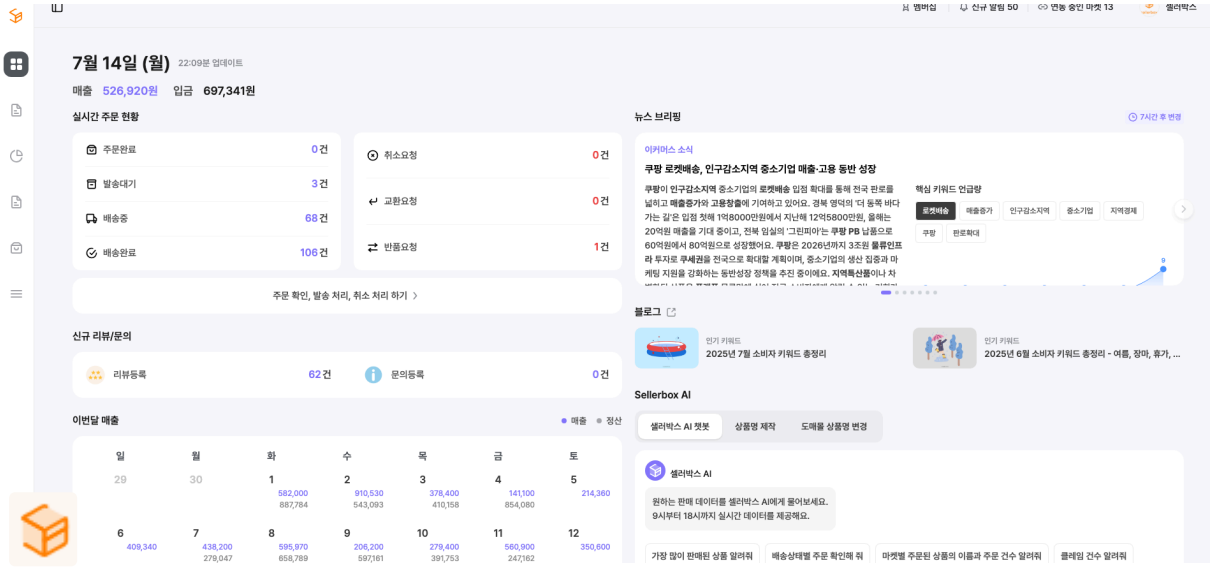


셀러박스 주문 수집고도화

태그	인프라 AWS ETL 데이터엔지니어 서클플랫폼
날짜	@2025년 6월 10일 → 2025년 7월 15일

개요

문제 상황

- 이전 셀러박스 고도화 프로젝트에서 수집서버를 EC2 → AWS Lambda 로 변경하며 대량요청을 빠르게 처리할 수 있었으나 **여러 인스턴스의 RPS 를 제어하기 힘들었음**

⚠️ 10분, 30분 마다 스케쥴러로 대량요청이 들어올 때 Lambda 인스턴스의 개수가 일정하지 않아 특정 마켓의 RPS 를 조절하기가 매우힘들었음

- A 라는 마켓은 연동중인 모든 **셀러의 규모에 따라 API 엔드포인트단위로 전체 RPS 를 공유**하게 되는데 **AWS Lambda 만으로는 이러한 제한사항에 쉽게 대처하기 힘들었음**

