실시간 주식 감성 분석 파이프라인 개발

주식 데이터의 실시간 수집, 뉴스 및 소셜 미디어 감성 분석,
대시보드 시각화까지 직접 구현해보는 프로젝트

강의 개요

본 강의에서는 **"직접 구현하며 배우는"** 방식으로 진행됩니다.

1. 각 컴포넌트마다 **아키텍처**를 설계해봅니다
2. 컴포넌트에 필요한 **요구사항**과 **패키지 정보**를 확인합니다
3. 제시된 기능을 **직접 구현**합니다
4. 모범 답안과 **비교 분석**합니다
5. 실제 환경에서 **테스트**합니다

실제 구현 코드와 함께 실시간 주식 데이터 수집 및 감성 분석 시스템의 A to Z를 배웁니다.

학습 목표

- Apache Kafka를 활용한 **실시간 데이터 스트리밍** 구현
- **NLP 기반 감성 분석** 파이프라인 구축
- **주식 뉴스 및 소셜 미디어 데이터** 수집 및 처리
- **실시간 대시보드** 구현을 통한 데이터 시각화
- **Docker** 기반 애플리케이션 컨테이너화

시스템 아키텍처 설계하기

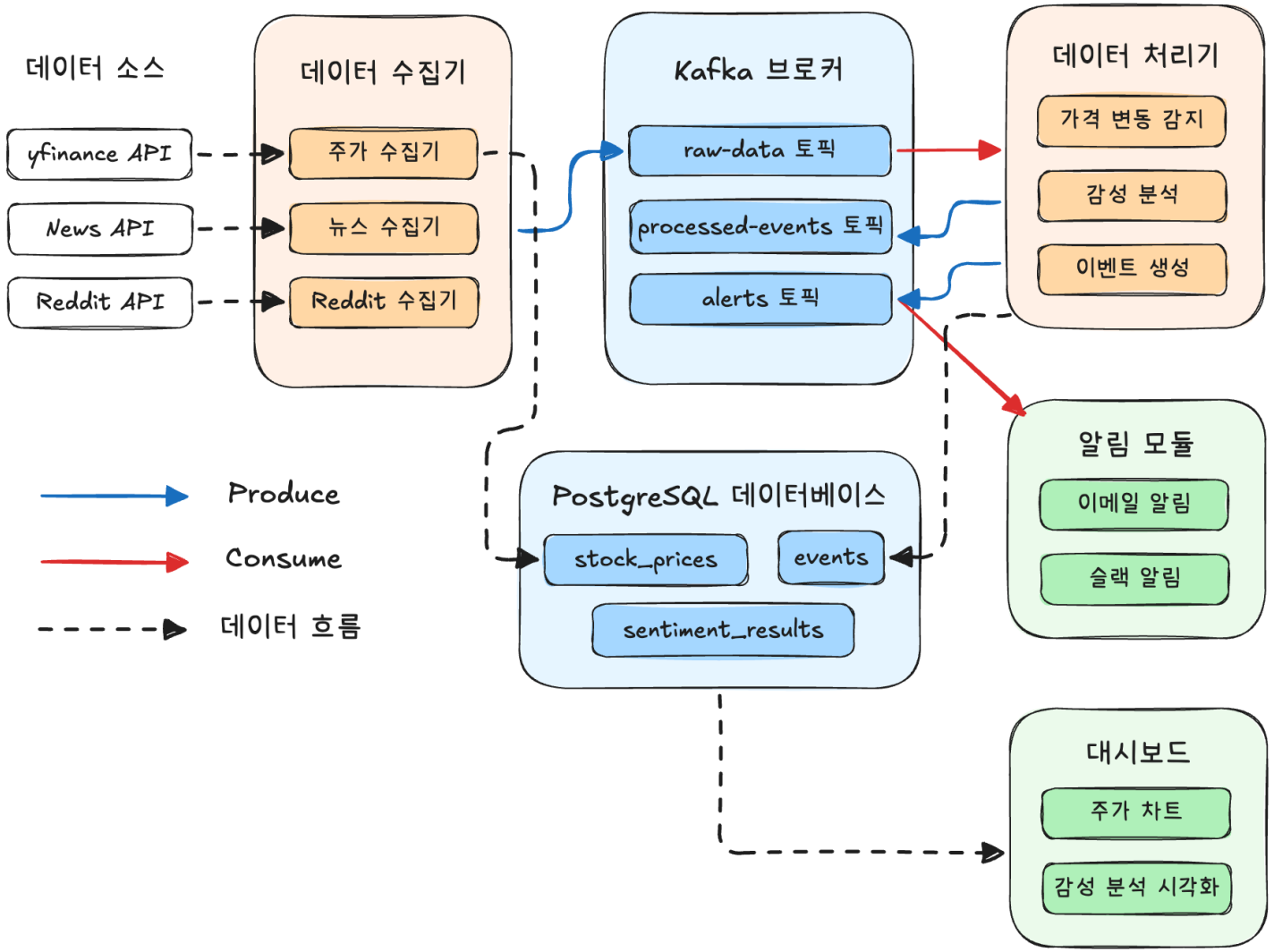
🚀 실습 1: 주식 감성 분석 파이프라인 아키텍처 설계

요구사항:

- 실시간 주가 데이터를 수집해야 함
- 뉴스와 소셜 미디어(Reddit) 데이터를 수집해야 함
- 텍스트 데이터에 대한 NLP 기반 감성 분석을 수행해야 함
- 사용자 정의 알림 조건을 처리해야 함
- 여러 알림 채널(Slack, 이메일)을 지원해야 함
- 데이터를 시각화하는 실시간 대시보드를 제공해야 함

실습: 이 요구사항을 만족하는 시스템 아키텍처를 그려보세요.

주식 감성 분석 파이프라인 아키텍처 (참고)



시스템 컴포넌트 개요

1. **데이터 수집기** - 주식 데이터, 뉴스, Reddit 게시물 수집하여 Kafka로 전송
