gafka

Go 기반 분산 메시지 큐 설계 및 구현

Kafka를 직접 구현하며 분산 시스템 역량 강화

👨‍💻 발표자: 한결

📅 발표일: 2025-07-27

🧠 Why - 왜 만들었는가?

기존 아키텍처의 문제

- 서비스 간 직접 호출: **결합도 높음**
- 장애 전파: **신뢰성/복원력 부족**
- 네트워크 이슈 시 **데이터 유실**

학습 목표

- Kafka의 핵심 개념 실습
- WAL, 배칭, 복제 등 분산 시스템 설계 감각 체득
- 실시간 로그/이벤트 기반 처리 경험 축적

🎯 설계 목표 및 성능 지표

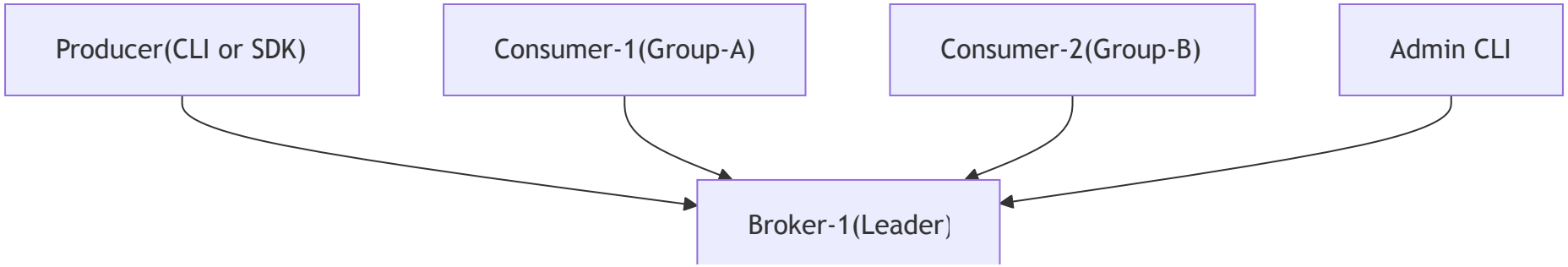
항목	목표	테스트 기준
처리량	≥ 5,000 msg/s	1KB 메시지 기준 wrk 부하
지연시간	평균 ≤ 100ms, P95 ≤ 200ms	Producer → Consumer E2E 측정
내구성	100% 복구 보장	WAL 기반 재시작 시나리오
장애 복구	≤ 30초	브로커 kill 후 재가동
순서 보장	파티션 내 순서 100% 유지	오프셋 기반 순차 처리 검증

📌 What - 무엇을 만드는가?

핵심 기능 요약

- Producer ↔️ Broker ↔️ Consumer 구조
- **WAL 기반 메시지 영속성**
- **Long Polling 기반 Pull 모델**
- **컨슈머 그룹** + 오프셋 기반 재처리
- **리더-팔로워 복제 구조**
- **Key 기반 파티셔닝 (hash(key) % N)**

🧩 시스템 아키텍처 (MVP)



> MVP 단계에서는 HTTP + JSON 통신 / 인메모리 코디네이터

🔄 데이터 흐름

1. Producer가 Key 기반 파티션 선택
2. Broker가 메시지 수신 → WAL 기록
3. 리더 → 팔로워 비동기 복제
4. Consumer는 Long Polling으로 Pull
5. 오프셋 기반 재처리 가능

💾 메시지 구조와 WAL

```
type Message struct {
        Topic     string
        Partition int32
        Offset    int64
        Key       string
        Value     string
        Timestamp int64
        Size      int32
}
```

- **WAL**: 로그 단위 저장, 장애 시 복구
- **Checksum 포함**으로 무결성 보장
- **세그먼트 기반 저장 구조** 도입 예정

👥 Consumer Group 및 리밸런싱

- 그룹 내 한 파티션은 **한 Consumer만** 소비
- **Heartbeat** 기반 생존 확인
- Coordinator가 **리더 선출 → 파티션 재할당**
- 리밸런싱은 **자동/동적**으로 수행됨

⚙️ 기술 스택

항목	기술	이유
언어	Go 1.22	동시성, 빠른 빌드
통신	HTTP + JSON	MVP 개발 편의성
저장	In-memory + WAL	속도 + 영속성 균형
CLI	cobra	명령어 기반 유틸
테스트	go test	TDD 적용
성능 측정	wrk	TPS, Latency 측정
로깅	logrus	단계별 로깅

🧪 성능 테스트 계획

항목	도구	목표	검증 방법
TPS	wrk	≥ 5K msg/s	1KB 메시지 30초 부하
Latency	go client	P95 ≤ 200ms	produce → consume
장애 복구	수동	≤ 30초	브로커 kill 후
배치 효과	wrk	개선 확인	배치 vs 단건

🗓️ 개발 일정 요약 (9주)

주차	구현 목표
1~2주	파티션, WAL
3~4주	API, Producer
5~6주	Consumer Group, 리밸런싱
7~8주	복제, 장애 복구
9주차	통합 테스트, 문서화

🔮 향후 확장 계획 (Phase 2+)

- gRPC / TCP 커스텀 프로토콜
- 메시지 압축 (gzip/snappy)
- 리더 자동 선출 (Raft)
- Prometheus 모니터링
- 인증 및 TLS
- 쿠버네티스 배포 / 오토스케일링
- 멀티 데이터센터 복제

✅ 요약

- Kafka 핵심 아키텍처를 직접 설계/구현한 학습 프로젝트
- Go의 동시성과 WAL, 배칭 등을 이용한 고성능 메시지 큐
- **Throughput / Durability / Scalability**를 모두 고려
- TDD, 문서화, 자동화 테스트 포함한 **실무 중심 설계**

❓ Q&A

> 궁금한 점 물어보세요!
> 혹은 이 시스템을 실무에 적용한다면 어떤 문제가 생길까요?