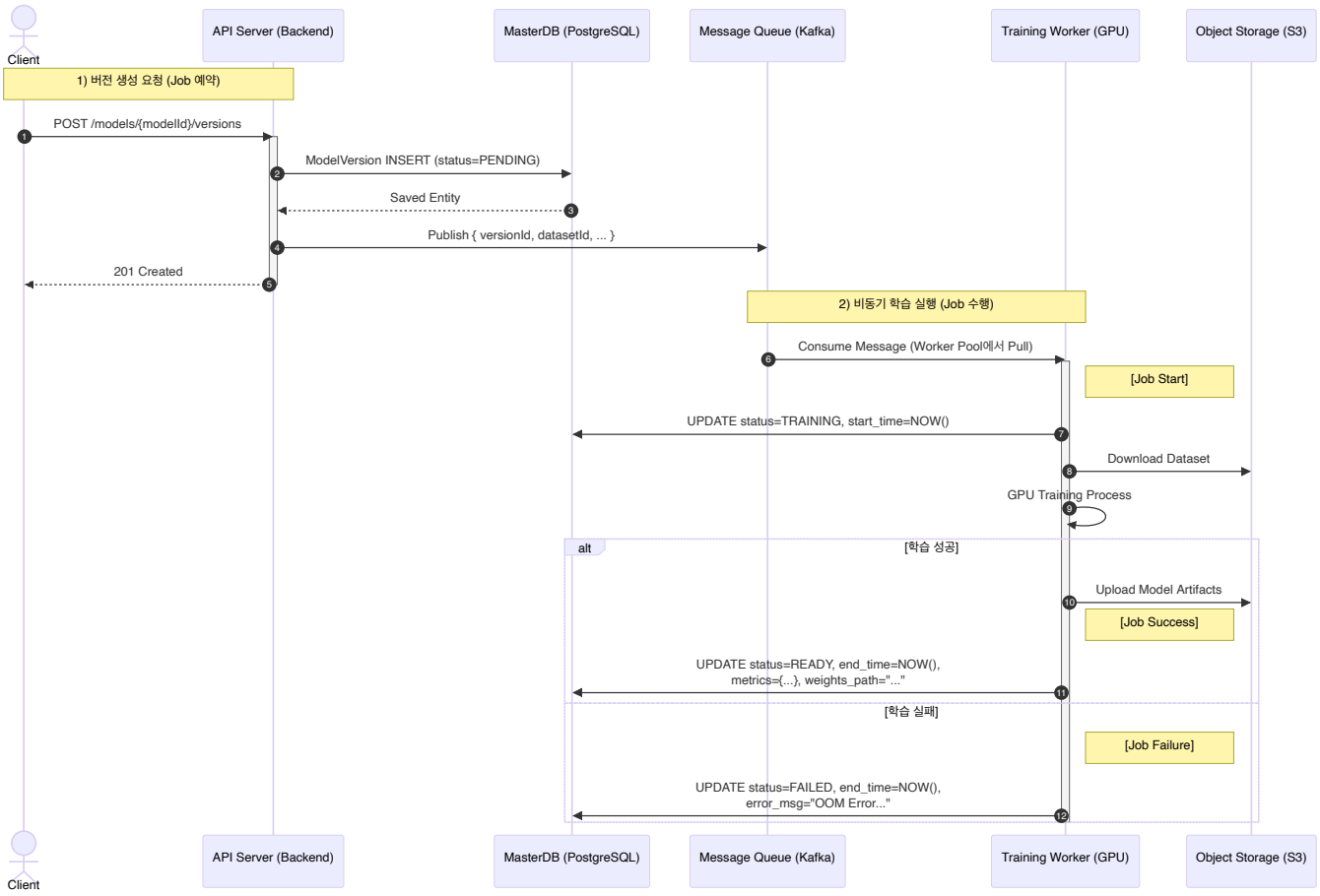
장애 및 엣지 케이스 대응 (학습 파이프라인)

학습 파이프라인의 안정성을 확보하기 위해 다음과 같은 장애 대응 메커니즘을 적용합니다.

대용량 파일 OOM 방지: Worker Pod는 S3에서 데이터를 다운로드할 때 스트리밍(Chunk) 방식으로 파일을 읽어 처리하여 메모리 사용량을 일정 수준 이하로 유지합니다.