2. **데이터 처리기** - Kafka에서 데이터를 소비하고 가격 변동 감지 및 감성 분석 수행
3. **감성 분석 프로세서** - 뉴스 및 소셜 미디어 텍스트에 대한 감성 분석 수행
4. **알림 모듈** - Slack 및 이메일을 통한 알림 전송
5. **데이터 스토리지** - PostgreSQL 데이터베이스에 데이터 저장
6. **대시보드** - Dash 및 Plotly를 이용한 데이터 시각화

Kafka 토픽 구조

시스템에서 사용하는 Kafka 토픽은 다음과 같습니다:

1. **raw-data**: 원본 데이터가 저장되는 토픽
    - 주식 가격 데이터
    - 뉴스 기사 데이터
    - Reddit 게시물 데이터
2. **processed-events**: 데이터 처리 결과가 저장되는 토픽
    - 가격 변동 감지 결과
    - 감성 분석 결과
    - 기타 처리된 이벤트

3. **alerts**: 알림 메시지가 저장되는 토픽
    - 중요 가격 변동 알림
    - 감성 분석 기반 알림
    - 사용자 정의 알림

데이터 수집기에서는 모든 수집 데이터를 `raw-data` 토픽에 전송해야 합니다.

데이터 처리 흐름 이해하기

데이터 처리 파이프라인의 흐름은 다음과 같습니다:

1. **수집된 데이터** → raw-data 토픽에 저장
    - 주식 가격 데이터 (data_type: 'stock_price')
    - 뉴스 데이터 (data_type: 'news')
    - Reddit 데이터 (data_type: 'reddit')
2. **데이터 처리기**가 raw-data 토픽에서 메시지를 소비
    - 데이터 타입에 따라 적절한 처리 함수 호출
    - process_stock_data(): 주식 가격 변동 감지
    - process_news_data(): 뉴스 및 Reddit 게시물의 감성 분석

