오랫동안 링크를 모아 공유하는 페이지를 운영해 왔다. 업계 동향, 읽을 거리, 개발 문서, 유용한 도구 같은 것을 한 자리에 모아두면 동료들이 자연스럽게 그곳으로 모인다. 문제는 시간이 지나면 금방 낡아진다는 점이다. 링크가 깨지거나 주제가 겹치고, 중복 등록이 생기고, 스팸이 스며든다. 사람 손으로만 정리하면 초반의 의욕이 사라지는 순간 링크모음은 순식간에 박물관 전시가 된다. 주소모음이나 사내 북마크 포털이 실패하는 패턴을 여러 번 봤다.

자동화는 요란한 도구를 잔뜩 붙이는 일이 아니다. 흐름을 단순하게 설계하고, RSS와 IFTTT 같은 단순한 기성 툴을 기본 축으로 잡은 뒤, 마지막 10퍼센트의 빈틈만 작은 스크립트로 메우는 편이 오래간다. 이번 글에서는 내가 써 온 구조와 시행착오, 그리고 실제로 손에 익는 스크립트와 설정을 나누겠다. 키워드 필터링과 신뢰도 점수, 중복 제거처럼 품질을 지키는 세부도 함께 살펴본다. 링크모음, 주소모음 무엇을 만들든 논리는 같다.

제대로 굴러가는 링크 파이프라인의 모양

성공하는 파이프라인은 세 층으로 나뉜다. 첫째는 수집, 둘째는 정리와 저장, 셋째는 배포다. 수집은 RSS가 바탕이다. 가능하면 소스로부터 직접 RSS를 받는다. RSS가 없을 때는 보조적인 수집 장치를 둔다. IFTTT의 RSS 트리거와 이메일 파서, 웹훅이 그 자리를 채운다. 정리 단계에서는 중복 제거와 정규화, 태그 분류, 신뢰도 평가가 돌아간다. 배포는 정적 사이트나 Notion 데이터베이스, 구글 시트 같은 곳으로 흘러간다. 눈여겨볼 점은 끊어질 수 있는 연결을 최소화하는 것이다. 소스가 바뀌어도 파이프라인 대부분은 그대로 남아 있어야 유지보수가 쉽다.

나의 기준으로 보면, 이 구조는 다음 원칙을 따른다. 업데이트가 없을 때는 아무 일도 하지 않는다. 실패가 나도 다음 주기에 스스로 회복한다. 사람이 개입해야 할 분기점은 알림으로 노출된다. 실무에서 실제로 지켜지지 않으면 주말에 로그를 까보며 시간을 버리게 된다. 자동화는 시간을 벌어줄 때만 가치가 있다.

RSS를 먼저, 다른 건 나중

RSS는 오래된 기술이지만 여전히 링크 자동화에서 왕이다. 이유는 단순하다. 변경이 생길 때마다 새 항목만 깔끔히 준다. 인증 없이 가져올 수 있고, 형식이 일정하다. 나는 가능한 모든 소스에서 RSS 주소를 찾는다. 보이지 않으면 HTML 헤더의 alternate 링크를 확인하거나, 서비스 도움말을 더 살핀다. 그래도 없을 때만 보조책을 쓴다.

두 가지 팁이 있다. 첫째, RSS가 하루 수백 개 항목을 쏟아내는 대형 피드라면 수집 주기를 짧게 가져가야 중복 위험이 줄어든다. 둘째, 제목이나 링크에 붙는 추적 파라미터를 초기에 제거할수록 향후 중복 제거가 깔끔해진다. UTM 파라미터 같은 것이 켜켜이 쌓이면 사실상 같은 문서를 서로 다른 링크로 간주하게 된다.

다음은 Python으로 여러 RSS를 합쳐 수집하는 최소 예제다. Feedparser는 오래되고 믿음직하다.

