



이해

ETL

데이터 파이프라인
최신 데이터 아키텍처

칭찬



databricks

매트 팔머 매트 팔머

@ 데이터브릭

델타 라이브 테이블

안정적인 데이터 파이프라인이 쉬워졌습니다.

DLT(Delta Live Tables)는 안정적인 데이터 파이프라인을 구축하기 위해 간단한 선언적 접근 방식을 사용하는 최초의 ETL 프레임워크입니다. DLT는 규모에 맞게 인프라를 자동으로 관리하므로 데이터 분석가와 엔지니어는 도구 사용에 소요되는 시간을 줄이고 데이터에서 가치를 얻는 데 집중할 수 있습니다.



c ::; B



@]

ETL 개발 가속화

자동으로 데이터에 대한 확신을 가지세요
인프라를 관리하세요

배치 및 스트리밍 단순화

더 알아보기



ETL 이해

최신 데이터를 위한 데이터 파이프라인
아키텍처

매트 팔머

베이징·보스턴 파님 세바스토풀 도쿄·

O'REILLY®

Matt Palmer의 ETL 이해

저작권 © 2024 O'Reilly Media, Inc. 모든 권리 보유.

미국에서 인쇄되었습니다.

출판사: O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly 도서는 교육, 비즈니스 또는 판매 홍보용으로 구입할 수 있습니다. 대부분의 타이틀에 대해 온라인 버전도 제공됩니다 (<http://oreilly.com>). 자세한 내용은 기업/기관 영업부(800-998-9938 또는 Corporate@oreilly.com)에 문의하세요.

인수 편집자: Aaron Black

개발 편집자: Gary O'Brien

프로덕션 편집자: 크리스틴 브라운

편집자: nSight, Inc.

교정자: M&R Consultants Corporation

인테리어 디자이너: David Futato

표지 디자이너: Ellie Volckhausen

일러스트레이터: 케이트 덜리아

2024년 3월:

초판

초판 개정 내역

2024-03-13: 첫 번째 릴리스

<http://oreilly.com/catalog/errata.csp?isbn=9781098159252>를 참조하세요. 릴리스 세부정보를 확인하세요.

O'Reilly 로고는 O'Reilly Media, Inc.의 등록 상표입니다. ETL 이해, 표지 이미지 및 관련 상품 외장은 O'Reilly Media, Inc.의 상표입니다.

본 저작물에 표현된 견해는 저자의 견해이며 출판사의 견해를 대변하지 않습니다. 출판사와 저자는 이 저작물에 포함된 정보와 지침이 정확한지 확인하기 위해 선의의 노력을 기울였으나, 출판사와 저자는 본 저작물의 사용으로 인해 발생하는 손해에 대한 책임을 포함하되 이에 국한되지 않고 오류나 누락에 대한 모든 책임을 부인합니다. 이 작업에 의존합니다. 이 저작물에 포함된 정보와 지침을 사용하는 데 따른 위험은 전적으로 귀하의 책임입니다. 본 저작물에 포함되거나 설명된 코드 샘플 또는 기타 기술이 오픈 소스 라이선스 또는 타인의 지적 재산권의 적용을 받는 경우, 이를 사용하는 것이 그러한 라이선스 및/또는 권리를 준수하는지 확인하는 것은 귀하의 책임입니다.

이 작업은 O'Reilly와 Databricks 간의 협력의 일부입니다. 편집 독립성 선언문을 참조하세요 .

978-1-098-15923-8

[LSI]

목차

소개.....	vii
1. 데이터 수집.....	1
데이터 수집 - 현재와 과거	2
소스 및 타겟	2
섭취 고려사항	10
솔루션 선택	19
2. 데이터 변환.....	25
데이터 변환이란 무엇입니까?	25
혁신 솔루션 구축 데이터 혁신의 미래	33
	40
3. 데이터 오케스트레이션.....	43
데이터 오케스트레이션이란 무엇입니까?	44
데이터 오케스트레이션 도구 설계 패턴	47
턴 및 모범 사례 데이터 오케스트레이션의 미래	54
	58
4. 파이프라인 문제 및 문제 해결.....	61
유지보수성 62	
오류 모니터링 및 벤치마킹으로 작업 흐름 개선	63
	68
	70

5. 효율성과 확장성.....	77
효율성과 확장성의 정의	78
환경 이해	79
<u>프로세스 효율성</u>	88
결론	92
결론.....	93

소개

귀하의 직위가 데이터 엔지니어이든 다른 데이터 중심 직업(분석가 및 과학자)이든 ETL이 라는 용어를 들어보셨을 것입니다. 비록 당신이 알지 못하더라도 ETL이 당신 삶의 일부가 될 가능성이 높습니다!

추출, 변환, 로드의 약어인 ETL은 소스 시스템에서 데이터를 가져와서 필요에 맞게 변경하고 대상에 로드하는 등 대부분의 데이터 실무자가 수행하는 기본 워크플로를 설명하는 데 사용됩니다.

제품 리더가 데이터 기반 결정을 내릴 수 있도록 돋고 싶으십니까? ETL은 보고서에 중요한 테이블을 구축합니다. 팀의 기계 학습 모델의 다음 반복을 훈련하고 싶으십니까? ETL은 고품질 데이터 세트를 생성합니다. 규정 준수 요구 사항을 충족하기 위해 회사의 스토리지 정책을 더 체계적이고 엄격하게 적용하려고 하시나요?

ETL은 워크플로에 프로세스, 계보 및 관찰 가능성을 제공합니다.

데이터로 무엇이든 하려면 안정적인 프로세스나 파이프라인이 필요합니다. 이러한 근본적인 사실은 기존 비즈니스 인텔리전스(BI) 워크로드부터 대규모 언어 모델(LLM) 및 AI와 같은 최첨단 기술까지 적용됩니다.

AI의 멋진 신세계

데이터 세계에서는 많은 추세가 왔다가 사라지는 것을 보았습니다. 일부는 공간을 변화시켰고 일부는 일시적인 유행으로 판명되었습니다. 가장 최근에는 의심할 여지 없이 생성적 AI(Generative AI)가 있습니다.

매 순간마다 AI, LLM 및 챗봇에 대한 대화가 이어집니다. OpenAI의 ChatGPT 출시로 인해 AI에 대한 최근의 관심이 언론의 관심을 넘어 사람들 사이에서 확대되고 있습니다.

연구자—이제 많은 사람들이 이를 필수적인 전략적 투자로 보고 있습니다. 누가 뒤처지길 원할까요?

LLM의 진정한 가치는 깨끗하고 선별된 데이터 세트에 대한 임베딩 또는 미세 조정 모델에서 비롯됩니다. 이러한 기술을 사용하면 환각과 같은 일반적인 오류를 피하면서 영역별 지식을 갖춘 모델을 만들 수 있습니다.

물론, 의미 있는 임베딩은 짐작할 수 있듯이 깨끗한 데이터세트에서 파생됩니다. 그런 의미에서 AI는 데이터 변환을 기반으로 구축됩니다.

성공 여부는 대규모로 일관된 고품질 데이터 세트를 생성하는 능력에 크게 좌우됩니다. 데이터는 단일 위치에서 이동, 변형 및 병합되어야 합니다. 즉, 추출, 변환 및 로드가 가능합니다.

그렇습니다. 심지어 가장 최첨단 기술의 뿌리도 ETL에 뿌리를 두고 있습니다.

변화하는 데이터 환경

최근 생성 AI의 급증 외에도 지난 10년 동안 다른 추세로 인해 데이터 환경이 재편되었습니다. 그러한 추세 중 하나는 스트리밍 데이터의 중요성이 커지고 있다는 것입니다. 이제 기업은 센서, 웹사이트, 모바일 애플리케이션 등을 통해 방대한 양의 실시간 데이터를 생성하고 있습니다. 이러한 변화로 인해 즉각적인 의사 결정을 위해서는 실시간 데이터 수집 및 처리가 필요합니다. 따라서 데이터 엔지니어는 대용량 스트리밍 데이터를 처리할 수 있는 연속 파이프라인을 구축하고 관리하기 위해 기존 일괄 처리를 뛰어넘어 확장해야 합니다.

또 다른 주목할 만한 발전은 **데이터 레이크 하우스 아키텍처**의 출현입니다. 데이터 레이크 하우스는 데이터 웨어하우스와 데이터 레이크의 기능을 통합하려는 새로운 개념을 나타냅니다.

Delta Lake 와 같은 새로운 스토리지 기술을 활용하여 데이터 레이크의 신뢰성과 성능을 향상시키는 레이크하우스 모델은 비용 효율적이고 확장 가능한 데이터 레이크 스토리지와 데이터 웨어하우스의 효율적인 트랜잭션 처리를 결합합니다. 이러한 합병을 통해 단일 프레임워크 내에서 AI 워크로드(일반적으로 데이터 레이크에서 처리됨)와 분석 워크로드(일반적으로 데이터 웨어하우스에서 수행됨)를 모두 실행할 수 있습니다. 이러한 통합은 병렬 아키텍처 유지, 일관된 데이터 거버넌스 보장 및 데이터 복제 관리와 관련된 복잡성을 크게 줄입니다.

ETL은 데이터 관리 분야에서 오랫동안 사용되어 온 개념이지만 현대 데이터 환경에서도 그 관련성은 여전히 줄어들지 않습니다. 이제 중요한 고려 사항은 ETL 프로세스가 배치 및 스트리밍 데이터를 모두 포함하도록 적용할 수 있는 방법과 ETL 프로세스가 데이터 레이크하우스 아키텍처 내에 효과적으로 통합될 수 있는 방법입니다. 이 가이드의 목표는 이러한 측면을 조명하여 이러한 발전 추세에 비추어 ETL을 이해하는 데 도움을 주는 것입니다.

ELT(및 기타 맛)는 어떻습니까?

데이터 엔지니어링을 자세히 살펴보면 ETL 외에 ELT와 같은 용어를 접할 수 있습니다. "와, 이 사람들은 교정자를 고용해야지"라고 생각할 수도 있지만 실제로는 다릅니다.

자귀.

ELT의 주요 차이점은 순서에 있습니다. ELT에서는 모든 것이 준비 리소스에 로드된 다음 다운스트림으로 변환됩니다. 많은 사람들이 "스토리지가 저렴하다"고 말하는 것처럼 ELT는 점점 더 표준이 되어 많은 상황에서 ETL을 대체하고 있습니다. "ETL"이라는 용어는 오랫동안(데이터베이스 자체가 생성된 이후) 널리 사용되어 ELT가 더 정확하더라도 여전히 일반적으로 참조됩니다.

이제 우리는 클라우드 스토리지 비용 절감과 데이터 생성 용이성으로 촉진되는 "선 저장, 나중에 조치" 시대에 있습니다.

분석을 위해 잠재적으로 유용한 모든 데이터를 유지하는 것이 일반적입니다. 메달리온 아키텍처 및 데이터 레이크하우스와 같은 기술 발전은 수운 스키마 진화 및 시간 여행과 같은 기능을 통해 이러한 접근 방식을 지원합니다. 이 가이드 전반에 걸쳐 이에 대해 논의할 것입니다.

우리는 주로 "ETL"이라는 용어를 사용하지만 논의된 원칙과 고려 사항은 ETL과 ELT뿐만 아니라 역방향 ETL(정리된 데이터를 다시 비즈니스 도구에서 웨어에서 수집하는 방식)과 같은 다른 변형에도 적용할 수 있다는 점에 유의하는 것이 중요합니다. - 아니면 호숫가. 아니요, 역 ETL != LTE입니다. 그렇습니다. 혼란스럽습니다. 하지만 여기서는 다른 방향으로 나아갑니다.

"ETL"이라는 용어가 현재 프로세스를 정확하게 설명하는지 여부에 관계없이 데이터 수집, 변환 및 조정의 기본 사항을 이해하는 것은 여전히 중요합니다. 이는 또한 관찰 가능성, 문제 해결, 확장 및 최적화와 같은 영역의 모범 사례로 확장됩니다. 구하가 사용하는 특정 데이터 처리 방법에 관계없이 이 가이드가 귀중한 리소스가 되기를 바랍니다.

오라일리 온라인 학습

40년이 넘는 세월 동안 **오라일리 미디어(O'Reilly Media)**는 기업의 성공을 돋기 위해 기술 및 비즈니스 교육, 지식, 통찰력을 제공했습니다.

전문가와 혁신가로 구성된 우리의 독특한 네트워크는 책, 기사, 온라인 학습 플랫폼을 통해 자신의 지식과 전문성을 공유합니다. O'Reilly의 온라인 학습 플랫폼을 사용하면 실시간 교육 과정, 심층 학습 경로, 대화형 코딩 환경, O'Reilly 및 200개 이상의 기타 출판사가 제공하는 방대한 텍스트 및 비디오 컬렉션에 대한 주문형 액세스를 제공합니다. 자세한 내용은 <https://oreilly.com>을 방문하세요.

저희에게 연락하는 방법

이 책에 관한 의견과 질문은 출판사에 문의하십시오.

오라일리 미디어, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-889-8969(미국 또는 캐나다) 707-827-7019(국제
또는 현지) 707-829-0104(팩스)
support@oreilly.com
<https://www.oreilly.com/about/contact.html>

이 책의 웹페이지에는 정오표, 예제 및 추가 정보가 나열되어 있습니다. 이 페이지는 <https://oreil.ly/understandingETL>에서 액세스할 수 있습니다.

도서 및 강좌에 대한 뉴스와 정보를 보려면 <https://oreilly.com>을 방문하세요.

LinkedIn에서 우리를 찾아보세요: <https://linkedin.com/company/oreilly-media>.

YouTube에서 시청하세요: <https://youtube.com/oreillymedia>.