3. **처리 결과** → processed-events 토픽에 저장
    - 주가 급등/급락 이벤트
    - 거래량 급증 이벤트
    - 감성 분석 결과
4. **알림 메시지** → alerts 토픽에 저장
    - 중요 주가 변동 알림
    - 강한 감성 (매우 긍정적/부정적) 알림
5. **데이터베이스**에 중요 정보 저장
    - 주가 데이터
    - 이벤트 데이터
    - 감성 분석 결과

데이터 수집기

🚀 실습 2-1: 주식 데이터 수집기

```
def collect_stock_data(self):
    for ticker in self.stocks:
        # TODO: yfinance로 주식 데이터 가져오기
        # TODO: 최신 주가 정보 추출
        # TODO: 주가 데이터 메시지 생성
        stock_data = {
            "ticker": ticker, "price": price, "volume": volume,
            "timestamp": timestamp.isoformat(),
            "data_type": "stock_price"
        }
        # TODO: Kafka에 데이터 전송 - 'raw-data' 토픽 사용
        # TODO: 데이터베이스에 저장 (db_helper.py 참고)
```

🚀 실습 2-2: 뉴스 데이터 수집기 구현

기능 요구사항:

- News API를 통해 주식 관련 뉴스 기사 수집
- 각 모니터링 대상 주식 종목별로 관련 뉴스 검색
- 수집한 뉴스 데이터를 Kafka 'raw-data' 토픽으로 전송
- 오류 처리 및 재시도 로직 포함

기술 스택:

- newsapi-python 또는 requests
- kafka-python

Kafka 토픽 구조 참고:

- `raw-data` : 원본 데이터 저장 (주식 가격, 뉴스, Reddit 데이터)
- `processed-events` : 처리된 이벤트 데이터 저장 (가격 변동, 감성 분석 결과)
- `alerts` : 알림 메시지 저장 (중요 이벤트 발생 시)

```
def collect_news_data(self):
    """뉴스 API를 통한 주식 관련 뉴스 수집"""
    for ticker in self.stocks:
        # TODO: 뉴스 API 쿼리 파라미터 구성
        # TODO: 뉴스 API 요청 및 결과 처리
        # TODO: 수집한 뉴스 데이터를 Kafka 'raw-data' 토픽으로 전송
        # TODO: 오류 처리
```

🚀 실습 2-3: Reddit 데이터 수집기 구현

기능 요구사항:

- Reddit API(PRAW)를 통해 관련 서브레딧에서 주식 관련 게시물 수집
- 모니터링 대상 주식과 관련된 게시물 필터링
- 수집한 Reddit 데이터를 Kafka 'raw-data' 토픽으로 전송

기술 스택:

- praw (Python Reddit API Wrapper)
- kafka-python

Kafka 활용 가이드:

- 모든 수집 데이터는 `raw-data` 토픽에 전송
- **데이터 유형(data_type)을 명확히 지정**하여 후속 처리가 용이하도록 합니다
- 메시지 키(key)는 관련 주식 티커를 사용합니다

```
def collect_reddit_data(self):
    """Reddit API를 통한 주식 관련 게시물 수집"""
    # TODO: Reddit API 연결
    # TODO: 지정된 서브레딧에서 게시물 수집
    # TODO: 주식 관련 게시물 필터링
    # TODO: Reddit 데이터를 Kafka 'raw-data' 토픽으로 전송
    # TODO: 오류 처리
```

컴포넌트 2: 데이터 처리기

🚀 실습 3-1: 데이터 처리기 클래스 설계

기능 요구사항:

- Kafka 'raw-data' 토픽에서 데이터 소비
- 데이터 유형(주식 가격, 뉴스, Reddit)에 따른 처리 로직 구현
- 처리 결과를 Kafka 'processed-events' 및 'alerts' 토픽으로 전송
- 중요 데이터 및 이벤트를 PostgreSQL에 저장