Import time Import hashlib Import json From urllib.parse import urlparse, urlunparse, parse_qsl, urlencode Import feedparser FEEDS = ["https://hnrss.org/frontpage", "https://www.reddit.com/r/programming/.rss", "https://martinfowler.com/feed.atom",] Def normalize_url(url): U = urlparse(url) # 추적 파라미터 제거 Q = [(k,v) for k,v in parse_qsl(u.query, keep_blank_values=True) If not k.lower().startswith("utm_") and k.lower() not in "ref","source"] Cleaned = u._replace(query=urlencode(q, doseq=True)) # 스킴, 호스트 소문자 Cleaned = cleaned._replace(scheme=cleaned.scheme.lower(), netloc=cleaned.netloc.lower()) Return urlunparse(cleaned) Def dedupe_key(entry): Url = normalize_url(entry.get("link","")) Title = (entry.get("title","") or "").strip().lower() Base = url or title Return hashlib.sha1(base.encode("utf-8")).hexdigest() Def collect(limit_per_feed=50): Seen = set() Items = [] For feed in FEEDS: D = feedparser.parse(feed) For e in d.entries[:limit_per_feed]: Key = dedupe_key(e) If key in seen: Continue Seen.add(key) Items.append("title": e.get("title","").strip(), "url": normalize_url(e.get("link","")), "published": time.strftime("%Y-%m-%dT%H:%M:%SZ", getattr(e, "published_parsed", time.gmtime())), "source": d.feed.get("title","unknown"), "summary": e.get("summary",""),) Return items If __name__ == "__main__": Data = collect() Print(json.dumps("items": data, "fetched_at": time.time(), ensure_ascii=False, indent=2))

이 스크립트는 여러 피드를 긁어와 중복을 제거하고 JSON으로 저장한다. 핵심은 dedupe_key다. URL을 정규화해 동일 문서를 하나로 본다. 제목 기반 보완은 피드에 링크가 비어 있는 특수 케이스를 커버하기 위함이다.

RSS가 없을 때 IFTTT를 꺼내는 법

IFTTT는 가볍고 쉽게 붙는다. RSS가 아예 없는 소스를 표현하기에도 괜찮고, Gmail이나 Slack 같은 내가 이미 쓰는 채널로 알림을 뿌릴 수도 있다. 사이트의 이메일 뉴스레터를 자동으로 수집해 링크를 추출하고, 내 저장소로 넘기는 시나리오를 자주 만든다.

예를 들어, 특정 뉴스레터의 새 메일이 오면 본문에서 링크 상위 3개를 뽑아 웹훅으로 보낸다고 하자. Gmail 트리거와 Webhooks 액션만으로 가능하다. 웹훅은 나의 수집 API 엔드포인트에 도달한다. IFTTT는 포맷팅 템플릿을 제공하므로, JSON 형태를 미리 맞춰두면 서버 측 파싱이 단순해진다.

다음은 내가 권하는 기본 설정 흐름이다.

- IFTTT에서 Gmail New email from search 트리거를 고르고, 뉴스레터 발신 주소를 검색 조건에 넣는다.
- 액션으로 Webhooks Make a web request를 선택한다.
- URL에는 나의 수집 엔드포인트를 입력한다. 예를 들어 https://links.example.com/ingest.
- Method는 POST, Content Type은 application/json, Body에는 제목, 발신자, 본문 요약, 본문 내 첫 링크 정도를 JSON 형식으로 넣는다.
- 테스트 메일로 흐름을 확인한 뒤 필터 코드가 필요하면 추가한다. 예를 들어 제목에 광고 표시가 있으면 무시하도록 한다.

이 다섯 단계만으로 뉴스레터 기반 링크 수집이 굴러간다. 여기서 조심할 점은 중복 방지다. 같은 메일이 여러 번 들어올 수 있으니 Message-ID 같은 헤더를 키로 삼아 중복 처리를 해야 한다. 또한 Gmail의 레이트 리밋에 걸리지 않도록 범위를 좁히는 것이 좋다.

하는 김에 Google Apps Script도

IFTTT 외에 Google Apps Script로 Gmail을 직접 스캔해도 된다. 스케줄러가 있어 하루 한 번, 혹은 매시간 실행할 수 있다. 장점은 비용이 없고, 필터링을 코드로 섬세하게 할 수 있다는 점이다. 단점은 구글의 실행 제한과 보안 승인 절차다.