감사의 말

우리 모두는 개인의 어깨 위에 서 있지만, 특히 이 가이드는 매우 헌신적이고 배려하는 일부 개인의 멘토링, 도움 및 지원 없이는 불가능했을 것입니다.

먼저 O'Reilly와 Databricks의 파트너들에게 감사드립니다. 저에게 글을 쓸 기회를 준 Aaron Black; 뛰어난 개발 편집자이자 절친한 친구였던 Gary O'Brien; 가이드 전체를 구성하는 데 도움을 준 Ori Zohar; 훌륭하고 세심한 기술 리뷰를 해주신 Sumit Makashir와 Pier Paolo Ippolito도 감사드립니다.

스트리밍 및 스트림 처리를 이해하는 데 도움을 준 Zander Matheson에게 감사드립니다. 놀라운 도구 (Bytewax)를 개발하는 동시에 Zander는 훌륭한 친구이자 일반 데이터 전문가였습니다.

나의 첫 번째 가이드를 작성하기 위해 협력하고 나에게 배우고 성장할 수 있는 충분한 기회를 제공한 Aleks Tordova와 Coalesce 팀에 감사드립니다.

나의 결점, 특이성 및 일반적인 말장난에도 불구하고 데이터와 삶의 여정에 무조건적인 지원을 제공한 가족에게 감사드립니다. 감사합니다, Jasmine, Violet, Paul(그리고 강아지 Enzo와 Rocky!)

다음으로, 제가 전국으로 이사하고, 새 직장을 구하고, 이 가이드를 작성하고, 자아 발견의 길을 계속할 때 저를 지원해 준 놀라운 친구들이 있다는 축복을 받았습니다. 힘든 시기를 헤쳐나가는 데 도움이 된 문자, 슬랙스, 전화, 밍이 많았습니다.

알파벳순으로 JulieAnn, Kandace, Rob, Srini, Tyson에게 감사드립니다.

마지막으로 데이터 커뮤니티에 감사드립니다. 오픈 소스에 기여하고 컨퍼런스에 참석하는 개인, 개선을 위해 매일 깨어나는 실무자, 현장에서 우리를 계속 발전시키는 교육자/멘토, 텍스트, 아이디어 및 콘텐츠는 우리가 현재의 위치에 도달하는 데 도움이 되었습니다. 다음에 우리가 무엇을 성취할지 빨리 보고 싶습니다!

제1장

데이터 수집

본질적으로 데이터 수집에는 소스에서 지정된 대상으로 데이터를 전송하는 작업이 포함됩니다. 주요 목표는 준비, 처리, 분석 및 인공 지능/기계 학습(AI/ML)을 위해 준비된 환경으로 데이터를 안내하는 것입니다. 대규모 조직은 내부적으로(팀 간) 데이터를 이동하는 데 중점을 둘 수 있지만, 우리 대부분의 경우 데이터 수집은 외부 소스에서 데이터를 가져와 내부 대상으로 전달하는 것을 강조합니다.

비즈니스와 제품 개발 모두에서 데이터가 핵심적인 중요성을 갖는 시대에 정확하고 시의적절한 데이터의 중요성은 아무리 강조해도 지나치지 않습니다. 이렇게 데이터에 대한 의존도가 높아짐에 따라 팀은 정보를 추출하여 의사 결정 프로세스를 개선하고 뛰어난 제품을 제작하며 다양한 기타 작업을 수행할 수 있는 다양한 "소스"가 생겨났습니다. 예를 들어 마케팅 팀은 Meta, Google(광고 및 분석 포함), Snapchat, LinkedIn 및 Mailchimp와 같은 여러 광고 및 분석 플랫폼에서 데이터를 검색해야 합니다.

그러나 시간이 지남에 따라 API와 데이터 소스는 수정됩니다. 열이 추가되거나 제거될 수 있고 필드 이름이 변경될 수 있으며 새 버전이 오래된 버전을 대체할 수 있습니다. 단일 소스의 변경 사항을 처리하는 것이 가능할 수 있지만 5개, 10개, 심지어 100개 등 여러 소스의 변경 사항을 저글링하는 것은 어렵습니까? 시급한 과제는 "신진 데이터 팀이 어떻게 일관되고 확장 가능한 방식으로 이러한 다양한 소스를 효율적으로 처리할 수 있는가?"입니다. 데이터 엔지니어로서 우리의 평판을 어떻게 보장할 수 있습니까?

특히 모든 부서의 요구가 지속적으로 증가하는 경우 안정적이고 간단한 데이터 액세스를 제공할까요?

데이터 수집 - 현재와 과거

설치 원칙은 대체로 동일하지만 많은 부분이 변경되었습니다. 데이터의 양, 속도, 다양성이 발전함에 따라 우리의 방법도 발전해야 합니다.

이를 수용하기 위해 업계에서 클라우드로의 이동, 웨어하우스에서 데이터 레이크, 레이크하우스로의 이동, 스트리밍 기술의 단순화 등 다양한 업계 변화가 있었습니다.

이는 추출-변환-로드에서 전환으로 나타났습니다.

(ETL) 워크플로우를 추출-로드-변환(ELT)으로 전환합니다. 주요 차이점은 이제 모든 데이터가 대상 시스템에 로드된다는 것입니다. 우리는 2 장에서 변화의 맥락에서 이러한 환경을 논의할 것입니다.

우리는 ETL과 ELT라는 용어에 대해 너무 현학적인 표현을 자제합니다. 그러나 우리는 거의 모든 최신 데이터 엔지니어링 워크플로가 거의 모든 데이터를 클라우드에 스테이징하는 작업을 포함한다는 점을 강조하고 싶습니다.

주목할만한 예외는 수백 조 행의 고도로 세분화된 데이터(예: 사물 인터넷(IoT) 또는 센서 데이터)가 매일 처리되는 경우이며, 준비하기 전에 데이터를 집계하거나 삭제하는 것이 합리적입니다.

끊임없는 변화에도 불구하고 추출의 근본적인 진실은 데이터가 소스에서 가져와서 대상에 기록된다는 것입니다. 그러므로 추출에 관한 논의는 바로 여기에 집중되어야 한다.

소스 및 타겟

대부분의 경우 섹션을 추출과 연관시키지만 로딩과도 밀접하게 연관되어 있습니다. 결국 모든 소스에는 대상이 필요합니다. 이 가이드에서는 구하가 웨어하우스나 데이터 레이크를 구축했다고 가정합니다. 따라서 저장은 이 장의 주요 주제가 아닙니다. 대신, 스테이징 모범 사례와 이상적인 스토리지 구현의 특징을 모두 강조하겠습니다.

"완벽한" 솔루션이 존재하지 않을 수도 있다는 점을 염두에 두고 아키텍처 설계를 위한 툴킷을 제공하는 것이 우리의 임무입니다. 소스를 평가하고 데이터 수집의 고유한 매듭을 풀기 위한 프레임워크를 살펴보겠습니다. 우리의 높은 수준의 접근 방식은 풍경에 대한 조감도를 제공하여 정보에 입각한 적절한 결정을 내릴 수 있도록 설계되었습니다.

소스

데이터 수집에 대한 주요 고려 사항은 소스와 해당 특성입니다. 아주 운이 좋지 않은 이상 소스는 많이 있을 것입니다. 적절한 리소스를 확보하고 수집 솔루션에 대한 기준을 설정하기 위해 각각을 별도로 평가해야 합니다.

엄청난 양의 데이터 원본과 비즈니스 요구 사항의 특성으로 인해(소스를 제거하라는 요청을 받은 적이 거의 없지만 소스를 추가하는 것은 또 다른 목요일일 뿐입니다), 적합하지 않은 하나 이상의 소스를 접하게 될 가능성이 높습니다. 단일 솔루션, 신뢰를 쌓는 데는 몇 주, 몇 달, 몇 년이 걸리지만 하루 만에 무너질 수도 있습니다. 안정적이고 시기적절한 섭취가 가장 중요합니다. 그렇다면 소스를 선택할 때 중요한 것은 무엇입니까?

소스 검토 실용적

인 가이드로서 우리는 소스 데이터의 특성과 비즈니스 가치를 얻는 방법을 이해하는 데 도움이 되는 오랜 시간에 걸쳐 검증된 질문을 제시하는 접근 방식을 취합니다.

매우 중요한 입장을 취하는 것이 좋습니다. 소스 데이터가 필요하지 않거나 다른 소스가 비즈니스에 더 적합할 가능성이 항상 있습니다. 귀하는 조직의 데이터 전문가이며 가정을 확인하고 다시 확인하는 것이 귀하의 임무입니다. 행동과 복잡성에 대한 편견이 있는 것은 정상이지만 본질주의와 단순성을 반드시 고려해야 합니다.

소스를 조사할 때 업스트림 데이터에 대해 소프트웨어 엔지니어와 함께 작업할 가능성이 높지만 다운스트림 고려 사항도 마찬가지로 중요하다는 점을 명심하세요. 이러한 사항을 무시하면 비용이 많이 들 수 있습니다. 몇 주 동안 작업을 수행할 때까지 오류가 나타나지 않을 수 있기 때문입니다.

물어볼 질문

우리는 누구와 함께 일할 것인가?

인공지능 시대에 우리는 실제 지능을 최우선으로 생각합니다.

모든 데이터 파이프라인에서 가장 중요한 부분은 서비스를 제공할 사람들입니다. 관련된 이해관계자는 누구입니까? 주요 동기는 무엇입니까? OKR(목표 및 핵심 결과) 또는 조직의 명령은 인센티

브를 조정하고 프로젝트를 빠르게 진행하는 데 유용할 수 있습니다.

데이터는 어떻게 사용되나요?

"누가"와 밀접하게 연관되어 데이터를 어떻게 사용할 것인지는 후속 결정의 자침이 될 것입니다. 이는 이해관계자의 요구 사항을 확인하고 이해관계자가 해결하려고 하는 "문제 뒤에 있는 문제"를 배우는 방법입니다. 모호함을 피하기 위해 기술적인 예/아니요 요구 사항 목록을 적극 권장합니다.

빈도는 얼마나 됩니까?

나중에 자세히 설명하겠지만 대부분의 실무자는 즉시 일괄 처리와 스트리밍으로 전환합니다. 모든 데이터는 배치 또는 스트림으로 처리될 수 있지만 일반적으로 스트리밍하려는 데이터의 특성을 말합니다. 우리는 데이터가 경계가 있는지 없는지, 즉 데이터가 끝나는지(예: 2020년 인구 조사 미국 지역사회 조사 데이터 세트), 아니면 연속적인지(예: 광섬유 캐비닛의 로그 데이터) 먼저 고려하는 것을 응호합니다.

한계를 고려한 후 사용 가능한 최소 주파수에 따라 소스에서 가져올 수 있는 빈도에 대한 엄격한 제한이 설정됩니다. API가 매일 업데이트되는 경우 보고 빈도에 엄격한 제한이 있습니다. 경계, 속도 및 비즈니스 요구 사항에 따라 데이터 추출 빈도가 결정됩니다.

예상되는 데이터 양은 얼마입니까?

데이터 볼륨은 더 이상 데이터 저장 능력을 제한하는 요소가 아닙니다. 결국 "스토리지는 저렴하고" 컴퓨팅 비용은 비쌀 수 있지만 그 어느 때보다 저렴합니다(수백만 배, 그림 1-1 및 1 참조). -2). 그러나 볼륨은 데이터 작성 및 처리 방법과 원하는 솔루션의 확장성을 밀접하게 알려줍니다.

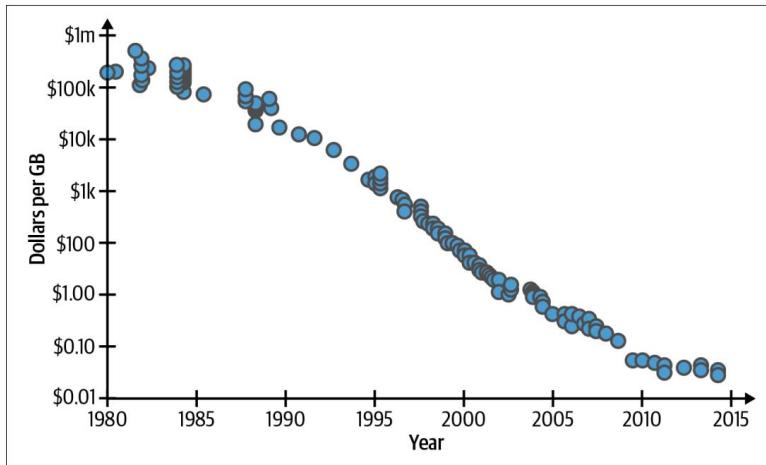


그림 1-1. 1980년부터 2015년까지 GB당 하드 드라이브 비용(출처: Matt Komorowski) y축 값은 로그 스케일입니다.

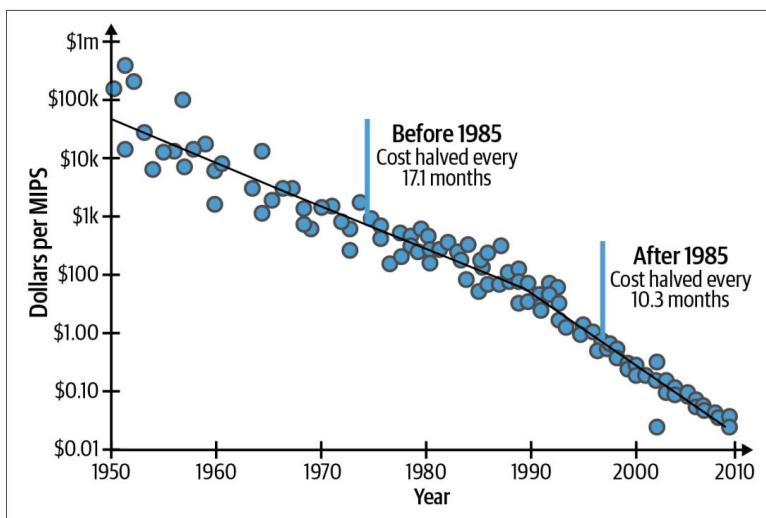


그림 1-2. 컴퓨팅 비용, 초당 수백만 명령(MIPS)(출처: Field Robotics Center) y축 값은 로그 스케일입니다.