기술 스택:

- kafka-python (데이터 소비 및 생산)
- nltk (자연어 처리 및 감성 분석)
- psycopg2-binary (데이터베이스 연결)
- pandas (데이터 처리)

```
class DataProcessor:
    """주식 데이터 및 뉴스/Reddit 데이터 처리기"""

    def __init__(self):
        """데이터 처리기 초기화"""
        self.stocks = MONITORED_STOCKS
        self.price_config = PRICE_ALERT_CONFIG
        self.sentiment_config = SENTIMENT_CONFIG

        # Kafka 헬퍼 초기화
        self.kafka = KafkaHelper()

        # 데이터베이스 헬퍼 초기화
        self.db = DatabaseHelper()

        # 감성 분석기 초기화
        self.sentiment_analyzer = SentimentIntensityAnalyzer()

        # 주식별 이전 가격 데이터
        self.prev_prices = {}

        # 주식별 마지막 알림 시간
        self.last_alert = {}
        for ticker in self.stocks:
            self.last_alert[ticker] = {
                'price_surge': datetime.now() - timedelta(days=1),
                'price_drop': datetime.now() - timedelta(days=1),
                'volume_surge': datetime.now() - timedelta(days=1)
            }
```

참고: 메소드 각각은 다음 실습에서 차근차근 구현 예정

```
class DataProcessor: # 이어서 구현
  def process_stock_data(self, message):
      """주식 가격 데이터 처리 및 변동 감지"""
      # 추후 구현

  def process_news_data(self, message):
      """뉴스 및 Reddit 데이터 처리 및 감성 분석"""
      # 추후 구현

  def run(self):
      """데이터 처리 메인 루프"""
      # 추후 구현
```

🚀 실습 3-2: 주식 가격 데이터 처리 구현

기능 요구사항:

- 주식 가격 및 거래량 데이터 추출
- 이전 가격 대비 변동률 계산
- 가격 급등/급락 및 거래량 급증 기준에 따른 이벤트 감지
- 감지된 이벤트를 'processed-events' 토픽으로 전송
- 중요 가격 변동 알림을 'alerts' 토픽으로 전송
- 이벤트 데이터를 데이터베이스에 저장

```python
def process_stock_data(self, message):
    """주식 가격 데이터 처리 및 이벤트 감지"""
    # TODO: 메시지 파싱 (ticker, price, volume, timestamp)

    # if ticker not in self.prev_prices:
      # TODO: ticker가 처음 등장한 경우 초기화 로직 작성 (DB에 저장된 이전 가격/거래량 조회 가능)
    # TODO: 이전 가격/거래량 가져오기

    # TODO: 가격 변동률(price_change_pct), 거래량 변동률(volume_change_pct) 계산

    # TODO: 가격 급등/가격 급락/거래량 급증 조건 판별 (임계값 초과 + 쿨다운 지남)
    # TODO: 이벤트 생성, Kafka 전송, DB 저장, last_alert 갱신

    # TODO: self.prev_prices 업데이트 (이동 평균 등 적용 가능)
```

🚀 실습 3-3: 뉴스 및 Reddit 데이터 처리 구현

기능 요구사항:

- 뉴스 또는 Reddit 메시지에서 제목(title)과 내용(content) 추출
- NLTK VADER 감성 분석기를 사용하여 텍스트 감성 분석
- 긍정(positive), 부정(negative), 중립(neutral) 감성 점수 및 라벨 생성
- 감성 분석 결과를 데이터베이스에 저장
- 특정 임계값을 넘는 중요 감성 변화 발생 시 알림 생성

데이터 구조:

