| **호선우**  Backend Engineer | [hocaron](https://github.com/hocaron)  [velog.io@haron](https://velog.io/@haron/posts)  010-9967-4547  hocaron.dev@gmail.com |
| --- | --- |

| **Education** | |
| --- | --- |
| 이화여자대학교 사이버보안학과 | 2017.03 ~ 2025.03  이화여자대학교 휴면기계바이오공학부 | 2017.03 ~ 2025.03 | |

| **Experience** | |
| --- | --- |
| **컬리** 회원 플랫폼 개발팀 | 2023.01 ~ 현재  12,000 TPS의 처리량(평균 응답시간 1.3ms, P99 3.7ms)을 보장하는 세션·OAuth 2.0 기반 통합 인증 시스템을 설계·운영하고 있습니다. 사용자 인증, 기기 관리, 이력 추적 기능과 fallback 처리를 통한 고가용성을 제공합니다. | |
| **기술 스택**   | Backend | Java, Spring Boot, Spring Cloud Gateway, Spring Cloud Config, gRPC | | --- | --- | | Database | MySQL 8.0(JPA), Redis(Lettuce, Redisson) | | Messaging | Kafka Streams | | |
| **인증 시스템 고도화**   * JWT 기반 클라이언트 측 토큰 갱신의 보안 취약점을 개선하기 위해 세션 기반 인증 방식과 쿠키를 도입하여 인증 체계를 강화했습니다. 글로벌 미들웨어를 구현하여 신구 인증 시스템간의 데이터 전환과 하위 호환성을 유지했습니다. 생산성 개선: 클라이언트 로컬 개발 환경과 개발 환경 서버의 도메인 차이로 인한 쿠키 이슈를 Spring Cloud Gateway Filter로 해결했습니다. Set-Cookie 헤더의 Domain과 Secure 속성을 재정의하고, 요청 헤더의 플래그 값을 기반으로 적절한 개발 서버로 라우팅하여 클라이언트 개발자가 로컬 개발환경에서 개발과 테스트를 진행할 수 있게 했습니다. 별도 인스턴스 구성을 통해서도 해결이 가능했지만, 추가 서버 인프라 비용, 관리 복잡성, 배포 오버헤드 등의 문제로 Gateway를 배치하여 추가 배포 및 프로파일 관리 없이 유지보수성을 높였습니다. * 인증 방식 변경으로 증가한 내부 서버 호출을 개선하고자 gRPC를 도입했습니다. POC와 성능 테스트를 거쳐 운영환경에 적용한 결과, JSON 기반 통신 대비 지연 시간을 50% 개선하고 리소스 사용률도 50% 줄여 인프라 효율성을 높일 수 있었습니다. 생산성 개선: Docker를 활용하여 로컬 환경의 protoc 설치 복잡성을 해결하고, 볼륨 마운트 방식으로 protoc 파일만 있으면 자동으로 인터페이스 코드를 생성하는 환경을 구축하여 개발 생산성을 개선했습니다. * 세션 저장소를 Redis와 MySQL로 이중화하여 빠른 응답 속도를 제공하고, 안정성을 높였습니다. 안정성 개선: DAU, 키 사이즈 분석으로 필요한 메모리를 계산했고, 스펙을 산정하여 Redis Scale Up을 진행했습니다. 과거 Scale Up 과정에서 발생했던 DNS Cache TTL 과 RDB 동기화 지연 이슈를 Lettuce DNS Cache 설정과 라우팅 조건 구현으로 무중단 확장을 진행했습니다. 향후 개선 포인트: 저장소 동시 장애에 대비하여 ALB Sticky Session 기반의 Local Cache 도입을 검토중입니다. * 로그인 이상 패턴을 IP 기반으로 실시간 탐지하기위해 Kafka Streams와 슬랙 알림을 활용하여 보안 위험에 대응하는 프로세스를 마련했습니다. 보안성 개선: 별도의 상태 저장소 구현 없이도 자체 상태 관리와 다양한 윈도우 연산을 제공하는 Kafka Streams를 선택했습니다. 연속된 시간대의 시도를 누적하여 분석할 수 있는 Sliding Window를 채택했고, 실제 공격 패턴으로 테스트하여 최적의 윈도우 임계값을 도출했습니다. X-Forwarded-For 헤더에서 WAF IP나 사내 IP가 오탐으로 간주되는 것을 방지하고자 IP 추출 로직을 개선했습니다. | |