형식은 무엇입니까?

결국 저장 형식을 선택하게 되지만 입력 형식은 중요한 고려 사항입니다. 데이터는 어떻게 전달되나요? API를 통한 JSON(JavaScript Object Notation) 페이로드를 통해 이루어지나요? 아마도 FTP 서버일까요? 운이 좋다면 그렇죠

이미 관계형 데이터베이스 어딘가에 살고 있습니다. 스키마는 어떤 모습인가요? 스키마가 있나요? 끝없이 많은 데이터 형식으로 인해 우리의 생계는 흥미로워지지만 동시에 어려움도 따릅니다.

품질은 어떻습니까?

데이터 세트의 품질에 따라 변환이 필요한지 여부가 크게 결정됩니다. 데이터 엔지니어로서 사용자를 위한 일관된 데이터 세트를 보장하는 것이 우리의 임무입니다. 누락된 특성을 보완하기 위해 데이터를 과도하게 처리하거나 외부 소스에서 강화해야 할 수도 있습니다.

우리는 마지막 질문에 답하기 위해 이러한 특성을 사용할 것입니다.

데이터는 어떻게 저장되나요?

앞서 언급했듯이 이 책에서는 데이터의 고정된 목적지를 가정합니다. 그럼에도 불구하고 데이터 스토리지에는 몇 가지 주요 고려 사항이 있습니다. 스테이지를 준비할지 여부(정말 질문인가요?), 비즈니스 요구 사항 및 이해 관계자 적합성이 가장 중요합니다.

소스 체크리스트

만나는 모든 소스에 대해 다음 안내 질문을 고려하십시오.

소스 수가 쌓이면 어려워 보일 수도 있지만 기억하세요. 이것은 글쓰기 작업이 아닙니다. 이는 각 소스의 과제를 해결하고 목표를 달성하는 적절한 솔루션을 스케치하는 데 도움이 되는 프레임워크입니다.

반복적으로 느껴질 수도 있지만 이 기초 작업은 장기적인 시간과 리소스를 절약해 줍니다.

질문	예시
는 누구와 협력할 것인가? 엔지니어링(결제)	
데이터는 어떻게 사용되나요?	재무 보고 및 분기별 전략 수립
소스가 여러개인가요?	예
형식은 무엇입니까?	반구조화된 API(스트라이프 및 내부)
빈도는 얼마나 됩니까?	시간별
볼륨은 얼마나 됩니까?	약 1,000개의 새 행/일, 기준 폴 ~100,000개 포함
어떤 처리가 필요합니까?	열 이름 변경, 보조 소스로부터의 강화 등 데이터 정리
데이터는 어떻게 저장되나요?	Databricks를 통해 Delta 테이블에 준비된 데이터 저장

목적지

엔드투엔드 시스템에는 바로 그러한 가상적인 고려 사항이 필요하지만 대부분의 독자는 스토리지 기술이 이미 선택되어 있는 기존 시스템에 파이프라인을 구축하는 임무를 맡게 될 것이라고 가정합니다.

데이터 스토리지 기술을 선택하는 것은 이 가이드의 초점 범위를 벗어나지만, 데이터 시스템이 창출하는 전체 가치에 매우 중요하므로 대상을 간략하게 살펴보겠습니다. 따라서 목적지를 분석(또는 고려)할 때 소스와 유사한 체크리스트를 사용하는 것이 좋습니다. 일반적으로 소스보다 대상이 훨씬 적으로 연습이 훨씬 간단합니다.

목적지 조사 목적지의 주

요 차별화 요소는 이해관계자가 우선시된다는 것입니다. 목적지에는 독특한 특성이 있습니다. 즉, 이해관계자를 중심으로 움직입니다. 이러한 대상은 BI, 분석 및 AI/ML 애플리케이션을 직접적으로 지원하거나 사용자 지향 앱은 물론이고 단계적 데이터를 처리할 때 간접적으로 지원합니다. 동일한 체크리스트를 권장하지만 엔지니어링 목표를 향해 노력하면서 이해관계자의 요구 사항을 충족하는지 확인하기 위해 이해관계자에 맞게 약간 구성하는 것이 좋습니다.

우리는 이것이 항상 가능한 것은 아니라는 점을 충분히 인식하고 있습니다. 엔지니어로서 귀하의 역할은 가장 적합한 솔루션을 만드는 것입니다. 비록 그것이 종간 지점에 안착하거나 명확한 답이 없다는 것을 인정하는 것을 의미하더라도 말입니다. 기술의 놀라운 발전에도 불구하고 특정 논리적 딜레마에는 직접적인 해결책이 없습니다.

수집된 데이터 준비 우

리는 데이터 수집에 대한 데이터 레이크 접근 방식을 옹호합니다. 이를 위해서는 대부분의 데이터를 S3, Google Cloud Platform 또는 Azure와 같은 클라우드 스토리지 시스템에 수집한 후 분석을 위해 데이터 웨어하우스에 로드해야 합니다.

한 단계 더 나아가 메타데이터를 사용하여 성능, 안정성 및 확장된 기능을 추가하는 Delta Lake와 같은 데이터 스토리지 프로토콜을 활용하는 레이크하우스입니다. 레이크하우스는 일부 창고 기능을 복제할 수도 있습니다. 이는 별도의 창고 시스템에 데이터를 로드할 필요가 없다는 것을 의미합니다. Databricks의 Unity Catalog과 같은 데이터 거버넌스 계층을 추가하면 더 나은 검색 가능성을 제공할 수 있습니다.

조직 전체의 모든 데이터 자산에 대한 액세스 관리 및 협업이 가능합니다.

데이터 준비를 위한 일반적이고 효과적인 방법은 Delta Lake, Apache Iceberg 또는 Apache Hudi를 포함한 메타데이터 중심의 Parquet 기반 파일 형식을 활용하는 것입니다. 대규모 데이터세트용으로 설계된 압축 열 형식인 Parquet를 기반으로 하는 이러한 형식은 메타데이터 레이어를 통합하여 시간 이동, ACID(원자성, 일관성, 격리 및 내구성) 규정 준수 등과 같은 기능을 제공합니다.

세 가지 서로 다른 품질 계층에서 단계적 데이터를 처리하는 메달리온 아키텍처와 이러한 형식을 통합하면 전체 데이터 기록의 보존이 보장됩니다. 이를 통해 새 열 추가, 손실된 데이터 검색 및 기록 데이터 백필이 쉬워집니다.

메달리온 아키텍처의 미묘한 차이는 데이터 변환에 관한 장 ([2장](#))에서 자세히 설명합니다. 현재 논의에서는 선택한 클라우드 스토리지 제공업체 내의 "스테이징 계층"으로 모든 데이터를 전달할 수 있는 가능성을 고려하는 것이 적절합니다.

OLAP 대 OLTP 데이터베이스

데이터 웨어하우징에서 가장 큰 선택은 클라우드 네이티브 데이터베이스를 사용할지 아니면 클라우드에 호스팅되는 기존 데이터베이스를 사용할지 여부입니다. 주요 차이점은 클라우드 네이티브 솔루션의

분산 특성과 열 중심 아키텍처입니다. 이는 일반적으로 OLAP(온라인 분석 처리) 및 OLTP(온라인 트랜잭션 처리)라고 합니다.

- OLAP 시스템은 대량의 데이터를 신속하게 처리하도록 설계되었습니다. 이는 일반적으로 분산 처리 및 열 기반 아키텍처를 통해 수행됩니다. 최신 클라우드 네이티브 데이터베이스는 OLAP 시스템입니다. 세 가지 OLAP 솔루션은 Amazon Redshift, Google BigQuery 및 Snowflake입니다.
- OLTP 시스템은 여러 사용자로부터 발생하는 대량의 트랜잭션 데이터를 처리하도록 설계되었습니다. 이는 일반적으로 행 기반 데이터베이스의 형태를 취합니다. 많은 기존 데이터베이스 시스템은 Postgres, MySQL 등 OLTP입니다.

OLAP 시스템은 속도, 안정성 및 낮은 유지 관리 비용으로 인해 분석 및 데이터 과학 팀에서 가장 일반적으로 사용됩니다.

데이터 웨어하우스 선택 시 고려해야 할 사항은 다음과 같습니다.

선택한 컴퓨팅 플랫폼 이러한 기술의 통합은 나

마지 기술 스택에 따라 크게 달라집니다. 예를 들어 조직의 모든 도구가 Amazon 생태계에 있는 경우 Redshift는 Google BigQuery보다 비용 효율적이고 구현이 간단할 수 있습니다. 마찬가지로 Databricks 에코시스템에는 스토리지(Lakehouse, Databricks SQL)부터 가버넌스(Unity Catalog) 및 컴퓨팅(Apache Spark)에 이르기까지 광범위한 기능이 포함되어 있습니다.

기능 BigQuery는

반구조화된 데이터 및 ML 작업을 훌륭하게 지원합니다. Redshift Spectrum은 S3의 데이터에서 외부 테이블을 생성하는 데 유용한 도구입니다. Databricks는 Spark, Databricks SQL 및 Delta Lake를 사용하여 스토리지와 컴퓨팅을 분리합니다. 모든 참고에는 강점과 약점이 있습니다. 이는 팀의 사용 사례에 따라 평가되어야 합니다.

비용 가

격 구조는 플랫폼마다 크게 다릅니다. 불행하게도 가격은 볼투명할 수 있습니다. 대부분의 경우 플랫폼을 사용하기 전까지는 데이터베이스가 어떻게 사용될지 실제로 알 수 없습니다.

가격 책정의 목표는 데이터베이스를 사용하여 비용을 최소화하는 방법과 해당 경로를 사용할 경우 발생할 수 있는 비용을 이해하는 것입니다. 예를 들어, 비용이 스캔된 데이터 양(BigQuery)과 직접적으로 연관되어 있는 경우 지능형 분할 및 쿼리 필터링이 큰 도움이 될 수 있습니다. 레이크하우스를 사용하면 레이크와 데이터 웨어하우스의 중복성과 복제된 데이터가 절약됩니다. 참고는 보관 비용과 제한 사항이 더 높은 경향이 있기 때문에 이는 특히 중요합니다.

비용에 대한 직접적인 대답은 없지만 참고는 특히 규모에 따라 비용이 많이 들 수 있으므로 조사해 볼 가치가 있습니다.

변경 데이터 캡처 변

경 데이터 캡처(CDC)는 소스 데이터베이스의 변경 사항을 캡처하고 추적하여 다운스트림 시스템을 업데이트하는 데이터 엔지니어링 설계 패턴입니다. CDC는 전체 데이터베이스를 일괄 로딩하는 대신 변경된 데이터만 전송하여 속도와 성능을 모두 최적화합니다.

자원 사용.

이 기술은 대상 시스템의 데이터가 최신 상태이고 소스와 동기화되도록 보장하므로 실시간 분석 및 데이터 웨어하우징에 매우 중요합니다. 충분 업데이트를 활성화함으로써 CDC는 데이터 파이프라인 전체에서 데이터 가용성과 일관성을 향상합니다. Joe Reis와 Matt Housley가 *Fundamentals of Data Engineering*(O'Reilly, 2022)에서 간단히 표현한 대로 "CDC는…소스 데이터베이스 시스템에서 변경 사항을 수집하는 프로세스입니다."

CDC는 불필요하게 복잡하고 시간이 많이 소요될 수 있는 프로세스인 SCD(Slowly Changing Dimension) 유형 1 및 2 테이블 생성과 같은 분석 및 엔지니어링 패턴에서 중요해졌습니다.

CDC를 기본적으로 지원하는 플랫폼이나 솔루션을 선택하면 일반적인 작업을 신속하게 수행할 수 있어 가장 중요한 일에 집중할 수 있습니다.

한 가지 예는 **SCD 유형 1 및 2에 대한 기본 지원**을 제공하는 Databricks의 DLT(Delta Live Tables)입니다. 일괄 및 스트리밍 파이프라인 모두에서.

목적지 체크리스트

다음은 목적지를 선택할 때 고려해야 할 질문에 대한 샘플 체크리스트입니다.

질문	예
우리는 누구와 협력할 것인가? 인적 자원	
데이터는 어떻게 사용되나요?	재직 기간/계약 기간이 수익에 미치는 영향을 이해하세요.
	결과적으로 다국적 조직이 됩니다.
목적지가 여러 개인가요? 데이터는 Delta Lake에 보관됩니다. 최종 테이블은 Databricks에 구축되었습니다.	
형식은 무엇입니까?	SQL.
빈도는 얼마나 됩니까?	델타 호수의 쪽모이 세공 마루. Databricks SQL의 구조화/반구조화.
볼륨은 얼마나 됩니까?	DLT를 사용하여 ~100K 다운스트림 처리의 기존 풀을 포함하여 하
어떤 처리가 필요합니까?	루에 약 1,000개의 새로운 행이 있습니다. Spark의 ML 모델 및 생성 AI 애플리케이션.
데이터는 어떻게 저장되나요?	Delta Lake의 Databricks SQL과 외부 테이블이 혼합되어 있습니다.

섬추 고려사항

이 섹션에서는 중요한 데이터 특성에 대해 간략히 설명합니다. 이 목록은 전체 목록이 아니며 특정 상황의 영향을 받지만 빈도, 양, 형식 및 처리와 같은 측면이 기본으로 나타납니다.