- 입력: 뉴스 또는 Reddit 데이터 (JSON 형식)
- 감성 점수 범위: -1.0 (매우 부정적) ~ +1.0 (매우 긍정적)
- 중요 감성 임계값: > 0.5 (매우 긍정적) 또는 < -0.5 (매우 부정적)

```python
def process_news_data(self, message):
    """뉴스 및 Reddit 데이터 처리 및 감성 분석"""
    # TODO: 메시지 파싱 (ticker, title, content, source, url, published_at)

    # TODO: 감성 분석을 위한 텍스트 구성 (title + content)
    # TODO: 감성 점수 계산 (self.sentiment_analyzer.polarity_scores 사용)
    # TODO: 감성 점수(compound_score)를 기반으로 라벨 결정 (positive, negative, neutral)
    # TODO: 감성 분석 결과를 DB에 저장

    # TODO: 임계값 기준으로 알림 조건 판단 (compound_score > 0.5 or < -0.5)
    # TODO: 알림 메시지 구성 (Kafka 전송용 딕셔너리 생성)
    # TODO: Kafka alerts 토픽으로 메시지 전송
```

실행 메소드 구현 - run

데이터 처리기의 메인 루프는 아래와 같이 구현할 수 있습니다:

```
def run(self):
    """데이터 처리 메인 루프"""
    # TODO: 처리기 시작 로그 출력
    # TODO: Kafka 토픽 존재 확인
    # TODO: Kafka consumer 생성 (raw_data 토픽 대상)
    #       → 실패 시 로그 출력 후 종료
    try:
        for message in self.kafka.consumer:
            data = message.value
            # TODO: 데이터 타입 확인 (예: 'stock_price', 'news', 'reddit')
            # TODO: 데이터 타입에 따라 적절한 처리 함수 호출
            #       - stock_price → process_stock_data
            #       - news/reddit → process_news_data
            #       - 기타 → 경고 로그 출력
    except KeyboardInterrupt:
        pass
    finally:
        # TODO: Kafka, DB 등 자원 정리
        pass
```

컴포넌트 3: 알림 모듈

🚀 실습 4: 알림 모듈 설계 및 구현

기능 요구사항:

- Kafka 토픽에서 알림 이벤트 소비
- 이메일 알림 전송 기능 구현
- Slack 웹훅을 통한 알림 전송 기능 구현
- 알림 유형별 서로 다른 포맷 적용 (가격, 거래량, 감성)
- 중복 알림 방지 로직 구현
- 알림 전송 실패 시 재시도 로직 구현

기술 스택:

- kafka-python
- requests (Slack 웹훅)
- smtplib (이메일 전송)
- jinja2 (템플릿 엔진)

알림 모듈 - 클래스 설계

```python
class Notifier:
    """이메일과 슬랙을 통한 알림 처리기"""

    def __init__(self):
        """알림 처리기 초기화"""
        # TODO: 환경 설정 로드
        # TODO: Kafka 컨슈머 초기화
        # TODO: 중복 알림 방지를 위한 추적 데이터 구조 초기화

    def send_email(self, subject, message, severity="info"):
        """이메일 알림 전송"""
        # TODO: 이메일 메시지 생성
        # TODO: SMTP 서버 연결 및 인증
        # TODO: 이메일 전송
        # TODO: 오류 처리 및 재시도 로직

    def send_slack(self, message, severity="info"):
        """슬랙 알림 전송"""
        # TODO: 슬랙 메시지 페이로드 구성
        # TODO: 웹훅 요청 전송
        # TODO: 응답 확인 및 오류 처리

    def process_alert(self, alert):
        """알림 처리 및 전송"""
        # TODO: 알림 타입 및 메시지 추출
        # TODO: 중복 알림 검사
        # TODO: 알림 유형에 따른 서로 다른 처리
        # TODO: 알림 전송

    def run(self):
        """알림 처리 메인 루프"""
        # TODO: Kafka에서 알림 메시지 소비
        # TODO: 각 알림 처리
        # TODO: 예외 처리
```