| **성능 최적화 및 시스템 안정화**   * 외부 이벤트로 인한 트래픽 처리를 위해 Lambda를 통한 라우팅 후 Spring Boot Server로 요청을 전달하는 구조에서, 스파이크 트래픽(500 TPS) 발생 시 12초가 소요되던 초기 응답 시간을 2초 내외로 개선했습니다.  성능 향상: Lambda의 Cold Start 문제는 Java 코드를 Python으로 변경하여 4초에서 1초로 개선했습니다. 성능 향상: Spring Boot Server 초기 지연 문제는 JVM 모니터링과 jconsole 기반 Code Cache 사용량 분석을 통해 성능 병목을 파악했고, Tomcat Thread Pool과 JVM JIT 컴파일러 최적화를 위한 웜업 코드 구현을 하였습니다. 이 외에 DB 커넥션 풀 및 트랜잭션 위치 최적화, Feign Client의 Apache HttpClient 기반 풀 커넥션 방식 적용을 통해 초기 응답 지연을 8초에서 3초로 개선했습니다. 하지만 높은 초기 트래픽으로 인한 지연이 여전히 존재하여, Lambda Concurrency 설정을 조정해 초기 트래픽을 제어함으로써 최종적으로 응답 시간을 1초 내외로 단축할 수 있었습니다. * MySQL InnoDB FK Shared Lock과 인덱스 Gap Lock으로 인한 데드락 문제를 Redis 분산락 도입과 DB 제약조건 조정으로 개선했습니다. 생산성 개선: MySQL Lock 관련 이슈를 전사에 공유하여, 유사 이슈 발생 시 신속하게 대응할 수 있도록 기반을 만들었습니다. * 기능의 동적 제어를 위한 Feature Flag 시스템을 Redis Pub/Sub 기반으로 구축하여 Polling 방식 대비 시스템 효율성을 높였습니다. * PHP-FPM의 요청별 상태 격리로 인한 커넥션 풀 관리 한계와 DB 부하 집중 문제를 Java 기반 API 로 재구축하여 해결했습니다. 커넥션 풀 도입으로 DB 비용을 60% 절감했고, HTTP 프로토콜 변경, 캐시 도입으로 12000TPS에서 안정적인 서비스를 제공할 수 있었습니다. * 클라이언트 API 호출 로직을 BFF 패턴으로 개선하고, WebClient 와 서킷 브레이커를 통해 외부 서비스 호출의 성능과 안정성을 높였습니다. 성능 향상: 비동기 호출 구현 시, CompletableFuture는 다수 요청 시 스레드 풀 한계로 지연 및 거부가 발생할 수 있어 제외했습니다. Reactive Feign은 WebClient 추상화로 인한 오버헤드가 있었고, 성능 테스트 결과도 WebClient가 가장 우수하여 채택했습니다. | |
| **개발 생산성 및 운영 효율성 향상**   * 서비스별 User-Agent 파싱 로직 불일치 문제를 해결하기 위해 ANTLR 기반 공통 라이브러리를 개발했습니다. 정규표현식으로도 구현이 가능하지만 복잡한 패턴 매칭과 유지보수의 어려움이 생겨, 구조적 분석과 확장이 용이한 ANTLR 를 사용했습니다. 생산성 개선: [🔗GitHub Actions 플러그인](https://github.com/marketplace/actions/lib-watchdog) 개발을 통해 라이브러리 버전 업데이트 시, 라이브러리를 사용중인 서비스의 업데이트 PR 생성 및 슬랙봇 알림을 자동화하여 변경 사항 반영 프로세스를 개선하였습니다. * 기존에는 개발환경 인프라 구축 시 티켓 발급을 통한 수동 작업이 필요했으나, ECS로 전환하여 개발자가 GitHub Actions으로 ALB 등록부터 배포 방식까지 관리할 수 있게 되었습니다. 