Worker Pod 장애 시 재시도: Kafka의 재시도 메커니즘을 활용하여, 학습 중 Worker Pod에 장애가 발생하면 메시지를 다른 Worker에 자동으로 재전달합니다. 일정 횟수 재시도 후에도 실패할 경우 해당 메시지를 데드-레터-큐(DLQ)로 이동시켜 운영팀이 확인할 수 있도록 합니다.

스토리지 장애 대응: 학습 과정에서 S3 접근 실패 시 재시도를 수행하며, 최종 실패 시 해당 작업의 `ModelVersion` 상태를 `FAILED`로 기록하여 사용자에게 실패 상태를 전달합니다.



4.3 모델 추론 서빙 (실시간 동기)

추론은 사용자에게 실시간 응답을 제공해야 하는 동기 API 작업이며, **JSONB** 기반의 유연한 스키마 처리와 **MongoDB** 로깅이 핵심입니다.

1. **요청 수신 및 검증: APIServer**가 추론 요청을 수신하고, **PostgreSQL**에서 **JSONB** 필드를 조회하여 요청된 모델 버전의 **스키마 유효성을 검증**합니다.
2. **Worker 할당 및 모델 로드:** 요청을 **gRPC**를 통해 **Inference Serving Pod** (Server)에 전달합니다. Server는 **S3**에

서 모델 파일을 로드합니다.

3. **추론 및 로깅:** Server가 추론을 수행하고, 완료 즉시 추론 요청 및 최종 응답 데이터를 **MongoDB**에 **비동기적으로 기록**합니다.
4. **응답:** 최종 추론 결과를 사용자에게 신속하게 API 응답으로 반환합니다.

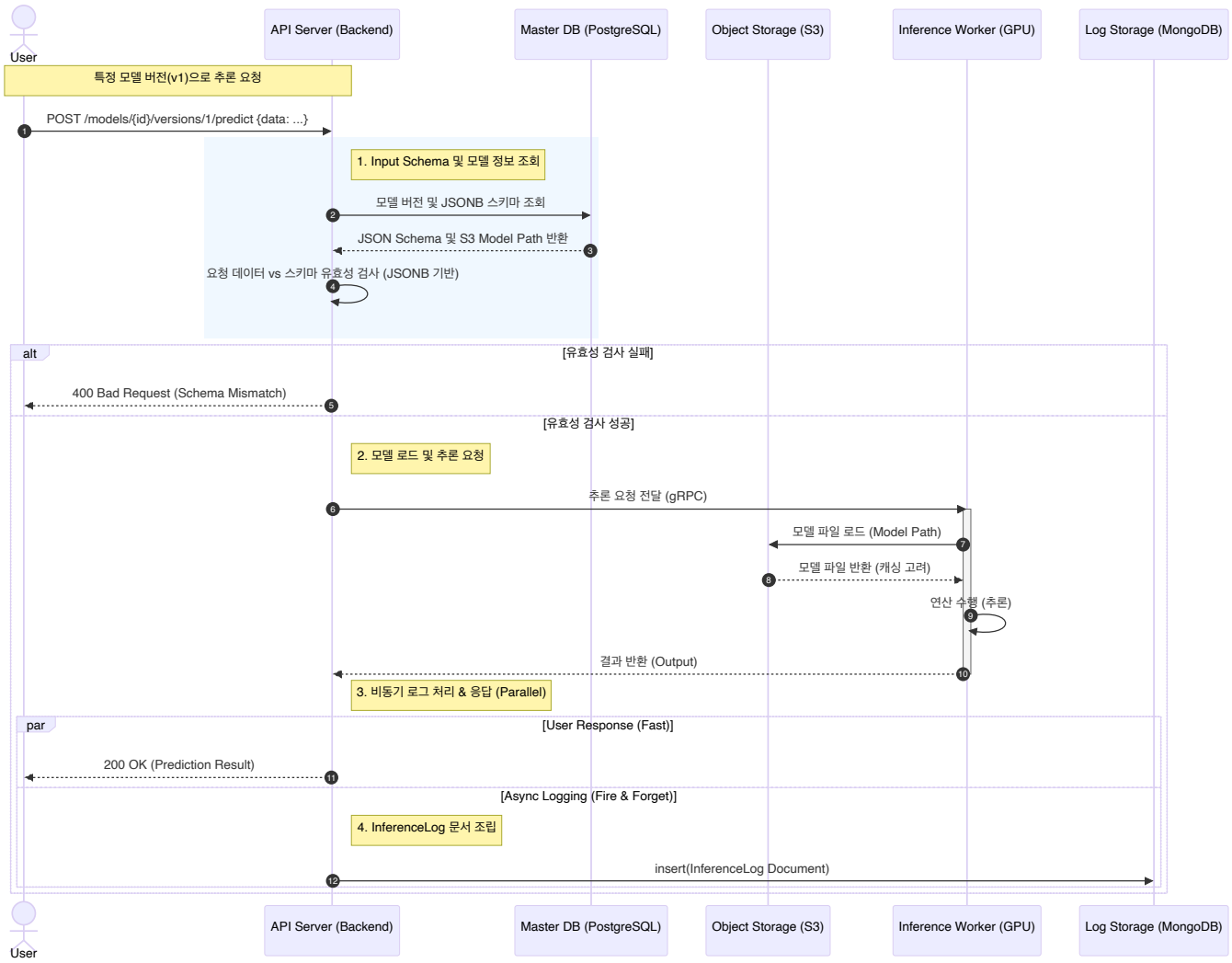
장애 및 엣지 케이스 대응 (추론 서빙)