send_slack

```
def send_slack(self, message, severity="info"):
    """
    슬랙 알림 전송

    Args:
        message (str): 슬랙 메시지
        severity (str): 알림 심각도 (info, warning, critical)

    Returns:
        bool: 성공 여부
    """
    # 심각도에 따른 색상 설정
    color = "#3498db"  # 기본 파란색 (info)
    if severity == "warning":
        color = "#f39c12"  # 노란색
    elif severity == "critical" or severity == "high":
        color = "#e74c3c"  # 빨간색

    # 슬랙 메시지 페이로드 구성
    payload = {...}

    # 슬랙 웹훅 요청
    response = requests.post(
        self.slack_config['webhook_url'],
        data=json.dumps(payload),
        headers={"Content-Type": "application/json"}
    )

    if response.status_code == 200:
        logger.info(f"슬랙 알림 전송 성공")
        return True
    else:
        logger.error(f"슬랙 알림 전송 실패: {response.status_code} - {response.text}")
        return False
```

process_alert

```
def process_alert(self, alert):
    """
    알림 처리 및 전송 (스켈레톤)

    Args:
        alert (dict): Kafka 'alerts' 토픽에서 받은 알림 데이터
    """

    # TODO: 알림 정보 추출 (type, message, severity)
    #       → alert.get('type', 'unknown') 등 기본값 설정 추천

    # TODO: 중복 방지를 위한 고유 알림 키(alert_key) 생성
    #       → 예: f"{alert_type}:{message}"
    current_time = datetime.now()

    # TODO: recent_alerts에서 alert_key 존재 여부 확인
    #       → 있으면 마지막 전송 시각과 비교하여 300초 이내면 skip (return)

    # TODO: alert_type에 따라 이메일 제목(subject) 설정

    # TODO: 이메일 전송 (self.send_email 함수 활용)

    # TODO: 슬랙 전송 (self.send_slack 함수 활용)

    # TODO: recent_alerts에 alert_key와 current_time 저장

    # TODO: 처리 로그 기록 (logger.info 등)
```

run

```
    def run(self):
        """알림 처리 메인 루프"""
        # TODO: Kafka 토픽 존재 확인 (self.kafka.ensure_topics_exist)

        # TODO: Kafka consumer 생성 ('alerts' 토픽 대상)
        #           → 실패 시 로그 출력 후 종료

        try:
            # TODO: Kafka 컨슈머로부터 메시지 계속 받아오기 (for 루프)
            for message in self.kafka.consumer:
                # TODO: 각 알림 메시지에 대해 self.process_alert(alert) 호출
        except KeyboardInterrupt:
            # TODO: 종료 요청 로그 출력
            pass
        finally:
            # TODO: Kafka 등 자원 정리 (self.kafka.close)
```

컴포넌트 4: 대시보드

🚀 대시보드 설계 및 구현

기능 요구사항:

- 주식 가격 데이터 시각화
- 감성 분석 결과 시각화
- 주요 이벤트 타임라인 표시
- 최근 뉴스 및 감성 점수 표시
- 실시간 데이터 업데이트
- 사용자 인터랙션 기능 (종목 선택, 기간 선택)

기술 스택:

- Dash (Plotly)
- Plotly (데이터 시각화)

코드 분석

컴포넌트 5: 배포 구성

🚀 실습 5: Docker 및 Docker Compose 구성

기능 요구사항:

- 각 컴포넌트를 Docker 컨테이너로 패키징
- Docker Compose로 전체 시스템 통합 구성
- 환경 변수 관리 및 설정
- 볼륨 마운트 및 데이터 영속성 보장
- 컨테이너 간 네트워크 구성
- 로깅 및 모니터링 설정

기술 스택:

- Docker

Dockerfile 구현

```
FROM python:3.9-slim

WORKDIR /app

# 필요한 패키지 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# NLTK 데이터 다운로드
RUN python -m nltk.downloader vader_lexicon punkt stopwords

# 소스 코드 복사
COPY . .

# 환경 변수 설정
ENV PYTHONUNBUFFERED=1

# 실행 명령 설정
CMD ["python", "run.py"]
```