Function fetchNewsletters() Var threads = GmailApp.search('from:(newsletter@example.com) newer_than:7d'); Var items = []; Threads.forEach(function(thread) Thread.getMessages().forEach(function(msg) \(AD\)/i)) return; Var body = msg.getBody(); Var urlMatch = body.match(/https?:\/\/[^\s"']+/g); If (!urlMatch) return; Items.push(Id: id, Subject: subject, Url: urlMatch[0], Date: msg.getDate().toISOString()););); Var payload = JSON.stringify(items: items); Var resp = UrlFetchApp.fetch('https://links.example.com/ingest', Method: 'post', ContentType: 'application/json', Payload: payload, MuteHttpExceptions: true); Logger.log(resp.getResponseCode());

이 코드는 일주일치 뉴스레터에서 첫 링크만 추출해 보낸다. 초안용으로 충분하고, 정교화는 서버 측에서 처리하는 편이 안전하다. 정규식으로 링크를 뽑을 때는 본문 포맷과 인코딩에 주의하자.

저장소는 가볍게, 포맷은 표준으로

데이터 저장은 화려할 필요가 없다. 많은 팀이 Notion 데이터베이스를 선호한다. 테이블이 눈에 보이고, 태그와 상태를 끌어다 놓기로 바꿀 수 있기 때문이다. 반면 자동화된 배포와 API 접근성은 제한이 있다. 구글 시트는 스크립트 접근이 쉬워 IFTTT와 자주 붙는다. 정적 사이트 배포를 염두에 둔다면 SQLite나 단일 JSON 파일도 충분하다. 무

엇을 고르든 두 가지는 공통으로 챙긴다. 항목의 고유 키를 정해 중복을 막고, 내보내기를 Atom 또는 JSON Feed 규격 중 하나로 제공해 외부 도구가 다시 소비할 수 있게 한다.

JSON Feed는 가벼운 선택지다. 스키마가 단순해 구현이 쉽다. 내 링크모음이 다른 서비스의 주소모음에 흘러들어 가길 원한다면 이 표준이 이점이 있다.

배포는 정적이 편하다

링크모음은 CRUD가 적다. 주기적으로 파일을 바꿔치기하면 충분하다. 그래서 정적 사이트가 편하다. GitHub Pages나 Cloudflare Pages 같은 무료 호스팅을 쓰면 운영비가 0에 가깝다. GitHub Actions로 수집 스크립트를 실행해 빌드된 HTML과 JSON Feed를 푸시하는 구조만 잡으면 된다.

예시로, 리포지토리 루트에 data/items.json을 갱신하고, Eleventy나 Astro 같은 정적 사이트 생성기가 이를 읽어 페이지를 만든다고 하자. Actions 워크플로는 이렇게 단순해진다.

Name: collect-and-build On: Schedule: - cron: "*/30 * * * *" Workflow_dispatch: Jobs: Build: Runs-on: ubuntu-latest Steps: - uses: actions/checkout@v4 - uses: actions/setup-python@v5 With: Python-version: '3.11' - run: pip install feedparser requests - run: python scripts/collect.py > data/items.json - run: npm ci - run: npm run build - name: Deploy Uses: peaceiris/actions-gh-pages@v3 With: Github_token: $ secrets.GITHUB_TOKEN Publish_dir: ./dist

30분마다 새로 수집하고 빌드해 배포한다. 스케줄은 소스의 발행 빈도를 보고 정한다. 너무 잦으면 외부 피드에 부담을 준다. 너무 드물면 신선도가 떨어진다.

중복, 스팸, 저품질을 잡아내는 기술적 습관

링크 자동화가 망가지는 첫 원인은 중복이다. 같은 글이 URL만 조금씩 달라져 들어온다. 아예 같은 글을 서로 다른 미러에서 수집할 수도 있다. 중복 키를 URL과 제목만으로 만들면 충분하지 않을 때가 많다. 본문 요약을 가볍게 정규화해 해시로 섞어두면 효과가 좋다. 단, 비용이 커지므로 일정 길이 이상의 텍스트에서만 적용하자.

스팸 억제는 키워드 블랙리스트와 도메인 신뢰도 점수의 조합이 간단하고 잘 먹힌다. 예를 들어 무료넷플릭스처럼 오해 소지가 큰 키워드는 기본적으로 경고 라벨을 달아 검수 대기열로 보낸다. 합법적 혜택을 소개하는 글일 수도 있지만, 경험상 클릭베이트 비율이 높다. 반대로 교육기관, 공공기관, 유명 기술 블로그 도메인에는 기본 신뢰 점수를 준다. 점수 합계가 특정 임계치를 넘지 못하면 공개 피드에는 노출하지 않는다.