우려.

빈도 우리는 빈도

의 첫 번째 고려 사항이 경계, 즉 데이터 세트가 경계가 있는지 또는 경계가 없는지 이미 언급했습니다. 경계와 비즈니스 요구 사항에 따라 일괄 또는 스트리밍 형식으로 데이터를 수집하는 빈도가 결정됩니다. 그림 1-3은 배치 프로세스와 스트리밍 프로세스의 차이점을 깔끔하게 보여줍니다. 스트리밍은 이벤트가 발생할 때마다 캡처하고 이름에서 알 수 있듯이 일괄 그룹화합니다.

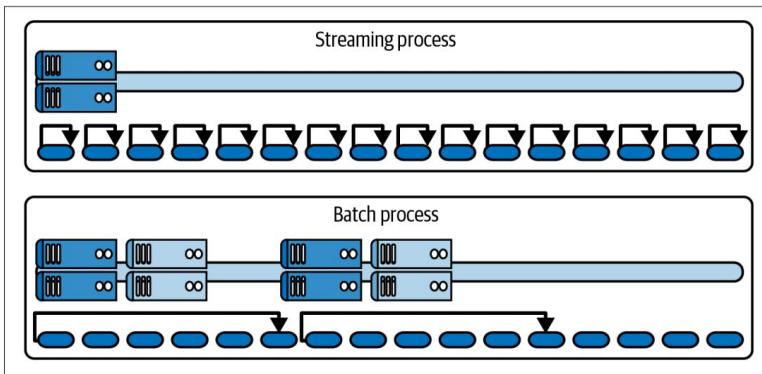


그림 1-3. 자연 시간은 "배치" 또는 "스트리밍" 프로세스를 정의하는 속성입니다. 임의의 자연 시간 임계값을 초과하면 데이터가 "스트리밍"된 것으로 간주합니다(Denny Lee 제공).

가장 적절한 빈도와 호환되는 수집 솔루션을 선택하는 데 도움이 되도록 배치, 마이크로 배치, 스트리밍을 우리의 생각과 함께 제시하겠습니다.

일괄

일괄 처리는 데이터를 한 번에 모두 처리하는 것이 아니라 일괄적으로 처리하는 행위입니다. 소스를 반복하는 for 루프와 마찬가지로 배치에는 단순히 제한된 데이터세트를 청크하고 각 구성 요소를 처리하거나 데이터가 도착할 때 제한되지 않은 데이터세트를 처리하는 작업이 포함됩니다.

マイ크로배치

マイ크로 배치는 단순히 배치 처리에서 "다이얼을 낮추는 것"입니다. 일반적인 배치 수집이 매일 작동하는 경우 마이크로 배치는 시간 단위 또는 분 단위로 작동할 수 있습니다. 물론, “어느 시점에서 이것이 의미론에 불과합니까?”라고 말할 수도 있습니다. 자연 시간이 100ms인 마이크로 배치 파이프라인은 제게는 스트리밍과 매우 비슷해 보입니다.”

우리는 동의한다!

이 장(및 책)의 나머지 부분에서는 대기 시간이 짧은 마이크로 배치 솔루션을 스트리밍 솔루션으로 치환합니다. 가장 확실한 것은 Spark Structured Streaming입니다. "기술적으로" 마이크로 배치인 반면 수백 밀리초의 대기 시간은 Spark 구조적 스트리밍을 효과적으로 실시간 솔루션으로 만듭니다.

스트리밍 스

트리밍은 데이터 세트가 생성될 때 제한이 있거나 제한되지 않은 데이터 세트를 지속적으로 읽는 것을 의미합니다. 스트리밍에 대해 자세히 논의하는 않지만 추가 연구가 필요합니다. 스트리밍 데이터 소스를 포함적으로 이해하려면 [Streaming 101](#), [스트리밍 데이터베이스](#), 물론 [데이터 엔지니어링의 기초도 다룹니다](#).

행동 양식. 제한되지 않은 데이터를 스트리밍하는 일반적인 방법은 다음과 같습니다.

윈도우잉 시간

적 경계를 기준으로 데이터 소스를 유한한 덩어리로 분할하는 것입니다.

고정창

데이터는 기본적으로 "마이크로 배치"되어 작은 고정 창에서 대상으로 읽혀집니다.

슬라이딩 창 고정 창

과 비슷하지만 경계가 겹치는 점이 있습니다.

세션 이벤

트 시퀀스가 비활성 간격으로 분리되는 동적 창입니다. 세션에서 "창"은 데이터 자체에 의해 정의됩니다.

시간에 구애받지

않음 시간이 중요하지 않고 종종 배치 워크로드를 활용하는 데이터에 적합합니다.

[그림 1-4](#)는 고정 창, 슬라이딩 창 및 세션 간의 차이점을 보여줍니다. 불일치가 발생할 수 있으므로 실제 이벤트 시간과 처리 시간을 구별하는 것이 중요합니다.

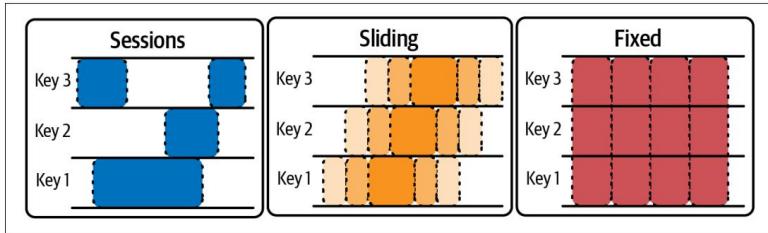


그림 1-4. 고정 창, 슬라이딩 창 및 세션은 스트리밍 데이터 도착을 매우 다르게 처리합니다.

메시지 서비스. "메시지 서비스"라고 하면 스트리밍 데이터를 전달하고 전송하는 "전송 계층" 또는 시스템을 의미합니다. 한 가지 중요한 점은 이것이 직접적인 비교가 아니라는 것입니다. 이러한 서비스에는 중복이 있지만 많은 서비스는 근본적으로 다른 아키텍처에서 작동하므로 "Kafka 대 Pub/Sub" 또는 "Kinesis 대 Redpanda" 논의는 거의 관련이 없습니다.

Apache Kafka

2011년 LinkedIn에서 시작된 Apache Kafka는 메시지 대기열 시스템으로 시작했지만 빠르게 분산 스트리밍 플랫폼으로 발전했습니다. Kafka의 디자인은 높은 처리량과 확장성을 허용하지만 본질적인 복잡성은 많은 사람들에게 여전히 장애물로 남아 있습니다.

Redpanda

Kafka의 대안으로 개발된 Redpanda는 단순화된 구성 및 설정으로 비슷한 성능을 자랑합니다. Redpanda는 Java가 아닌 C++를 기반으로 하며 Kafka API와 호환됩니다.

Pub/Sub

Pub/Sub는 내구성이 뛰어난 동적 메시징 대기열을 위한 Google Cloud 제품입니다. Kafka와 달리 Google Pub/Sub는 가변적인 작업 부하를 처리하기 위해 동적으로 확장됩니다. Pub/Sub는 '스트림'과 '샤드'를 처리하는 대신 '주제'와 '구독'을 선택합니다. Pub/Sub의 가장 큰 장점은 대부분의 '유지관리' 작업이 없어 거의 완벽하게 관리된다는 점입니다.

카네시스

Kinesis는 또 다른 강력하고 완전관리형 서비스입니다. Amazon 서비스인 Kinesis는 자동 확장성과 실시간 데이터 처리 기능을 제공하는 동시에 다른 Amazon Web Services(AWS) 제품과 쉽게 통합할 수 있다는 점을 제공합니다. Pub/Sub와 마찬가지로 Kinesis는 관리형 서비스 특성이 뛰어나 Apache Kafka 보다 운영 부담이 적습니다.

스트림 처리 엔진. 스트림 처리는 실시간 데이터(스트림)를 분석하고 이에 대한 조치를 취하는 것입니다. Kafka의 수명을 고려할 때 가장 인기 있고 잘 알려진 세 가지 스트림 처리 도구는 다음과 같습니다.

Apache Flink 가

동 중지 시간을 최소화하면서 제한되지 않은 데이터 세트와 제한된 데이터 세트를 모두 지속적으로 처리하는 오픈 소스 엔진입니다.

Apache Flink는 인메모리 계산을 통해 짧은 대기 시간을 보장하고 단일 실패 지점을 제거하여 고가용성을 제공하며 수평적으로 확장됩니다.

Apache Spark 구조적 스트리밍 실시간 데

이터 처리를 처리하도록 설계된 Apache Spark 에코시스템의 일부입니다. 이는 스트리밍 데이터에 Spark의 DataFrame 및 Dataset API의 친숙함과 강력한 기능을 제공합니다. 구조적 스트리밍은 데이터 처리 분야에서 Apache Spark의 인기와 Databricks와 같은 도구의 엔진 편재성을 고려할 때 매력적인 옵션이 될 수 있습니다.

Apache Kafka Streams

Kafka를 기반으로 구축된 라이브러리로 상태 저장 처리 기능을 제공하지만 Java 와의 연결이 제한될 수 있습니다.

스트림 처리 단순화. 비교적 새로운 여러 솔루션은 성능과 단순한 개발 주기에 초점을 맞춘 간단한 클라이언트를 제공하여 스트림 처리를 단순화합니다.

관리형 플랫폼

Databricks를 예로 들면 DLT(Delta Live Tables)와 같은 도구를 활용하거나 단순히 Databricks 런타임에서 Spark Streaming 작업을 실행하는 것은 복잡성을 강력하게 추상화하고 스트리밍 시스템 구축 프로세스를 대폭 단순화할 수 있습니다.

Confluent Kafka

Apache Kafka 기능을 Python에 도입하려는 시도이지만 Java 기능에 비해 아직 초보적입니다. Confluent Kafka는 psycopg2가 Postgres 클라이언트 라이브러리인 것과 같은 방식으로 단순한 클라이언트 라이브러리입니다.

Bytewax

스트림 처리를 처리하는 보다 직관적이고 Python적인 방법을 제공하여 더 많은 개발자가 더 쉽게 액세스할 수 있도록 함으로써 격차를 해소하는 것을 목표로 하는 라이브러리입니다. Rust를 기반으로 구축된 Bytewax는 성능이 뛰어나고 Flink보다 단순하며 피드백 루프가 짧고 배포/확장성이 쉽습니다.

Apache Beam 또는 Estuary Flow와 같은 스트림 처리를 통합하거나 스트림 처리를 데이터베이스(스트리밍 데이터베이스)와 직접 결합하려는 새로운 도구의 인기가 높아지고 있습니다.

스트리밍 시스템을 권장합니다 및 스트리밍 데이터베이스 자세히 살펴보려면.

스트리밍 환경은 복잡하기는 하지만 특히 Databricks와 같은 관리형 플랫폼과 Spark Structured Streaming과 같은 대기 시간이 짧은 마이크로 배치 솔루션을 고려할 때 단순화와 사용자 친화성 측면에서 큰 진전을 보였습니다.

많은 사람들이 실시간 데이터를 궁극적인 목표로 생각하지만 우리는 "직시" 접근 방식을 강조합니다. 모든 솔루션과 마찬가지로 대기 시간은 무한히 최적화될 수 있지만 솔루션 비용(및 복잡성)은 그에 비례하여 증가합니다. 대부분의 사람들은 일별 또는 반일별 데이터에서 시간별/시간별 데이터로 전환하는 것이 완벽하게 수용 가능한 솔루션임을 알게 될 것입니다.

페이지드

"페이지드"라는 용어는 데이터 라우팅, 처리 또는 형식 지정에 사용되는 메타데이터 또는 헤더와 함께 전송되는 실제 메시지를 나타냅니다. 데이터 페이지드는 상상할 수 있는 거의 모든 형태를 취할 수 있으므로 본질적으로 광범위하게 정의됩니다. 이 섹션에서는 일반적인 페이지드 특성에 대해 설명합니다.

용량

볼륨은 데이터 수집 결정에서 중추적인 요소로, 처리 및 스토리지 솔루션의 확장성에 영향을 미칩니다. 데이터 볼륨을 평가할 때는 다음과 같은 요소를 고려해야 합니다.

비용 볼

률이 높을수록 스토리지 및 컴퓨팅 리소스 측면에서 비용이 증가하는 경우가 많습니다. 솔루션에 따라
창고, 호수 또는 호수집과 관련된 보관/준비 비용을 포함하여 비용 요소를 예산 및 프로젝트 요구 사항
에 맞게 조정하십시오.

지연 시간 실

시간, 거의 실시간 또는 일괄 데이터 수집이 필요한지 여부에 따라 지연 시간이 중요한 요소가 될 수
있습니다. 실시간 처리에는 더 많은 리소스가 필요할 뿐만 아니라 볼륨이 급증할 때 더 큰 효율성도
필요합니다. 데이터 볼륨에 관계없이 시스템이 대기 시간 요구 사항을 처리할 수 있는지 확인하십시오.

처리량/확장성 엄청난 양의 수신

데이터를 처리하는 수집 도구의 능력을 아는 것이 중요합니다. 데이터 원본이 대량의 데이터를 생성하
는 경우 수집 도구는 병목 현상을 일으키지 않고 해당 데이터를 수집할 수 있어야 합니다.

보유

볼륨이 높을수록 데이터 보존 정책이 더욱 중요해집니다. 오래된 데이터를 오래 사용하거나 더 저렴
한 장기 스토리지 솔루션으로 옮기는 전략이 필요합니다. 오래된 데이터를 저장하는 것 외에도 손실
된 데이터에 대한 보안 및 백업(복원)도 고려해야 합니다.