또한 새로운 인스턴스 기동과 도커 컨테이너 구동까지 걸리던 8분의 배포 시간을 기존 EC2에 컨테이너만 추가 구동하는 방식으로 2분 이내로 단축했습니다. 향후 개선 포인트: Spring Boot 애플리케이션의 fat jar 방식에서 도커 레이어드 jar 방식으로 전환하여 이미지 빌드 시간 단축하고자 합니다. * 비즈니스 지표(회원가입/탈퇴율 등), 시스템 지표(CPU, Thread 개수)와 메트릭 지표(지연 시간, Cache Hit)를 파악할 수 있는 대시보드를 만들고, 임계치 초과 시 즉각적인 대응이 가능하도록 슬랙 알림을 연동했습니다. 생산성 개선: MDC 로깅을 추가하여 사용자 정보, IP 등 VOC 대응과 장애 추적에 필요한 정보를 파악할 수 있도록 개선했습니다. 특히 커스텀 토큰의 내부 필드들을 MDC에 자동으로 포함시켜 별도의 API 없이 요청 토큰 정보를 확인할 수 있도록 개선했습니다. | |
| **데이터 처리 및 저장소 최적화**   * MySQL에서 대규모 이력 데이터의 스키마 변경과 조회 성능 문제를 해결하기 위해 파티셔닝과 Redis를 도입했습니다.  안정성 개선: 모듈러 연산 데이터 버케팅을 통한 점진적 이관과 AOP 기반 이중 적재 방식으로 안전한 전환을 진행했습니다. 문제 발생 시 즉각적인 롤백이 가능하도록 구성했으며, 마이그레이션 완료 후에는 AOP 클래스 제거만으로 손쉽게 전환할 수 있었습니다. ALB Sticky Session 라우팅을 통해서도 마이그레이션이 가능했지만, 인프라 구조 세팅의 리소스 부담으로 인해 애플리케이션 레벨에서 직접 구현했습니다. 향후 개선 포인트: 모듈러 연산의 고정된 마이그레이션 비율 한계를 극복하기 위해 Range 방식 또는 일관성 해시를 도입하여 동적 비율 조정의 유연성을 확보하고자 합니다. * 삭제만 발생하는 데이터 특성에 Bloom Filter를 활용하여 DB 쿼리 수를 61% 절감했고, Stream 방식으로 메모리 효율성을 개선했습니다. * 단건 카운트 증가 쿼리의 Exclusive Lock 타임아웃 이슈를 Kafka Batch Listener 를 활용하여 다건 처리로 개선했습니다.  생산성 개선: Spring 배치는 처리 이력 관리용 테이블이 추가로 필요하고 실시간성이 떨어져, Kafka 배치 처리를 채택했습니다. 성능 개선: Kafka 배치 처리 중 발생하는 데드락을 해결하기 위해 파티션 키 방식와 메시지 정렬 방식을 검토했습니다. 파티션 키 방식은 특정 컨슈머로의 메시지 몰림으로 인한 처리 지연이 우려되어, 메시지 정렬 방식을 적용하고 트랜잭션 범위를 최소화하는 방식을 채택했습니다. | |

| **Side Project** | |
| --- | --- |
| **이벤트 순차 처리 및 비정규화 테이블을 활용한 성능 최적화 시스템 구축** 기술 스택 | Spring Boot, JPA, MySQL 8.0, AWS ECS, AWS ALB   * 빈번한 조회 데이터의 조인으로 인한 성능 저하를 비정규화 테이블 설계로 해결했습니다. * 비정규화로 인한 일관성 관리를 위해 BlockingQueue 기반 이벤트 순차 처리를 구현하여 동시성 이슈를 방지했습니다. 향후 개선 포인트: PostgreSQL로의 마이그레이션을 통해 Materialized View 도입을 검토하여 데이터 일관성 관리 부담을 줄일 예정입니다. | |
| **웹소켓을 이용한 게임 서버 구축** 기술 스택 | NestJS, Web Socket, Prisma, Nginx, Github Action   * 실시간 게임 서비스의 매칭-플레이-결과 전달 플로우를 웹소켓으로 구현하였습니다. * Let's Encrypt와 Nginx를 활용한 인프라 구성으로 안정성을 확보하고 비용을 절감했습니다. | |

| **Skills** | |
| --- | --- |