언어 감지는 다국어 소스를 섞을 때 유용하다. Title과 summary에서 256자 정도만 샘플링해 fastText나 cld3 같은 라이브러리로 감지한다. 사이트의 주 언어와 다르면 태그를 붙이고 별도 페이지로 라우팅한다.

태그는 사람이, 규칙은 기계가

완전 자동 태깅은 원하는 결과가 잘 나오지 않는다. 하지만 80퍼센트까지는 규칙으로 끌어올릴 수 있다. 제목에 Kubernetes가 있으면 devops 태그를 붙이는 식의 규칙은 반복 성과가 뚜렷하다. 나머지 20퍼센트는 사람이 교정한다. 이때 UI가 중요하다. 내가 쓰는 방식은 공개 페이지에만 보이는 작은 수정 링크를 두고, 클릭하면 내부 편집 화면으로 이동해 태그를 고치게 한다. 수정 내역은 Git으로 남겨 회귀를 추적할 수 있다. 초기에 조금 귀찮아 보여도 일주일 지나면 링크모음의 질감이 달라진다.

간단한 정규화와 품질 점수 코드

아래는 수집된 항목에 간단한 점수를 매기는 예다. 도메인 신뢰도, 길이, 금칙어, 제목 품질로 합산한다. 이런 점수는 공개 여부의 절대 기준이 아니라, 검수 우선순위를 정하는 용도로 쓴다.

From urllib.parse import urlparse TRUST = "arxiv.org": 2.0, "developer.apple.com": 2.0, "cloud.google.com": 1.5, "medium.com": 0.5, # 품질 편차가 커서 낮게 BLOCK_WORDS = "click here", "무료넷플릭스", "무료 시청", "비트코인 투자", "카톡 오픈채팅" Def quality_score(item): Url = item["url"] Host = urlparse(url).netloc.lower() Score = TRUST.get(host, 1.0) Title = (item["title"] or "").strip() If any(bw in title.lower() for bw in BLOCK_WORDS): Score -= 3.0 If len(title) < 5: Score -= 1.0 If len(title) > 120: Score -= 0.5 If "?" In title and not any(x in title.lower() for x in ["how", "why", "what"]): Score -= 0.2 Return score

이 점수는 완벽하지 않지만, 상식적인 잡음을 거르는 데 실용적이다. 금칙어에는 서비스 맥락에 맞는 것들을 넣되, 오탐을 줄이기 위해 매주 리뷰한다. 특히 무료넷플릭스처럼 사용자 관심을 끄는 키워드는 무조건 차단하기보다 문맥을 확인하게 라벨만 붙여놓는 식으로 완충을 두는 편이 좋다.

프런트엔드에서 체감되는 차이

자동화의 엔진이 좋아도 이용자가 느끼는 품질은 프런트에서 갈린다. 타임라인이 한눈에 들어오고, 태그와 출처로 걸러볼 수 있어야 한다. 링크 카드에는 다음 세 가지 정보만 남겨둔다. 제목, 출처, 발행일. 요약은 마우스오버로만 보이게 하면 페이지가 가벼워진다. 모바일에서는 요약을 한 줄로 줄여 대체한다.

페이지 하단에는 내보내기 링크를 둔다. RSS, Atom, JSON Feed를 모두 제공하면 호환성이 넓어진다. 주소모음이나 다른 링크모음 서비스가 내 피드를 구독하게 만드는 관문이 된다. 여기서 피드의 URL은 절대 바꾸지 않는다. 바꾸면 외부 구독이 끊어진다. 어쩔 수 없이 바꿔야 한다면 구버전 피드에서 301 리디렉션을 걸어준다.