추론 서빙의 안정성과 가용성을 확보하기 위해 다음과 같은 장애 대응 전략을 적용합니다.

모델 로딩 실패 대응: 추론 과정에서 S3 접근 실패로 모델 파일을 로드하지 못할 경우, 재시도를 수행합니다. 최종 실패 시에는 사용자에게 `503 Service Unavailable` 응답을 반환하여 일시적인 장애 상황임을 알립니다.

스키마 검증 실패 처리: 요청 데이터가 모델의 입력 스키마(JSONB)와 불일치할 경우, 어떤 필드가 잘못되었는지 오류 메시지와 함께 `400 Bad Request`를 반환합니다.

추론 서버 과부하 대응: Inference Worker Pod의 CPU/Memory 사용률이 임계치를 초과할 경우, Kubernetes의 HPA(Horizontal Pod Autoscaler)가 자동으로 Pod 수를 증가시켜 부하를 분산합니다.



5. 운영 및 관리 방안

5.1 데이터 분석 환경 구축 (MLOps 관점)

내부 사용자의 대용량 데이터 분석 요구사항을 충족하고 운영 부하를 분산하기 위해 격리된 분석 환경을 제공합니다.

- **분석용 데이터 소스 활용:**
  - **메타데이터 분석: PostgreSQL SlaveDB**를 활용하여 통계 쿼리 부하를 Master DB에서 분산합니다.
  - **추론 데이터 분석: MongoDB**에 기록된 대용량의 비정형 추론 로그에 직접 접근하여 모델 품질 분석을 수행합니다.
  - **원본 데이터 접근: S3**에 저장된 원본 데이터셋에 대한 직접 접근 경로를 제공합니다.
- **데이터 접근 제어:** 각 저장소에 대한 접근은 **최소 권한 원칙**을 따르며, 내부 사용자 그룹에 대해 분석 및 학습에 필요한 데이터에 대한 **읽기(Read-Only) 권한**을 부여하는 **RBAC(Role-Based Access Control) 정책**을 수립하고 적용합니다.

5.2 운영 정책 및 리소스 관리

시스템의 안정적인 운영과 GPU 자원의 효율적 활용을 위해 다음과 같은 운영 정책을 수립합니다.

5.2.1 데이터 보존 및 삭제 정책

정책 항목	세부 정책
엔티티 삭제 (Soft Delete)	사용자가 데이터셋, 모델, 모델 버전을 삭제 요청 시, 물리적 삭제 대신 `deleted_at` 필드를 기록하는 **논리적 삭제(Soft Delete)**를 적용합니다. **90일** 간 보관 후 백그라운드 배치 작업을 통해 물리적으로 삭제합니다.
학습 이력 데이터 정리	`TRAINING_JOBS` 테이블은 로그 성격의 데이터로, 모델 버전이 `READY` 또는 `FAILED` 상태로 완료된 후 **180일** 경과 시 **자동 아카이빙**(S3로 이동) 또는 삭제를 수행합니다. 운영 DB 부하 경감 및 스토리지 비용 최적화를 목적으로 합니다.
추론 로그 보관 주기	MongoDB에 저장되는 `InferenceLog` 문서는 생성일 기준 **30일** 후 자동 삭제합니다(TTL Index 활용). 장기 분석이 필요한 경우 S3/Data Lake로 별도 백업합니다.