대규모 데이터 세트를 처리하려면 Apache Avro 또는 Parquet와 같은 입출력 형식을 사용하는 것이 중요
합니다. 각각은 특히 스키마 진화와 관련하여 뚜렷한 장점과 제약을 제공합니다. 이러한 각 고려 사항에 대
해 반드시 미래를 고려하십시오. 유지 가능한 솔루션은 데이터 볼륨이 엄청나게 증가하면 빠르게 붕괴될 수
있습니다.

구조 및 형태 데이터는 깔끔

한 관계형 "구조화된" 데이터부터 보다 자유로운 형식의 "비구조화된" 데이터에 이르기까지 형식과 구조가
매우 다양합니다.

중요한 것은 구조가 품질과 동일하지 않다는 것입니다. 이는 단지 스키마의
존재를 의미할 뿐입니다.

오늘날의 AI 중심 환경에서는 대규모 언어 및 기계 학습 모델의 발전으로 이러한 데이터에서 풍부한 통찰력
을 얻을 수 있으므로 구조화되지 않은 데이터의 가치가 치솟고 있습니다. 그럼에도 불구하고 인간은

심층 분석에 있어 구조화된 데이터를 선호하는 경향이 있는데, 이는 거의 반세기 동안 데이터 분석의 주요 요소인 SQL(구조적 쿼리 언어)의 지속적인 인기로 입증되었습니다.

구조화되지 않았습니다. 앞서 언급했듯이 구조화되지 않은 데이터는 사전 정의된 스키마나 구조가 없는 데이터입니다. 대부분 텍스트로 표현되지만 다른 형태의 미디어도 구조화되지 않은 데이터를 표현합니다.

비디오, 오디오, 이미지는 모두 수치적으로 분석할 수 있는 요소를 가지고 있습니다. 구조화되지 않은 데이터는 Pierce Brown 소설의 텍스트일 수 있습니다.

“사람은 자신이 날 수 있다고 생각하지만 점프하는 것을 두려워합니다. 불쌍한 친구가 뒤에서 그를 밀어낸다.” 그는 나를 올려다본다. “좋은 친구가 함께 뛰어요.”

이러한 데이터는 종종 기계 학습이나 AI 애플리케이션에 제공되어 이해관계자의 요구 사항을 광범위하게 이해해야 할 필요성을 강조합니다. 기계 학습의 복잡성을 고려할 때 이 비정형 데이터를 수집하기 전에 어떻게 활용될지 파악하는 것이 중요합니다.

텍스트 길이나 압축되지 않은 크기와 같은 측정항목이 모양을 측정하는 역할을 할 수 있습니다.

반 구조화. 반구조화된 데이터는 구조화된 데이터와 구조화되지 않은 데이터 사이 어딘가에 있습니다. XML과 JSON은 널리 사용되는 두 가지 형식입니다. 반구조화된 데이터는 JSON 페이로드 형식을 취할 수 있습니다.

```
'["친구": ["steph", "julie", "thomas", "tommy", "michelle", "tori",  
"larry"]]'
```

데이터 플랫폼이 계속해서 성숙해짐에 따라 반구조화된 데이터를 직접 처리하고 분석하는 능력도 발전할 것입니다. 다음 스니펫은 Google BigQuery에서 반구조화된 JSON을 구문 분석하여 목록을 데이터 행으로 피벗하는 방법을 보여줍니다.

```
j_data AS 사용 (  
선택하다 (  
  
JSON '{"친구": ["스테프", "줄리", "토마스",  
"토미", "마셀", "토리", "래리"]}'  
) AS my_friends_json  
, l_data AS ( SELECT  
  
JSON_EXTRACT_ARRAY(JSON_QUERY(j_data.my_friends_json, "$.friends"),  
'$'
```

```

        ) my_friends_list 로
    j_data 에서
)
선택하다
my_friends
FROM l_data, UNNEST(l_data.my_friends_list) as my_friends ORDER BY
RAND()

```

어떤 상황에서는 데이터 처리 다운스트림을 분석 계층으로 이동하는 것이 가치가 있습니다. 이를 통해 분석가와 분석 엔지니어는 데이터를 쿼리하고 저장하는 방법에 있어 더 큰 유연성을 얻을 수 있습니다. 웨어하우스의 반구조화된 데이터(또는 외부 테이블을 통해 액세스)는 스키마를 변경하거나 데이터가 누락되는 경우에도 유연성을 제공하는 동시에 테이블 형식 데이터 조작 및 SQL의 모든 이점을 제공합니다.

하지만 누락된 데이터로 인해 오류가 발생하지 않도록 하려면 이 데이터의 유효성을 올바르게 검사해야 합니다. 많은 잘못된 쿼리는 NULL 데이터를 부적절하게 고려하여 발생합니다.

JSON의 형태를 설명하는 데에는 키, 값, 중첩 요소 수에 대한 논의가 포함되는 경우가 많습니다. [JSONLint](#) 와 같은 도구를 적극 권장합니다. 및 [JSON 크랙](#) 이 목적을 위해 JSON/XML 데이터의 유효성을 검사하고 형식을 지정하는 VS Code 확장도 있습니다.

구조화되어 있습니다. 황금색의 "이상적인" 데이터인 구조화된 소스는 고정된 스키마와 변하지 않는 키로 깔끔하게 구성됩니다. 50년 넘게 SQL은 구조화된 데이터를 쿼리하는 데 선택되는 언어였습니다. 구조화된 데이터를 저장할 때 열 수와 테이블 길이(행)에 관심을 두는 경우가 많습니다.

이러한 특성은 구체화, 증분 빌드 및 전체적으로 OLAP 대 OLTP 데이터베이스(열/행 중심)의 사용을 알려줍니다.

오늘날 많은 데이터에는 구조가 부족하지만 여전히 SQL이 분석을 위한 주요 도구입니다. 이것이 바뀔까요? 그럴 수도 있지만 앞서 보여드린 것처럼 SQL은 단순히 반구조화된 형식을 수용하도록 적응할 가능성이 더 높습니다. AI의 발전과 함께 언어 기반 쿼리 도구가 등장하기 시작했지만 SQL은 종종 종개자 역할을 합니다. SQL이 데이터 분석에서 사라지더라도 API로 계속 존재할 가능성은 높습니다.

체재

소스 데이터에 가장 적합한 형식은 무엇입니까? JSON과 CSV(쉼표로 구분된 값)가 일반적인 선택이지만 형식 고려 사항은 무한할 수 있습니다. 예를 들어 일부 오래된 SFTP/FTP 전송은 압축되어 도착할 수 있으므로 추가 추출 단계가 필요합니다.

데이터 형식에 따라 처리 요구 사항과 사용 가능한 솔루션이 결정되는 경우가 많습니다. Airbyte와 같은 도구는 CSV 소스와 원활하게 통합될 수 있지만 사용자 지정 압축 방법이나 기발한 Windows 인코딩으로 인해 문제가 발생할 수 있습니다(믿어주세요. 실제로 발생 합니다).

가능하다면 친숙하고 널리 사용되는 데이터 형식을 선택하는 것이 좋습니다. 차량 수리와 마찬가지로 형식이 인기가 많을수록 리소스, 라이브러리 및 지침을 더 쉽게 찾을 수 있습니다. 하지만 우리의 경험에 따르면 복잡한 형식과 씨름하는 것은 일종의 통과의례이지만 그것이 우리 일을 재미있게 만드는 이유 중 하나입니다!

다양성

여러 소스를 다루게 되어 다양한 페이로드를 처리하게 될 가능성이 높습니다. 데이터 다양성은 수집 솔루션을 선택하는 데 큰 역할을 합니다. 이는 서로 다른 데이터 유형을 처리할 수 있을 뿐만 아니라 스키마 변경 및 다양한 형식에 적응할 수 있을 만큼 유연해야 합니다. 다양성은 거버넌스와 관찰 가능성 특히 어렵게 만듭니다. 이에 대해서는 5 장에서 논의하겠습니다.

데이터 다양성을 고려하지 못하면 병목 현상, 대기 시간 증가, 무질서한 파이프라인이 발생하여 수집된 데이터의 무결성과 유용성이 손상될 수 있습니다.

솔루션 선택

팀을 위한 가장 좋은 도구는 소스와 타겟을 지원하는 도구입니다. 각 조직의 고유한 데이터 요구 사항을 고려하여 상황에 따라 선택을 하게 됩니다. 그럼에도 불구하고 멘토, 동료, 업계 전문가의 지식을 활용하는 것의 중요성은 아무리 강조해도 지나치지 않습니다. 컨퍼런스는 비용이 많이 들 수 있지만 지식의 양은 매우 귀중할 수 있습니다. 예산이 부족한 사람들에게는 데이터 커뮤니티가 매우 중요할 수 있습니다. Slack, LinkedIn과 같은 플랫폼과 업계 뉴스레터를 통해 찾아보세요.

수집 솔루션을 고려할 때 우리는 일반적인 고려 사항과 솔루션별 고려 사항의 관점에서 생각합니다. 전자는 우리가 고려할 모든 도구에 적용되고 후자는 도구 클래스에만 적용됩니다.

일반적인 고려 사항에는 확장성, 구축 비용, 유지 관리 비용, 전환 비용 및 기타 시스템 수준이 포함됩니다.

우려.

솔루션별 고려 사항은 일반적으로 두 가지 형식 중 하나를 취하는 도구 클래스에 따라 다릅니다.

- 선언적 솔루션이 결과를 결정합니다. 예를 들어, "AWS Glue UI를 사용하여 데이터를 체계적으로 처리하는 크롤러를 구축합니다." 또는 "UI를 통해 새로운 Airbyte 연결을 생성합니다."
- 필수적인 해결책은 행동을 지시합니다. 예를 들어, "Stripe API를 호출하고, 데이터를 인코딩/디코딩하고, 이를 Snowflake에 점진적으로 로드하는 람다를 구축합니다."

이러한 각 솔루션에는 장단점이 있습니다. 각각에 대해 간략하게 논의하고 데이터 통합에 접근하기 위한 권장 방법을 제시하겠습니다.

선언적 솔루션

우리는 코드형 인프라, DRY(반복하지 마세요) 및 기타 소프트웨어 엔지니어링 모범 사례와 같은 최신 데이터 스택(MDS) 원칙을 준수하는지에 따라 선언적 솔루션을 레거시, 최신 또는 기본으로 분류합니다. 기본 플랫폼은 처음 두 플랫폼과 다릅니다. 즉, 클라우드 제공업체에 직접 통합됩니다.

레거시

Think Talend, WhereScape 및 Pentaho. 이러한 도구에는 강력한 커넥터가 있으며 풍부한 커뮤니티와 광범위한 지원의 이점을 누릴 수 있습니다. 그러나 데이터 환경이 발전함에 따라 이러한 도구 중 상당수는 뒤쳐져 MDS의 요구 사항에 부합하지 않습니다. 특별한 이유가 없는 한 레거시 엔터프라이즈 도구 이외의 다른 도구를 살펴보는 것이 좋습니다.

현대의

Fivetran, Stitch 및 Airbyte가 작동하는 곳입니다.
"커넥터"를 중심으로 설계된 이 도구는 최첨단 기술을 기반으로 하고 최고의 MDS를 활용하여 다양한 소스와 대상을 원활하게 연결할 수 있습니다.

토종의

처음 두 가지 솔루션에서는 데이터가 한 소스에서 다른 소스로 이동되어야 한다는 가정하에 작업하고 있습니다.

즉시 수집을 지원하는 관리형 플랫폼이 있다면? 예를 들어 Databricks는 메시지 버스와 클라우드 스토리지에서 기본적으로 수집할 수 있습니다.

```
스트리밍 테이블 생성 raw_data
AS 선택 *
FROM cloud_files("/raw_data", "json");

SUM(이익)을 선택하여 스트리밍 테이블 clean_data
생성 ...
원시 데이터 에서 ;
```

"올바른" 유형의 선언적 솔루션은 없지만 많은 사람들이 이러한 솔루션, 특히 기존 클라우드 공급자/플랫폼에 기본으로 제공되는 솔루션을 구축하고 유지 관리하는 데 드는 비용 절감의 이점을 누릴 것입니다.

구축/유지 비용

선언적 솔루션은 자체로 무간섭입니다. 여기에서 비용을 절감할 수 있습니다! 전담 엔지니어가 커넥터 개발 및 유지 관리를 담당합니다. 이는 무거운 작업을 전문가에게 위임한다는 의미입니다. 대부분의 유료 솔루션에는 지원 팀 또는 전담 고객 관리자가 함께 제공되어 구하의 요구에 맞는 통찰력과 지침을 제공합니다. 이러한 전문가는 데이터 환경에 대한 조감도를 갖고 있으며 특정 데이터 문제에 대한 모호성을 탐색하거나 다른 실무자와 연결하는 데 도움을 줄 수 있습니다.

확장성 확장성

은 기존 솔루션을 기반으로 구축하는 것이 얼마나 쉬운가에 달려 있습니다. 새로운 커넥터가 Airbyte 또는 Fivetran에 추가될 가능성은 얼마나 됩니까? 동일한 프레임워크를 기반으로 직접 구축할 수 있나요? 아니면 제품 팀을 위해 몇 주/개월/년을 기다려야 합니까? Stitch를 사용하면 충분 합니까, 아니면 보완적인 솔루션이 필요합니까? 여러 솔루션을 저글링하면 비용이 부풀려지고 워크 플로가 복잡해질 수 있다는 점을 기억하세요. 선언적 솔루션에서 확장성은

거대한 솔루션이 자신의 요구 사항 중 15%만 해결한다는 사실을 깨닫고 솔루션을 채택하고 싶은 사람은 아무도 없습니다.