장애와 실패, 그리고 스스로 회복하는 법

이런 파이프라인을 몇 달 굴리다 보면 실패 패턴이 보인다. 외부 피드의 일시적 다운, DNS 문제, 인증서 만료, 내 저장소의 잠깐의 잠금 같은 것들이다. 대응은 두 겹으로 둔다. 첫째는 재시도 정책이다. HTTP 5xx는 지수 백오프로 3회 재시도, 4xx는 즉시 실패 처리, 타임아웃은 조금 더 빠른 주기 재시도. 둘째는 스냅샷이다. 이전 실행에서 성공적으로 수집된 데이터의 스냅샷을 남겨 주소모음 두었다가 실패 시 그대로 재배포한다. 최신성은 잠깐 떨어지지만 서비스가 완전히 멈추는 일은 막을 수 있다.

로그는 짧게. JSON 라인 포맷으로 남기고, GitHub Actions의 아티팩트 기능이나 CloudWatch 같은 곳에 하루치만 보관해도 충분하다. 경험상 로그가 과하면 아무도 안 본다.

윤리와 법적 경계

링크를 수집할 때는 robots.txt와 이용약관을 준수한다. 피드를 제공하는 서비스는 보통 사용 범위를 명시한다. 특히 상업적 사용 금지 조항이 있는 곳은 주의가 필요하다. 이메일 뉴스레터를 자동 처리할 때도 사전 동의와 보관 기간을 명확히 해둬야 한다. 스크래핑을 할 때는 요청 속도를 낮추고, 변칙적인 크롤링을 지양한다. 링크 카드에 원문의 요약을 담을 때는 과도한 발췌를 피한다. 저작권 문제로 이어질 수 있다.

광고와 추천 수수료 링크는 표시한다. 투명성이 신뢰다. 북마크에 제휴 링크가 섞일 수 있다면 disclosure를 사이트에 명시하고, 태그로도 구분해준다.

실전 시나리오 예: 개발자 위클리 링크모음

매주 월요일 오전 9시에 30개 내외의 링크를 내보내는 개발자 위클리를 상상해보자. 수집은 네 갈래다. 기술 블로그 RSS 20개, 오픈소스 릴리스 RSS 5개, 뉴스레터 3종의 이메일, 트위터 북마크 수동 선택 5개. 자동화는 RSS에서 1차 후보를 모으고 품질 점수로 상위 60개를 고른다. 이메일에서 올라온 링크 중 광고 의심 키워드를 포함한 것은 검수 대기열로 보내고, 트위터 북마크는 사람이 고른 만큼 가중치를 주어 상단에 배치한다.

월요일 7시 수집이 끝나면 편집자는 검수 대기열에서 10분 정도 손을 본다. 제목을 간단히 다듬고 태그를 교정한다. 7시 30분에 빌드가 돌아가 정적 사이트가 업데이트된다. JSON Feed와 RSS가 새로 생성되어, 사내 슬랙 봇이 채널에 자동으로 알린다. 9시에 뉴스레터로도 발송된다. 이 구조로 운영하면 공휴일에도 시스템이 돌아가고, 필요하면 편집을 0으로 줄여 완전 자동 발행으로 돌릴 수 있다.

IFTTT와 Slack, Notion을 얹는 작은 팁

IFTTT로 Slack에 올릴 때는 채널 하나를 큐로 쓰는 전략이 편리하다. 예를 들어 #links-queue에 올라온 메시지는 모두 수집기로 흘러간다. 슬랙 메시지에 이모지로 태그를 지정하는 것도 쓸 만하다. :fire:는 트렌딩, :book:은 튜토리얼 식으로 매핑한다. 수집기는 메시지의 리액션을 읽어 태그로 반영한다.

Notion 데이터베이스를 저장소로 쓰면 API 속도가 병목이 된다. 페이징 단위가 작고, 속성이 많으면 느려진다. 해결은 단순하다. 링크 원본, 제목, 태그, 출처, 발행일, 점수 정도의 핵심 속성만 API로 건드리고, 부가 정보는 JSON 파일로 별도 호스팅한다. 상세 페이지에서 JSON을 비동기로 읽어 보완하면 체감 속도가 확 좋아진다.

사소하지만 큰 차이를 만든 운영 습관

백업을 잊지 않는다. JSON Feed와 데이터베이스를 하루 한 번 덤프해 S3나 GDrive에 저장한다. 삭제는 연성 삭제로 처리한다. Is_deleted 플래그를 두고, 실제 삭제는 30일 후 배치로 한다. 초기에 가끔 잘못된 규칙이 대량의 항목을 지우는 사태가 생긴다. 롤백이 쉽도록 설계해두면 침착하게 대응할 수 있다.