- 제한된 RPS 에 여러 서비스를 추가하게 되면서 각 서비스의 RPS 경쟁이 치열해져 유료서비스에 차질이 생겼음

⚠️ 최근 주문 발송처리, **수동 주문수집 기능과 AI 문의 답변, AI 리뷰 답변 처리**등의 서비스를 추가하면서

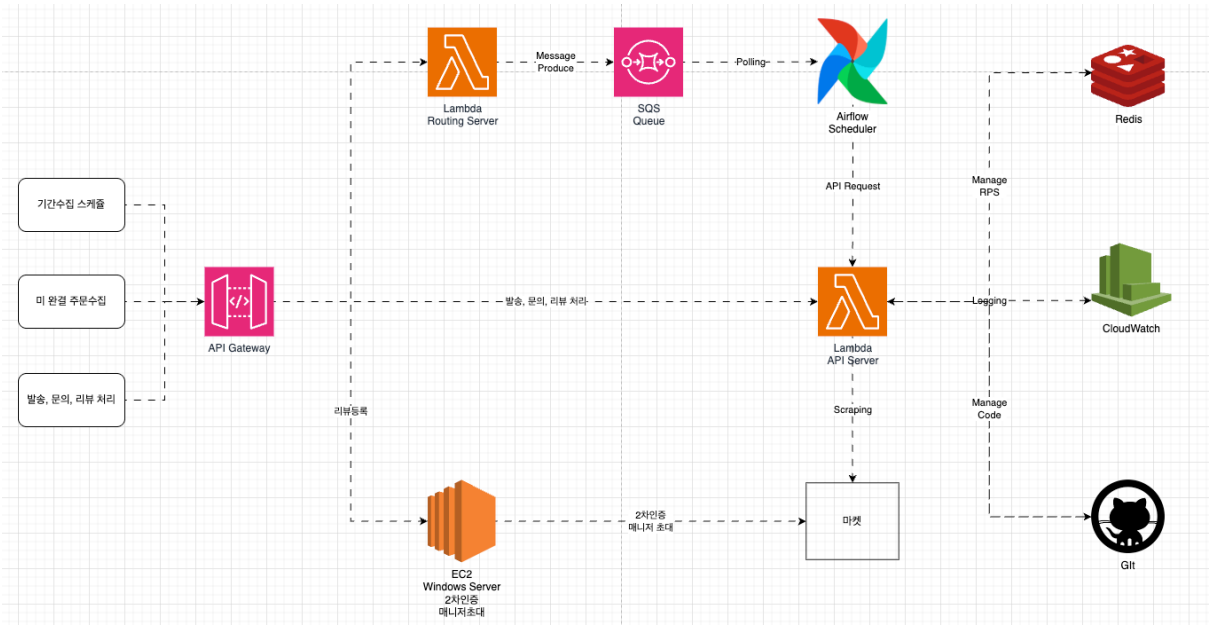
- 스케쥴러가 작동하는 시간동안에는 유료기능이 실패하는일이 잦아지고 있었음
- 또한 제한된 RPS 이상의 호출량을 요청하는 일도 생겨 고객사로부터 RPS 를 준수해달라는 요청을 받게됨

- 또한 **셀러박스 웹**을 공개하면서 상품 조회, 수정, 등록 및 과거주문조회 등의 기능추가 요청까지 생기면서 정해진 RPS 를 초과하지 않으면서도 효율적으로 사용해야하는 요구사항이 생겼음

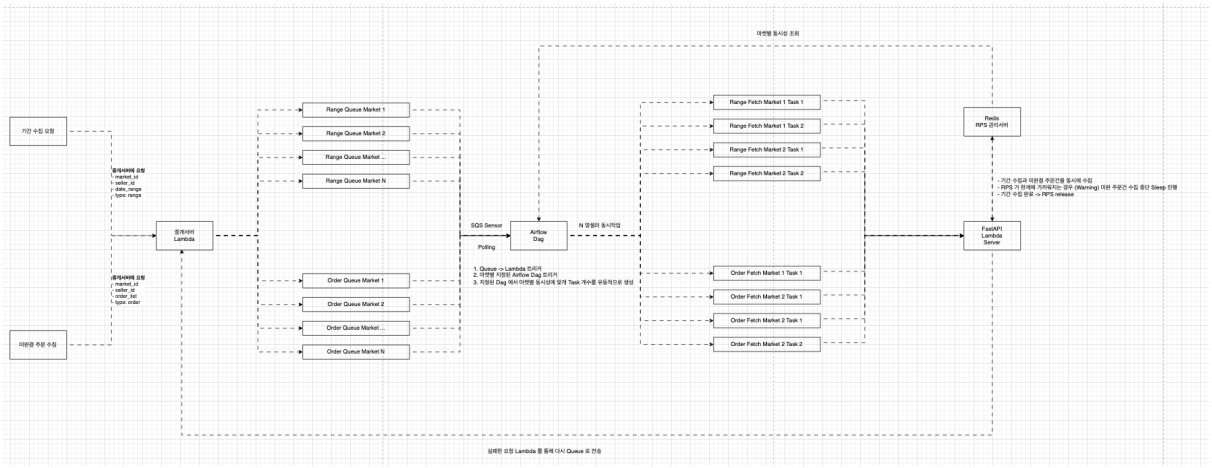
이에 서비스 기능 추가 전 아래 목표를 달성하기로 했습니다.