전환 비용

전환 비용은 선언적 도구의 한계가 드러나는 부분입니다. 독점 프레임워크와 특정 CDC 방법으로 인해 다른 도구로 마이그레이션하는 데 비용이 많이 듭니다. 공급업체를 고수하는 것은 필요한 절충안일 수 있지만 잠재적인 솔루션을 평가할 때 이를 고려하는 것이 중요합니다.

필수 솔루션 필수 데이터 수집 접

근 방식은 사내 Singer 텝, 램다 기능, Apache Beam 템플릿 또는 Apache Airflow와 같은 시스템을 통해 조정된 작업일 수 있습니다.

일반적으로 상당한 리소스를 보유한 대규모 조직은 필수 방법론을 채택하는 데 가장 큰 가치를 찾습니다. 맞춤형 사내 수집 프레임워크를 유지 관리하고 확장하려면 일반적으로 여러 데이터 엔지니어 또는 전담 인력의 전문 지식이 필요합니다.

팀.

명령형 솔루션의 가장 큰 이점은 확장성입니다.

확장성 본질적

으로 필수적인 것은 맞춤형입니다. 즉, 각 텝과 대상이 비즈니스 요구 사항에 맞게 조정된다 는 의미입니다. 데이터 통합 옵션을 탐색할 때 모든 기준을 충족하는 단일 도구는 없다는 사실이 금새 명백해집니다. 표준 솔루션에는 필연적으로 타협이 수반됩니다.

그러나 명령형 접근 방식을 사용하면 원하는 사양에 따라 정확하게 설계할 수 있는 자유가 있습니다. 불행하게도 대규모의 전담 데이터 엔지니어링 조직이 없으면 이 확장성을 구축하고 유지하는 데 엄청난 비용이 듭니다.

구축/유지 비용

명령형 솔루션은 복잡하고 어려운 수집 문제를 해결할 수 있지만 상당한 엔지니어링이 필요합니다. [Stripe 엔터티 관계 디아이어그램](#) 살펴보기 이것은 엄청나게 시간이 많이 걸릴 수 있다 는 것을 당신에게 확신시키기에 충분할 것입니다. 또한 스키마 변경이나 API 지원 중단과 같 은 데이터의 진화하는 특성으로 인해 복잡성이 증폭될 수 있습니다. 단일 데이터 원본을 관리 하는 것도 중요하지만 여러 원본으로 확장하는 경우에는 어떻게 되나요?

진정으로 탄력적이고 명령적인 시스템은 모듈성, 테스트 가능성 및 명확성을 강조하면서 소프트웨어 설계의 모범 사례를 통합해야 합니다. 이러한 원칙을 무시하면 시스템 복구 시간이 단축되고 확장성이 저하되어 궁극적으로 비즈니스 운영에 영향을 미칠 수 있습니다. 따라서 우리는 강력한 데이터 엔지니어링 인프라를 갖춘 기업만이 완전히 필수적인 방식을 고려할 것을 제안합니다.

전환 비용

서로 다른 공급자의 형식 간에 잠재적인 비호환성을 고려할 때 하나의 필수 솔루션에서 다른 솔루션으로 전환하는 것이 항상 간단하지는 않을 수 있습니다. 그러나 긍정적으로 보면 Singer와 같은 공통 프레임워크 기반 플랫폼은 더 높은 호환성을 보여 잠재적으로 Fivetran 또는 Airbyte와 같은 순수 선언적 도구에 비해 더 부드러운 전환을 제공할 수 있습니다.

하이브리드 솔루션 통합

에서 올바른 균형을 맞추는 것은 종종 하이브리드 접근 방식을 선택하는 것을 의미합니다. 여기에는 대부분의 통합 작업에 Fivetran과 같은 도구를 활용하는 동시에 고유한 소스에 대한 자체 솔루션을 제작하거나 Airbyte/Meltano와 같은 플랫폼을 선택하고 지원되지 않는 데이터 소스에 대한 사용자 정의 구성 요소를 만드는 것이 포함될 수 있습니다.

오픈 소스에 기여하는 것은 하이브리드 환경에서도 보람을 느낄 수 있습니다. 결함이 없는 것은 아니지만 Airbyte 또는 Singer Tap과 같은 하이브리드 커넥터는 광범위한 커뮤니티 지원의 혜택을 받습니다. 특히 해당 부문에 대한 Airbyte의 기여는 시장 역학에 긍정적인 영향을 미쳐 Fivetran과 같은 경쟁업체가 무료 계층을 도입하도록 유도했습니다. 또한 새로운 라이브러리와 도구를 탐색하는 데 적극적인 접근 방식을 권장합니다. 예를 들어, [dlt](#) (데이터 로드 도구 - Delta Live Tables와 혼동하지 마세요!)는 상당한 가능성을 보여주는 오픈 소스 라이브러리입니다.

자동차 선택과 유사한 데이터 통합을 고려해보세요. 모든 작업에 Formula One 자동차의 성능이 필요한 것은 아닙니다. 더 흔히 필요한 것은 신뢰할 수 있고 다재다능한 차량입니다. 그러나 Toyota는 99%의 용도를 충족하지만 F1 경주에서 Sienna를 찾을 수는 없습니다. 최적의 전략은? 신뢰할 수 있는 일상 차량에 의존해 필요한 경우 고성능 도구에 대한 액세스를 보장하세요.

제 2 장

데이터 변환

데이터 수집은 단순히 데이터를 A 지점에서 B 지점으로 전송하는 반면, 데이터 변환은 데이터 생명주기의 다양한 단계를 통해 원시 데이터를 귀중한 통찰력으로 전환합니다. 이 장에서는 데이터 실무자가 데이터 변환을 실행하는 데 사용할 수 있는 다양한 언어, 플랫폼 및 기술을 자세히 살펴봅니다.

데이터 변환이 효율적이고 잘 조율된 방식으로 수행되도록 보장하는 방법을 살펴보고 가이드 후반부에서 효율성, 확장성 및 관찰 가능성에 대한 보다 자세한 논의를 위한 토대를 마련할 것입니다.

데이터 변환이란 무엇입니까?

데이터 변환은 사용자와 프로세스에 더 나은 서비스를 제공하기 위해 데이터를 조작하고 향상시키는 기술입니다. 변환에는 원시 상태이든 거의 깨끗한 상태이든 일부 데이터를 가져와서 하나 이상의 작업을 수행하여 의도된 용도에 더 가깝게 이동하는 작업이 포함됩니다. ETL 파이프라인에서는 변환이 한 곳이 아닌 여러 곳에서 발생합니다.

데이터는 수집 시 변환될 수 있으며 다운스트림 지점 수에 관계없이 다시 변환될 수 있습니다. 데이터 변환의 목표는 데이터를 자산으로 전환하는 것입니다. 즉, 분석과 과학을 사용하여 비즈니스에 가치 있는 무언가를 창출하는 것입니다.

변환은 필터링과 같이 원치 않는 레코드를 제거하는 것처럼 간단할 수도 있고 소스 데이터를 완전히 재구성하는 것처럼 복잡할 수도 있습니다. 스펙트럼에는 변환이 존재합니다. 데이터를 변환하는 방법은 거의 무한합니다. 이것이 바로 사물의 흥미를 유지하는 이유입니다!

마찬가지로 변환은 클라우드 제공업체, 서버 또는 소스의 경계에 구애받지 않고 모든 기술과 언어로 조율될 수 있습니다. 이는 계획된 Spark 작업, 람다 함수, SQL 워크플로 또는 Python, JavaScript, Rust, Scala, R 또는 (숨을 참고) 스프레드시트 등 다른 것으로 나타날 수 있습니다. 우리는 익숙하고 대중적인 언어를 선택하는 것을 응호하지만, 데이터 "변환" 관행의 편재성을 언급하지 않는다면 우리는 태만해질 것입니다.

이 장에서는 변환이 무엇인지, 실무자로서 데이터를 변환하는 방법에 대한 개요를 제공합니다. 그런 다음 데이터 변환 및 업데이트의 일반적인 패턴부터 시작하여 스트리밍 변환에 대한 모범 사례와 고려 사항을 계속해서 변환 솔루션을 구축하는 과정을 살펴보겠습니다. 우리의 목표는 효율적인 변환 시스템을 구축하기 위한 일련의 고려 사항을 제공하고 이러한 지식을 귀하의 고유한 문제에 적용할 수 있는 도구를 제공하는 것입니다.

우리는 지금 어디에 있는가?

끊임없이 진화하는 데이터 엔지니어링 환경에서 데이터 변환의 역사는 성장, 단순화, 적응의 이야기입니다. 인터넷 초기에는 Google, Yahoo와 같은 거대 기업이 선구자였습니다. 그들은 Hadoop 및 MapReduce와 같은 빅 데이터 프레임워크로 혁신을 이루었습니다. 그러나 이러한 기술은 대학원 수준의 선형 대수학보다 약간 덜 복잡했습니다.

업계가 성숙해짐에 따라 도구 사용이 단순화되고 민주화되었습니다. Python, SQL 및 Scala의 API를 사용하여 간소화된 분산 엔진을 제공하는 Spark가 등장했습니다. 얼마 지나지 않아 Databricks는 Spark 배포를 단순화 하려는 사명을 가지고 등장했습니다. 동시에 데이터 웨어하우스가 체육관에 등장했습니다. BigQuery, Redshift 및 Snowflake는 확장 가능한 기술로 등장하여 결국 스토리지와 컴퓨팅을 분리했습니다. Presto, Trino 및 Athena가 경쟁에 합류하여 빛처럼 빠른 쿼리 성능을 제공하고 SQL을 빅 데이터 언어로 확장했습니다.

강력한 데이터 웨어하우스, 클라우드 스토리지의 유연성, 변환 도구의 가용성이 결합되어 데이터에 대한 접근성이 그 어느 때보다 높아졌습니다. 오늘날 레이크하우스는 통합되고 비용 효율적인 방식으로 대규모 데이터를 관리하기 위한 추가 옵션으로 등장했습니다.

데이터를 어떻게 변환합니까?

앞서 언급했듯이 데이터 변환은 본질적으로 광범위한 주제입니다.

변환 환경, 언어, 프레임워크 및 모범 사례를 논의하여 데이터 변환의 구조와 범위를 가져오려고 합니다.

환경

데이터를 변환하는 위치에 따라 데이터 변환 방법이 결정되는 경우가 많습니다.

데이터 변환을 위한 가장 일반적인 환경은 다음과 같습니다.

데이터 웨어하우스 워어하우스

우스의 변환은 SQL을 사용하여 수행됩니다.

최신 데이터 워어하우스에는 구체화, 증분 논리, 파티셔닝 등 다양한 데이터의 변환을 가능하게 하는 몇 가지 흥미로운 기능이 있습니다. 서비스 데이터 워어하우스는 집약적인 워크로드를 처리하기 위해 동적으로 확장할 수 있으므로 구조화된 대규모 데이터 세트의 데이터 변환에 적합합니다.

데이터 레이크

데이터 레이크는 대량의 데이터를 경제적으로 저장하는 데 탁월하므로 스테이징을 위한 주요 영역입니다. 데이터 워어하우스와 달리 데이터 레이크에는 컴퓨팅 리소스가 없으므로 외부 도구를 사용하여 변환을 조정하고 실행해야 합니다.

서비스.

데이터 레이크하우스 레

이크하우스는 데이터 레이크와 워어하우스의 측면을 결합하여 잠재적으로 더 낮은 비용으로 더 큰 유연성을 갖춘 솔루션을 제공할 수 있습니다. 실무자는 Apache Spark와 같은 서비스를 활용하여 PySpark 또는 Spark SQL에서 데이터를 변환하여 데이터 레이크의 컴퓨팅 문제를 해결할 수 있습니다. Databricks SQL은 Databricks 레이크하우스 플랫폼 위에 위치하는 워어하우스로, 서비스 데이터 워어하우스의 이점을 제공합니다.

데이터 레이크, 워어하우스 및 레이크하우스 중에서 선택하는 것은 프로젝트의 특정 요구 사항, 팀의 전문 지식, 조직의 장기 데이터 전략에 따라 달라지며, 각각은 고유한 도구 세트를 사용하여 데이터 변환의 다양한 측면을 충족합니다. 그리고 도전.

변환이 발생할 위치를 선택할 때 고려해야 할 몇 가지 질문은 다음과 같습니다.

- 특히 데이터 변환의 복잡성과 관련하여 데이터 레이크, 웨어하우스 및 레이크하우스 유지 관리 비용을 어떻게 비교합니까?
- 귀하의 팀이나 조직의 구조에 더 적합한 플랫폼이 있습니까? 현재 및 예상 워크플로를 고려할 때 유지 관리, 개발 또는 확장에 가장 적합한 것은 무엇입니까? • 데이터 보안 및 규정 준수에 대한 고려 사항은 무엇입니까?

데이터 플랫폼을 선택할 때는 예산, 팀의 기술과 경험, 현재와 미래의 요구 사항을 모두 고려하세요.

장기적인 성공을 위해서는 확립된 관행과 최첨단 기술 간의 균형이 매우 중요합니다.

데이터 스테이

정 변환 사이에 데이터가 스테이징되는 경우가 많습니다. 임시 상태로 적절한 위치(종종 클라우드 스토리지 또는 중간 테이블)에 기록됩니다. 단계적 데이터는 일시적 또는 무기한 보관될 수 있지만 변환에서 중요한 역할을 합니다.

레이크와 웨어하우스 접근 방식을 취하는 순수한 레이크하우스 구현을 택하든 관계없이 데이터를 어떻게 준비할 것입니까?

메달리온 아키텍처 (그림 2-1)은 세 가지 별개의 "레이어"를 설명하는 접근 방식입니다. 원시를 위한 브론즈, 조명 변환을 위한 실버, "클린"을 위한 골드입니다.