| Backend | Java, Spring Boot, Spring Data JPA Spring Cloud Gateway, Spring Cloud Config TypeScript, NestJS, TypeORM PHP, PHP-FPM  gRPC, Kafka Streams MySQL, Redis, MongoDB |
| DevOps | AWS EC2, ECS, ECR, S3, MySQL DynamoDB, ElastiCache, ALB 를 활용한 서버 운영  CloudWatch 를 활용한 로그 수집 Github Action, AWS CodeBuild, CodeDeploy, CodePipeline 을 이용한 배포 자동화 Let’s Encrypt, AWS Route 53, CA 를 활용한 도메인 구축 |
| Frontend | HTML5, CSS, Next.js |

| **Blog** | |
| --- | --- |
| [🔗](https://github.com/marketplace/actions/lib-watchdog)[COMMIT, MVCC 그리고 SET AUTOCOMMIT](https://helloworld.kurly.com/blog/commit-mvcc-set-autocommit/)  [🔗](https://github.com/marketplace/actions/lib-watchdog)[외래키(Foreign Key)와 데드락(DeadLock) 그리고 쿼리 지연 실행](https://velog.io/@haron/%EC%99%B8%EB%9E%98%ED%82%A4Foreign-Key%EC%99%80-%EB%8D%B0%EB%93%9C%EB%9D%BDDeadLock-%EA%B7%B8%EB%A6%AC%EA%B3%A0-%EC%BF%BC%EB%A6%AC-%EC%A7%80%EC%97%B0-%EC%8B%A4%ED%96%89-eruedsy4)  [🔗](https://github.com/marketplace/actions/lib-watchdog)[인덱스와 데드락](https://velog.io/@haron/%ED%8A%B8%EB%9F%AC%EB%B8%94%EC%8A%88%ED%8C%85-DB-%EC%9D%B8%EB%8D%B1%EC%8A%A4Index%EC%99%80-%EB%8D%B0%EB%93%9C%EB%9D%BDDeadLock-in8ryzsm)  [🔗](https://github.com/marketplace/actions/lib-watchdog)[[블로그] Connection Pool 획득 과정에서 발생하는 지연시간 개선하기](https://velog.io/@haron/HikariCP-Connection-Pool-%ED%9A%8D%EB%93%9D-%EA%B3%BC%EC%A0%95%EC%97%90%EC%84%9C-%EB%B0%9C%EC%83%9D%ED%95%98%EB%8A%94-%EC%A7%80%EC%97%B0%EC%8B%9C%EA%B0%84-%EA%B0%9C%EC%84%A0%ED%95%98%EA%B8%B0)  [🔗](https://github.com/marketplace/actions/lib-watchdog)[[블로그] 첫번째 요청에서 발생하는 지연 시간을 개선해보자](https://velog.io/@haron/Spring-%EC%B2%AB%EB%B2%88%EC%A7%B8-%EC%9A%94%EC%B2%AD%EC%97%90%EC%84%9C-%EB%B0%9C%EC%83%9D%ED%95%98%EB%8A%94-%EC%A7%80%EC%97%B0-%EC%8B%9C%EA%B0%84%EC%9D%84-%EA%B0%9C%EC%84%A0%ED%95%B4%EB%B3%B4%EC%9E%90-hzadsfk7) | |

| **Activity** | | |
| --- | --- | --- |
| IT 동아리 Mash-Up | 2021.07 ~ 현재  카카오 테크 캠프 2기 멘토 | 2024.06 ~ 2024.12  IT 동아리 디프만 | 2022.09 ~ 2023.01 | | |