그리고, 사람이 정한 규칙은 주기적으로 검토한다. 금칙어 리스트는 분기마다, 도메인 신뢰 점수는 반기마다 점검한다. 처음 설정이 영원히 맞을 리 없다. 제품이 변하고 업계가 변한다. 예전에 스팸이라고 생각했던 도메인이 품질을 끌어올리는 경우도, 그 반대도 늘 있다.

최소한의 설계 체크리스트

- 소스는 RSS 우선, 보조로 이메일 파서와 웹훅을 둔다.
- URL 정규화와 중복 키 전략을 문서화하고 코드로 강제한다.
- 신뢰도 점수와 금칙어 라벨링으로 검수 대기열을 만든다.
- 정적 배포와 표준 피드 제공을 기본으로 한다.
- 백업과 롤백 경로를 운영 시작 전에 준비한다.

이 다섯 가지만 지켜도 링크모음의 수명을 몇 배는 늘릴 수 있다.

주소 체계와 슬러그, 그리고 링크의 반감기

주소모음 페이지의 URL 체계는 단순해야 한다. 날짜 기반 경로는 좋다. /2026/05/ 같은 경로로 월간 페이지를 쌓아두면 아카이브 탐색이 쉬워진다. 개별 항목은 해시 슬러그로 충분하다. 제목 기반 슬러그는 언어와 인코딩 이슈로 자주 깨진다. 해시는 중복 감지 키와 공유할 수도 있어 디버깅에 유리하다.

링크의 반감기는 체감상 6개월에서 18개월 사이다. 튜토리얼과 릴리스 노트는 빨리 낡고, 이론과 개념 글은 오래 간다. 그래서 홈 화면에는 지난 30일의 상위 항목만, 주제별 아카이브에는 누적 상위 항목을 보여준다. 오래 살아남는 링크는 홈에서 내려가도 검색으로 잘 잡힌다.

데이터 노출과 개인정보

수집 중 이메일 메타데이터나 쿠키가 끼어 들어오지 않도록 주의한다. IFTTT로 메일을 파싱할 때 발신자 이름이나 사내 계정 정보가 원치 않게 기록될 수 있다. 로그에서 이런 정보를 가려내고, 공개 피드에는 실수로도 포함되지 않

게 한다. 링크 카드의 썸네일을 외부에서 프록시 없이 바로 불러오면 사용자 IP가 노출된다. 프록시 이미지를 쓰거나, 썸네일을 자체적으로 캐시한다.

마이그레이션을 두려워하지 않는 설계

도구는 바뀐다. IFTTT가 유료화 정책을 재조정하거나, 특정 피드가 종료될 수도 있다. 그래서 내부 표현을 중립적으로 둔다. 수집은 어떤 도구가 오든 items라는 배열과 동일한 필드를 만든다. 나머지는 어댑터다. IFTTT 어댑터, Apps Script 어댑터, RSS 파서 어댑터. 이렇게 쪼개두면 일부가 무너져도 전체는 유지된다. 실무에서 이 유연성이 운영자의 멘탈을 지킨다.

마감의 감각

마지막으로, 자동화에도 마감이 필요하다. 링크를 무기한 쌓아두면 품질이 흐려진다. 매주 혹은 매월, 상위 N개만 공개로 유지하고 나머지는 아카이브로 내린다. 큐레이션은 버리는 기술이 절반이다. 링크모음의 맛은 신선함과 밀도에서 온다. 기계는 신선함을, 사람은 밀도를 책임지면 역할 분담이 깔끔해진다.

이 정도 구성을 갖추면, 링크모음은 매일 저절로 살아 움직인다. 주소모음 형태든, 주간 뉴스레터든, 사내 지식 허브든, 변하는 것은 소스와 배포 경로뿐이다. RSS와 IFTTT로 뼈대를 잡고, 작고 단단한 스크립트로 살을 붙이면 된다. 시간을 먹는 반복 작업을 기계에 맡기고, 사람은 더 어렵고 가치 있는 판단에 집중한다. 그렇게 오래 가는 링크모음이 완성된다.