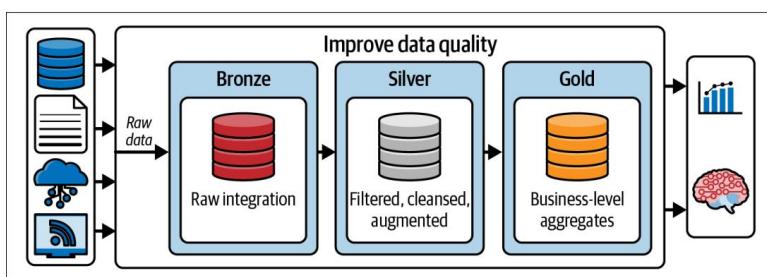


그림 2-1. 메달리온 아키텍처는 세 가지 별개의 데이터 준비 계층을 촉진합니다.

메달리온 아키텍처의 용어를 사용하면 단계적 데이터는 일반적으로 다음 패턴을 따릅니다.

- 브론즈 데이터는 필터링되지 않은 원시 데이터이며 종종 통합 소스에서 직접 가져온 것입니다. 예를 들어 원시 테이블은 API에서 직접 작성될 수 있습니다.
- 실버 데이터는 필터링, 정리 및 조정됩니다. 불필요하고 관련 없는 정보를 제거하고, 일관성을 보장하기 위해 변환을 적용하고, 필요에 따라 데이터를 강화할 수 있습니다.
- 골드 데이터는 일반적으로 이해관계자가 사용할 수 있으며 때로는 집계됩니다. 소비자(분석, 엔지니어링)는 대부분의 시간 동안 Gold 테이블을 쿼리해야 하며 Silver 테이블은 가끔씩만 필요합니다.
- 일반적으로 프로덕션 자산(시각화, 보고서)은 골드 테이블 이외의 다른 항목에 도달해서는 안 됩니다.

호수 하류의 창고에서도 변환된 데이터 수준을 나타내는 스테이징, 선별 및 마트 테이블과 유사한 접근 방식을 사용할 수 있습니다. 이것은 **시간 테스트를 거쳤습니다**. 데이터를 변환하는 방법과 접근 방식 **dbt** 현재 추천합니다.

따라서 변환 준비를 위한 아키텍처 패턴은 각 스토리지 계층을 가져와서 데이터 정리의 개별 "단계"로 나누는 것입니다. 이 접근 방식의 결합은 Delta Lake와 같은 스토리지 기능을 활용하는 것입니다. 어쨌든 **쓰기-감사-게시** 와 같은 패턴은 어떤 단계에도 적용할 수 있습니다.

언어 물론,

데이터를 변환하는 방법은 주로 사용 가능한 도구에 따라 결정됩니다. 우리의 가장 큰 선택인 프로그래밍 언어는 우리가 솔루션에 도달하는 데 사용하는 라이브러리, 프레임워크 및 방법에서 중요한 역할을 합니다. 다음은 데이터 분야에서 가장 인기 있는 몇 가지 라이브러리입니다.

Python

Python은 디지털 시대의 진화하는 환경, 특히 데이터 과학 및 혁신 분야에서 확고한 선택으로 남아 있습니다. 그림 2-2는 지난 10년간 Python의 지배력을 보여줍니다. Python의 데이터 과학 기능의 핵심은 데이터 조작을 위한 다양한 I/O 라이브러리로 강화된 Pandas 라이브러리입니다. 역사적으로 대규모 데이터 세트에 맞게 Python을 확장하는 것은 어려운 일이었으며 종종 Dask 및 Ray와 같은 라이브러리를 사용해야 했습니다. 그러나 최근 발전으로 인해 Python 기반 데이터 처리가 크게 발전 했으며, 이를 분야의 르네상스로 보고 있습니다.

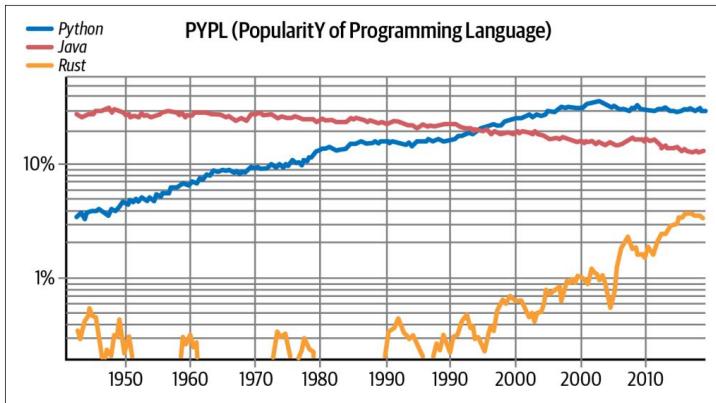


그림 2-2. Rust가 증가하고 있지만 Python은 계속해서 우위를 점하고 있습니다
(PYPL에서 채택). y축 값은 로그 스케일입니다.

Python에서 데이터를 변환할 때 첫 번째 단계는 적합한 라이브러리를 선택한 다음 변환을 조정하고 실행하기 위한 프레임워크를 선택하는 것입니다. 견고함으로 유명한 Pandas로 시작하는 것이 좋습니다. 그러나 Polars 및 DuckDB와 같은 신호 라이브러리도 고려할 가치가 있습니다. 이 장에서는 다양한 변환 기술을 살펴보고, 3 장에서 오케스트레이션에 대해 더 자세히 살펴보겠습니다.

SQL

SQL은 이제 두 번째로 등장하는 용어인 선언적 언어라는 이름을 자랑합니다(1 장 참조). 이 제목은 언어가 원하는 결과를 달성하기 위한 단계별 프로세스를 설명하는 것 보다 원하는 결과를 설명하는 데 더 가깝다는 것을 의미합니다. 예를 들어 `SELECT * FROM views.active_users` 문은 전통적인 의미의 코드가 아닙니다. “활성 사용자를 모두 불러와주세요”라는 정중한 요청에 가깝습니다.

우리는 SQL이 선언적 또는 명령형 언어로 사용될 수 있다고 주장합니다. 우리는 꽤 혁신적인 일부 쿼리를 보았지만(좋든 나쁘든) SQL은 여전히 본질적으로 다음과 같은 언어의 기능 부족으로 인해 제한됩니다. 파이썬은 그렇지 않습니다.

이러한 본질적인 제한으로 인해 대부분의 SQL 변환이 템플릿 솔루션과 결합되어 확장 가능한 SQL 변환이 촉진됩니다. Jinja/Python(dbt를 통해) 및 JavaScript(Dataform을 통해)와 같은 언어는 종종 SQL 워크플로를 보완합니다.

대부분의 SQL 변환은 데이터 웨어하우스에서 발생하거나 데이터 웨어하우스의 컴퓨팅을 활용하여 쿼리를 실행합니다.

녹

Rust는 많은 사람들이 데이터 엔지니어링의 미래라고 주장하는 떠오르는 변환 언어입니다. Mozilla에서 개발한 이 제품은 2010년에 처음 출시되었습니다. 이 제품의 가장 큰 장점은 Python에 비해 속도가 빠르고 강력한 형식의 특성으로 인해 생산 작업 흐름에 유리합니다. 그러나 Rust는 상대적으로 새로운 것이며 Python은 광범위한 커뮤니티 지원, 풍부한 라이브러리 생태계 및 산업 전반에 걸친 광범위한 채택을 통해 견고한 기반을 갖추고 있습니다.

Rust를 탐색하는 것이 권장되지만, 프로덕션 서비스의 경우 잘 채택된 언어를 고수하는 것이 좋습니다. Rust에 대한 실용적인 접근 방식은 Polars처럼 Rust로 작성된 Python 라이브러리를 사용하는 것일 수 있습니다.

프레임워크

변환 프레임워크는 기계나 클러스터 전반에 걸쳐 데이터 변환을 실행하기 위한 다국어 엔진입니다. 병렬화와 분산 컴퓨팅이 표준이 되면서 이러한 프레임워크는 대규모 혁신을 주도했습니다.

프레임워크의 이점은 워크플로 실행을 위한 여러 API가 존재한다는 것입니다. 즉, Python이나 SQL과 같은 다양한 언어로 변환을 조작할 수 있는 경우가 많습니다. 가장 널리 사용되는 두 가지 엔진인 Hadoop과 Spark에 대해 논의하겠습니다. 하지만 데이터 웨어하우스를 엔진으로 생각할 수도 있습니다.

Hadoop

Apache Hadoop은 빅데이터 처리를 위한 획기적인 솔루션으로 등장한 오픈 소스 프레임워크입니다. Apache Spark의 역사, 대기업의 사용 및 현재 상태는 Apache Spark의 등장 및 서비스 데이터 웨어하우스의 출현과 밀접하게 얹혀 있습니다. Hadoop은 2000년대 중반 Yahoo, Facebook, 이후 Google과 같은 거대 기술 기업이 늘어나는 데이터 볼륨을 관리하기 위해 Hadoop을 채택하면서 상당한 관심을 끌었습니다.

Hadoop은 대규모 데이터를 처리하는 데 중요한 역할을 했지만 한계도 있었습니다. 일괄 처리에 최적화된 Hadoop의 Map -Reduce는 실시간 또는 반복 작업 부하에 적합하지 않았습니다.

이로 인해 2010년대 초 Apache Spark가 등장하게 되었습니다.

인메모리 처리, 더 빠른 분석, 배치, 스트리밍, 기계 학습을 포함한 다양한 워크로드 지원.

Hadoop은 여전히 관련성을 유지하고 있지만 그 지배력은 약해졌습니다. 많은 조직이 클라우드 기반 솔루션과 서비스 데이터 웨어하우스를 기반으로 구축된 하이브리드 데이터 아키텍처로 전환했습니다.

Spark

Apache Spark는 속도, 다양성 및 핵심 기술과의 통합을 제공하여 빅 데이터 분석에 혁명을 일으킨 강력한 오픈 소스 데이터 처리 프레임워크입니다. Hadoop의 역사 는 MapReduce의 한계에 대응하여 버클리 캘리포니아 대학에서 개발되었으므로 Hadoop의 MapReduce에 대한 대안의 필요성과 밀접하게 연관되어 있습니다. Spark의 주요 혁신은 **탄력적인 분산 데이터 세트**입니다. (RDD), 메모리 내 데이터 처리 및 더 빠른 계산을 가능하게 합니다.

Apache Spark는 또한 최초 제작자가 설립한 회사인 Databricks와 깊은 관계를 공유하여 데이터 엔지니어, 과학자 및 분석가를 위한 통합 분석 플랫폼을 제공합니다.

Databricks는 Spark 채택을 단순화하고 협업을 강화하여 최신 데이터 분석의 종주적인 역할을 합니다.

Java, Scala, Python 및 R의 고급 API를 통해 거의 모든 데이터 전문가가 Spark에 액세스할 수 있습니다. Spark에서 데이터 변환을 시작하려면 진입 장벽이 낮고 시작하기 쉬운 Databricks와 같은 관리형 플랫폼을 고려하는 것이 좋습니다.

데이터베이스/SQL 엔진 서버

리스 데이터 웨어하우스가 등장하면서 빅 데이터 엔진이 여전히 필요한지 (타당하게) 의문이 생길 수 있습니다. 서비스 데이터 웨어하우스는 예상 수요에 맞게 확장하거나 축소할 수 있습니다. 데이터 레이크/레이크하우스 및 외부 테이블의 적절한 사용과 결합하면 대량의 데이터를 처리하고 변환하는 매우 효율적인 방법이 될 수 있습니다. 뿐만 아니라 SQL의 단순성과 편재성은 다른 기술의 복잡성과 극명한 대조를 이룹니다.

특정 시점이 되면 팀에서 Spark(또는 Hadoop)를 활용해야 할 가능성이 높지만 BigQuery, Databricks SQL, Redshift와 같은 쿼리 엔진은 특히 데이터 팀의 초기 단계에서 실행 가능한 솔루션으로 무시되어서는 안 됩니다.

최근 인메모리 컴퓨팅(MotherDuck 및 DuckDB 등)의 발전으로 인해 데이터 웨어하우스의 변환 기능이 지속적으로 확장될 가능성이 높습니다. 역사를 편들고 관계형 데이터베이스와 평범한 SQL의 내구성이나 확장성을 의심하지 마십시오.

다른 접근법

기존 변환 방법의 대안은 코드가 적거나 없는 플랫폼을 사용하는 것입니다. 사용자 친화적인 인터페이스로 인해 이러한 플랫폼은 기술 전문 지식이 부족한 조직에 특히 적합합니다. 이러한 플랫폼은 접근성을 제공하지만 맞춤화, 확장성 및 제어 기능이 저하되는 경우가 많으므로 모든 사람에게 적합한 것은 아닙니다.

그래픽 사용자 인터페이스(GUI) 도구에만 의존하면 특정 요구 사항에 맞게 소프트웨어를 미세 조정하는 능력이 방해를 받아 비효율성이나 기능 격차가 발생할 수 있습니다. 또한 GUI 기반 솔루션은 투명성이 낮고 버전 관리가 어려워 다른 개발자와의 협업이 어려울 수 있습니다.

BI 도구나 Google Sheets와 같은 기타 다운스트림 변환 방법은 대부분 엔지니어링 영역을 벗어납니다. 분석이나 특정 이해관계자에게는 적합하지만 해당 솔루션의 소유권은 사용자, 즉 분석가에게 직접 있는 것이 좋습니다. 왜? 이러한 솔루션은 생산화 능력이 부족한 경우가 많습니다. CI/CD(지속적 통합/지속적 배포), 버전 제어 및 코드 변환이 부족하여 소프트웨어 개발에 적합하지 않습니다.