5.2.2 학습 요청 제한 정책 (Rate Limiting)

GPU 자원 고갈 및 무분별한 학습 요청을 방지하기 위해 다음과 같은 **사용자별 Rate Limit**를 적용합니다.

제한 기준	제한 정책
동시 학습 요청 수	사용자당 **동시에 PENDING 또는 TRAINING 상태인 학습 작업을 최대 3개**로 제한합니다. 신규 요청 시 기존 학습 작업 상태를 확인하고, 초과 시 `429 Too Many Requests` 응답을 반환합니다.
시간당 학습 요청 횟수	사용자당 **1시간 동안 최대 5회**의 학습 요청만 허용합니다. Redis 기반 카운터를 사용하여 요청 빈도를 추적하고 제한합니다.
일일 학습 요청 할당량	일반 사용자는 **1일 최대 10회**, 내부 ML 엔지니어는 **1일 최대 50회**의 학습 요청 할당량을 부여받습니다. 초과 시 다음 날 00:00(UTC)에 할당량이 재설정됩니다.

5.2.3 학습 타임아웃 정책

정책 항목	세부 정책
최대 학습 시간 제한	학습 작업이 **36시간**을 초과할 경우, Worker는 작업을 자동 종료하고 해당 `ModelVersion` 상태를 `FAILED`로 업데이트합니다. 일반적인 12~24시간 학습을 고려하되, 예외 케이스(대용량 데이터셋, 복잡한 모델)를 수용하기 위한 버퍼를 포함합니다.
타임아웃 예외 처리	특정 사용자(ML 엔지니어 등)에게 **화이트리스트 기반 예외 권한**을 부여하여, 요청 시 `resource_config`에 `max_training_hours: 72`와 같은 확장 옵션을 명시할 수 있도록 합니다.
무응답 감지	Worker가 **1시간** 동안 상태 업데이트(heartbeat)를 하지 않을 경우, 해당 작업을 장애로 간주하고 재시도 또는 실패 처리합니다.

5.2.4 리소스 할당 우선순위 정책

Kafka의 **다중 파티션** 또는 **별도 토픽**을 사용하여 우선순위별 메시지를 분리하고, Worker가 우선순위 높은 파티션을 먼저 소비하도록 설정합니다.

우선순위	대상	설명
High	내부 사용자의 학습 요청	모델 개선 및 연구 목적의 학습 작업에 GPU 자원을 우선 할당합니다.
Normal	일반 사용자의 학습 요청	기본 우선순위로, FIFO(First-In-First-Out) 방식으로 처리됩니다.
Low	재시도 작업	실패 후 재시도하는 작업은 낮은 우선순위로 처리하여 신규 요청에 영향을 최소화합니다.

---

6. 결론

본 문서는 ArgMax Mini 백엔드 시스템의 전반적인 설계를 다루었습니다. 시스템은 데이터셋 관리, 모델 학습, 추론 서빙이라는 MLOps의 핵심 워크플로우를 지원하며, Kubernetes 기반의 인프라 위에서 GPU 자원을 효율적으로 활용하도록 설계되었습니다.

설계 과정에서는 장기간 소요되는 학습 작업과 실시간 응답이 필요한 추론 요청이라는 서로 다른 특성을 가진 워크로드를 분리하는 데 중점을 두었습니다. 이를 위해 API 서버, Training Worker, Inference Worker를 독립적인 컴포넌트로 구성하고, Kafka를 통한 비동기 메시징으로 학습 파이프라인의 안정성을 확보했습니다. 데이터 저장 전략은 PostgreSQL, S3, MongoDB를 각 데이터의 특성과 접근 패턴에 맞게 조합하여 구성했습니다.

운영 측면에서는 GPU 자원의 효율적 관리를 위한 정책들(Rate Limiting, Timeout, 우선순위)과 데이터 보존 정책을 수립했으며, 주요 장애 시나리오에 대한 대응 방안을 각 워크플로우별로 정리했습니다. 전체적으로 현재 요구사항을 만족하면서도 향후 확장 가능성을 고려한 설계를 지향했습니다.