Docker Compose 구성

```yaml
version: '3'

services:
  # Zookeeper
  zookeeper:

  # Kafka
  kafka:

  # PostgreSQL
  postgres:

  # 데이터 수집기
  data-collector:

  # 데이터 처리기
  data-processor:

  # 알림 모듈
  notifier:

  # 대시보드
  dashboard:

networks:
  app-network:
    driver: bridge

volumes:
  postgres-data:
```

환경 변수 설정 (.env)

주요 환경 변수를 `.env` 파일로 관리합니다:

```
# Kafka 설정
KAFKA_BOOTSTRAP_SERVERS=kafka:9092

# 데이터베이스 설정
DB_HOST=postgres
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=postgres
DB_NAME=stockdb

# 뉴스 API 설정
NEWS_API_KEY=your_news_api_key

# Reddit API 설정
REDDIT_CLIENT_ID=your_reddit_client_id
REDDIT_CLIENT_SECRET=your_reddit_client_secret
REDDIT_USER_AGENT=StockSentimentBot/1.0

# 감성 분석 설정
SENTIMENT_THRESHOLD=0.4

# 모니터링 설정
MONITORED_STOCKS=AAPL,MSFT,GOOG,AMZN,TSLA
COLLECTION_INTERVAL=300

# 이메일 설정
EMAIL_SMTP_SERVER=smtp.gmail.com
EMAIL_SMTP_PORT=587
EMAIL_USERNAME=your_email
EMAIL_PASSWORD=your_app_password
EMAIL_SENDER=your_email
EMAIL_RECIPIENTS=recipient1@example.com,recipient2@example.com

# Slack 설정
```

통합 실행 및 테스트

모든 컴포넌트를 함께 실행하고 테스트하는 과정입니다:

1. **Docker Compose 실행**

```
docker-compose up -d
```

2. **로그 확인**

```
docker-compose logs -f data-collector
docker-compose logs -f sentiment-processor
```

3. **대시보드 접속**
   - http://localhost:8050 에서 대시보드 확인
4. **감성 분석 결과 확인**
   - 데이터베이스에서 감성 분석 결과 확인

성능 최적화 및 모니터링

시스템 성능을 향상시키고 모니터링하기 위한 방법들:

1. **Kafka 튜닝**
    - 파티션 수 최적화
    - 리텐션 정책 설정
    - 배치 설정 최적화
2. **데이터베이스 인덱싱**
    - 주요 쿼리에 대한 인덱스 생성
    - 주기적인 VACUUM 설정
3. **효율적인 감성 분석**
    - 배치 처리 최적화

프로젝트 완료 및 실제 사용 예시

실제 사용 시나리오:

시나리오 1: 급격한 감성 변화 감지

```
[Slack] 2025-05-03 14:30:21
⚠️ TSLA 관련 감성 급변 알림
감성 변화: 중립 → 강한 부정 (-0.68)
관련 뉴스: "Tesla faces production challenges amid supply chain disruptions"
최근 5시간 동안 부정적 뉴스 비율: 78%
```

시나리오 2: 감성과 가격 움직임 연계

```
[이메일] 주식 감성-가격 연관성 알림
AAPL에 대한 긍정적 감성 급등 (24시간 내 +0.45)이 감지되었습니다.
이와 함께 주가도 상승 중 (현재 $183.45, +2.3%)
주요 원인: 신제품 출시에 대한 긍정적 반응
```

마무리 및 다음 단계

학습한 내용:

- Apache Kafka를 활용한 실시간 데이터 스트리밍
- NLP 기반 감성 분석 파이프라인 구축
- 분산 시스템 구성 및 메시지 처리
- Docker를 통한 마이크로서비스 배포
- 데이터 시각화 및 대시보드 구현

다음 단계:

- AI 기반 트레이딩 모델 개발 (다음 프로젝트)
- 커스텀 감성 분석 모델 학습
- 더 많은 데이터 소스 통합
- 클라우드 환경 배포 및 스케일링

감사합니다