👉
- RPS 를 최대한 효율적으로 사용할 수 있는 아키텍처 도입
- 과거 주문에 대한 신뢰성을 제고할 수 있는 주문건 별 수집 API 생성

프로젝트 진행



수집서버 인프라구성도



기존 주문 수집 + 미완결 주문수집 로직 결합

인프라

- API Gateway
  - API Gateway - 운영
  - API Gateway - 개발
- AWS Lambda (FastAPI 서버)
  - 고정IP Lambda 1 - 운영
  - 고정IP Lambda 2 - 운영
  - 유동IP Lambda 3 - 운영

    - 고정IP Lambda 1 - 개발
    - 고정IP Lambda 2 - 개발
    - 유동IP Lambda 3 - 개발
- AWS SQS (수집요청 Message Queue)
    - 마켓 별 SQS 구성
- Airflow (스케쥴러)
    - AWS EC2 Autoscaling + Celery Executor 구성

> 👉 대량 요청시 Worker 개수를 늘려 동시 요청을 처리합니다.

    - SQS Sensor, SimpleHttpOperator

> 👉 수집 요청 리스트가 들어있는 SQS 를 정해진 Dag 가 바라보면서 FastAPI 로 수집 요청 및 상태 최신화 B/E API 를 호출 합니다.

- CloudWatch

> 👉 Lambda 인스턴스는 요청이 들어오지 않으면 자동으로 종료되기 때문에 로그는 watchtower 를 사용하여 CloudWatch 로 보냅니다.

    - 운영 로깅
    - 개발 로깅
- Valkey(Redis) AWS Elasticache

> 👉 수집서버에 자체 구성된 aiohttp 로 구성된 자체 구성된 요청 Class 에 연동하여 여러 Lambda 요청량을 중앙에서 제어합니다.

RPS 사용 전략 도입

- API 중요도 나누기

> 서비스의 긴급성을 고려하여 API 의 priority 를 low, medium, high 3단계로 구성하였습니다.
>
> - high: 상
>   - **RPS 100% 사용**
>   - 상품 발송, 문의, 리뷰 답변등 반드시 성공해야하는 요청
>   - 성공할때까지 요청
> - medium: 중
>   - **RPS 80% 사용**
>   - 주문 수집, 개별 수집, 수동 주문 수집 버튼
>   - 성공할때까지 요청
> - low: 하
>   - **RPS 70% 사용**
>   - 개별 주문 수집, 과거데이터 수집, 상품 정보 수집
>   - 일정 시도이후에는 실패

👉

결과

- **해당 방식으로 중요도가 높은 요청이 오기전까지는 내부적으로 RPS 의 여유를 항상 남겨** 두어 유료서비스가 사용해야하는 RPS 를 항상 남겨 놓습니다
- 또한, 고객사의 RPS 를 초과하기전 Redis 서버에서 요청을 중단하도록 하여 고객사 서버의 부담을 최소화 할 수 있었습니다.
- 다양한 서**비스를 계속 추가하게 되는 환경**에서 **서비스간의 RPS 경쟁으로 인한 실패를 줄여 고객 경험을 저하하지 않을 수 있는 방법을 인프라적으로 해결**하였습니다.