혁신 솔루션 구축

지금까지 우리는 데이터 변환을 위한 언어, 환경, 프레임워크 등 퍼즐 조각에 대해 논의했습니다. 그런데 이러한 조각들을 어떻게 모아 가치 있는 것을 만들 수 있을까요?

다음 섹션에서는 최신 데이터 파이프라인에 사용되는 구조화된 데이터에 대한 변환 패턴과 모범 사례를 논의합니다.

데이터 변환 패턴

이 섹션에서는 프로덕션 환경에서의 적용 예와 함께 가장 일반적인 데이터 변환 패턴 모음을 제공합니다. 데이터 변환은 주로 패턴입니다.

매핑 연습 - 어떤 변환이 존재하는지, 언제 적용해야 하는지 이해하는 것이 대부분의 작업입니다.

풍부하게 함

여기에는 추가 소스를 사용하여 기존 데이터를 향상시키는 작업이 수반됩니다. 예를 들어 보다 완전한 그림을 제공하기 위해 고객 기록에 인구통계 정보를 추가하는 등이 있습니다. 강화에는 내부 구조 조인이나 외부 소스에서 추가 데이터 가져오기가 포함될 수 있습니다.

예: 원시 주문 테이블에는 주문 상태를 내부 코드로 나타내는 상태 열이 있습니다. 이는 분석을 혼란스럽게 만드는 사람이 읽을 수 없는 코드입니다. 상태 코드를 강화하기 위해 주문에 참여합니다: "1337"에서 "완료"까지.

조인 조인

에는 SQL의 JOIN 작업과 유사하게 공통 필드를 기반으로 두 개 이상의 데이터 세트를 결합하는 작업이 포함됩니다. 조인은 서로 다른 소스의 데이터를 통합할 때 필수적이며 데이터 아키텍처에서 중추적인 역할을 합니다. 분석을 개선하려면 어떤 팩트 테이블과 차원 테이블을 조인해야 합니까? 어느 것이 서로 다른 것으로 남아 있어야 합니까? 다양한 JOIN 유형을 이해하는 것이 가장 중요합니다.

예: 분석 팀과 대화하면서 보고서에서 주문 국가별로 판매 데이터를 자주 분석하지만 해당 데이터가 정리된 판매 테이블에 없다는 사실을 알게 되었습니다. 판매 데이터를 사용자 데이터와 결합하면 원래 국가를 가져올 수 있으며 분석 팀의 시간과 혼란을 줄일 수 있습니다.

필터링 필터

링 시 특정 기준에 따라 분석에 필요한 데이터 요소만 선택합니다. 이를 통해 대상 시스템에 로드되는 데이터의 불량률을 줄이고 품질을 향상시킬 수 있습니다.

예: 새 시스템에는 2024년 이전 연도의 데이터가 포함되어서는 안 되지만 일부 잘못된 기록이 존재하는 것을 발견했습니다. 잘못된 기록을 제거하기 위해 데이터를 필터링합니다. `SELECT * FROM buys WHERE date >= '2024-01-01'.`

구조화 구조화

에는 데이터를 필요한 형식이나 구조로 변환하는 작업이 포함됩니다. 이는 JSON 문서를 표 형식으로 변환하거나 그 반대로 변환하는 것을 의미할 수 있습니다.

예: 금융 데이터 API는 JSON 데이터를 반환합니다. 데이터의 압축을 풀고 구조화한 다음 S3의 Parquet 데이터 저장소에 추가합니다.

변환

변환은 문자열을 날짜/시간 형식으로 변환하거나 정수를 부동 소수점 형식으로 변환하는 것과 같이 특정 열이나 필드의 데이터 유형을 변경하는 것입니다. 이는 특히 반구조화된 데이터 소스와 구조화된 데이터 소스 간을 변환할 때 가장 일반적인 변환 유형 중 하나입니다.

예: 업스트림 API는 "2011-01-01 00:00:00" 문자열 형식의 타임스탬프를 반환하지만, 쉬운 검색과 시각화를 위해 타임스탬프를 데이터베이스의 해당 형식으로 캐스팅합니다. 보다 발전된 날짜/시간 변환에는 [Unix 타임스탬프가 포함될 수 있습니다](#). 단순히 캐스팅할 수는 없습니다.

집계 집계는 특정

기간 동안의 총 매출이나 값 집합의 평균을 계산하는 등 데이터를 요약하고 결합하는 것입니다. 대량의 데이터에서 결론을 도출하려면 데이터 집계가 필수적입니다. 엔지니어로서 우리는 종종 Shift-right 집계를 시도하고 가능한 한 분석에 가깝게 만들려고 노력합니다.

이를 통해 데이터를 가장 정밀하게 분석할 수 있습니다. 집계의 목표는 방대한 양의 데이터에서 통찰력을 얻고 비즈니스 결정을 알리고 데이터 자산에서 가치를 창출할 수 있도록 하는 것입니다.

예: 서비스 팀에서 배포한 IoT 센서 장치로부터 밀리초 수준의 데이터를 수신합니다. 엄청나게 많은 양의 정보로 인해 주어진 시간에 이 데이터의 작은 하위 집합을 저장하고 두 번째 데이터를 다운스트림 테이블에 집계합니다.

익명화 익명화는 개인정보

보를 보호하기 위해 데이터 세트 내의 민감한 정보를 마스킹하거나 난독화하는 것입니다. 데이터 실무자로서 우리는 법적, 윤리적 이유로 사용자의 개인정보를 보호하기 위해 최선을 다해야 합니다. 익명화는 다음과 같습니다.

이메일을 해상하거나 기록에서 개인 식별 정보(PII)를 삭제하는 행위.

예: 민감한 고객 정보가 다운스트림 데이터 사용자에게 도달하는 것을 방지하기 위해 이메일은 해시되어 고유 식별자로 저장됩니다. 정보는 액세스가 매우 제한된 테이블에 보관됩니다.

분할 단일

복잡한 데이터 열을 여러 열로 나누는 분할은 간단한 형태의 비정규화로 생각할 수 있습니다.

예: 데이터를 비공개로 유지하면서 사용자의 이메일 도메인에 대한 정보를 제공하려고 합니다. 이메일을 접두사 및 도메인 열로 분할하고 접두사를 삭제하여 도메인을 유지하면서 데이터를 익명으로 유지합니다.

중복 제거 데이터 정

규화의 일부인 중복 제거 프로세스는 중복 레코드를 제거하여 고유한 데이터 세트를 만드는 작업입니다.

중복 제거는 집계, 필터링 또는 기타 방법을 통해 발생할 수 있습니다.

예: 이벤트 스트림은 단일 발생에 대한 중복 이벤트(동일한 범용 고유 식별자(UUID) 포함)를 선택하는 경우가 있습니다. 이해관계자들은 가장 빠른 기록을 보관해야 한다는 데 동의합니다. 중복된 이벤트 UUID를 삭제하는 변환이 생성됩니다.

데이터 업데이트 패턴 데이터 변환의

일부는 대상 시스템에 이미 존재하는 데이터를 업데이트하는 방법을 이해하는 것입니다. 여기서는 원활한 변환 프로세스를 보장하기 위해 데이터를 업데이트하는 방법을 제시합니다.

덮어쓰기 데이

터를 업데이트하는 가장 간단한 형태는 기존 소스나 테이블을 완전히 삭제하고 완전히 새로운 데이터로 덮어쓰는 것입니다. 이는 가장 간단한 형태의 데이터 업데이트이지만 데이터 크기가 증가하면 빠르게 유지 불가능해질 수 있습니다.

하지만 단순함을 고려하면 좋은 출발점이 될 수 있습니다.

삽입 데

이터를 덮어쓰는 대신 기존 소스에 새 데이터를 추가하도록 선택할 수도 있습니다. 삽입하는 과정입니다. 데이터를 삽입하면 새 행만 추가됩니다. 기존 행은 변경되지 않습니다.

이상적인 삽입 사용 사례는 새 데이터가 이전 데이터와 완전히 독립적인 경우입니다. 예를 들어 특정 날짜의 거래 테이블이나 주식 시장 데이터 등이 있습니다. 이러한 경우 단순히 새 레코드를 점진적으로 삽입하는 것이 최신 데이터 원본을 유지하는 빠르고 간단한 방법입니다.

Upsert

업데이트 및 삽입의 약자인 Upsert는 변경 데이터 캡처, 세션화 및 중복 제거를 위한 애플리케이션이 포함된 보다 복잡한 업데이트 패턴으로, 복잡하기는 하지만 특히 유용합니다.

변경된 데이터는 미리 정의된 고유 식별자로 식별되며, 해당 레코드가 업데이트 및/또는 삽입됩니다. DIY upsert 변환의 본질적인 복잡성을 고려할 때 Databricks 와 같은 플랫폼에는 프로세스를 대폭 단순화하는 MERGE 기능이 있습니다.

Databricks의 다음 예에서는 UPSERT를 잘 보여줍니다.

```
people10m.id =
people10mupdates.id에서
people10mupdates를 사용하여 people10m 으로 병합
일치 했을 때
    업데이트 세트
        id = people10mupdates.id, firstName =
        people10mupdates.firstName, lastName = people10mupdates.lastName,
        birthDate = people10mupdates.birthDate, ssn =
        people10mupdates.ssn, 급여 = people10mupdates.salary 일치 하지 않
            는 경우
```

그런 다음 INSERT (ID, 이

름,
성, 생년월일,

sns, 월
급
)
값 (

```

    people10mupdates.id,
    people10mupdates.firstName,
    people10mupdates.lastName,
    people10mupdates.birthDate,
    people10mupdates.ssn,
    people10mupdates.salary
)

```

삭제 데이터

데이터 엔지니어링에서는 데이터 삭제의 개념을 잘못 이해하는 경우가 많습니다. 삭제에는 "하드"와 "소프트"라는 두 가지 주요 삭제 유형이 있습니다. 일시 삭제는 레코드를 "삭제됨"으로 표시하는 것과 관련될 수 있는 반면(예: 상태 설정 = "삭제됨") 영구 삭제는 데이터베이스에서 레코드를 완전히 제거하는 것을 의미합니다.

일반 데이터 보호 규정과 같은 개인정보 보호 규정에서는 영구 삭제를 선호하는 경향이 있습니다.

그러나 일시 삭제의 유용성을 간과해서는 안 됩니다.

예를 들어 디지털 자산을 관리하는 SaaS(Software as a Service) 회사에서는 일시 삭제를 통해 자산 상태에 대한 기록 기록을 생성할 수 있습니다. 이는 사용자와의 커뮤니케이션 유지, 계정 종료 분석 및 기타 운영 측면에 매우 중요할 수 있습니다. 반면, 영구 삭제는 이러한 기록 레코드를 제거하므로 데이터 복구 시 문제가 될 수 있습니다.

모범 사례

변환은 시스템에 따라 상당히 다르게 보일 수 있지만 일부 테마는 상수로 나타납니다. 어디에 위치하든 변환 솔루션을 구축할 때 다음 모범 사례를 고려하십시오.

스테이징

우리는 메달리온 아키텍처를 사용하여 호수나 창고에서 데이터를 스테이징하는 방법에 대해 이미 논의했지만 다시 반복할 가치가 있습니다. 데이터 손실을 방지하고(복구 가능성 향상) 장애 발생 시 짧은 복구 시간(TTR)을 보장하기 위해 데이터가 변환될 때마다 준비를 고려해야 합니다.

멱등성은 일관성

과 신뢰성을 보장하는 기본 개념입니다. 멱등성은 훌륭한 어휘이지만 약간 불투명해 보일 수 있습니다. 그러나 실제로는 간단한 개념입니다.

어떤 일을 여러 번 하면 일관된 결과가 나옵니다. 그런 의미에서 멱등성은 '재현성'과 같습니다. 새로운 소스 데이터 없이 파이프라인을 두 번 실행한다고 가정해 보겠습니다. 출력 데이터가 동일하게 보입니까? 멱등성은 분산 시스템에서 오류를 처리하고 데이터 일관성을 보장하는 데 중요합니다.

정규화 및 비정규화

이러한 용어는 새로운 데이터 실무자에게 혼란스러울 수 있습니다(처음 들었을 때 혼란스러웠다는 것을 알고 있습니다). 정규화에는 데이터를 우리가 기대하는 깨끗하고 질서정연한 형식으로 정제하는 작업이 포함됩니다. 이는 빠르게 복잡해질 수 있으므로 간략하게 설명하겠습니다. 정규화된 데이터는 고유하며 때로는 기본 키를 갖습니다.

비정규화는 정반대입니다. 여기에는 시스템 성능을 높이기 위해 기록과 정보를 복제하는 작업이 포함됩니다. 대규모 데이터 시스템에서는 중복 요소를 활용하여 분석 시스템 또는 다운스트림 요소의 성능을 향상시킬 수 있습니다.¹

증분성 데이터 업데이트

이트 주제에서 증분성은 파이프라인이 단순한 INSERT OVERWRITE 인지 아니면 더 복잡한 UPSERT인지를 정의합니다. 비증분 워크플로가 가능한지 여부는 볼륨, 컴퓨팅 및 비용에 따라 크게 결정됩니다. dbt 와 같은 도구에는 증분 워크플로를 구축하기 위한 사전 정의된 패턴이 많이 있습니다. 그리고 [공기 흐름. 데이터브릭스](#) 또한 증분 파이프라인을 위한 강력한 리소스 라이브러리도 있습니다.

데이터베이스에 데이터를 로드하는 경우 INSERT 대신 MERGE를 사용하면 놀라운 결과를 얻을 수 있습니다. 데이터를 레이크에 로드하는 경우 "상태 저장 처리" 또는 "북마크"를 사용하는 것, 즉 변환이 중단된 위치를 추적하는 것이 유용할 수 있습니다.

실시간 데이터 변환

[1 장의](#) 실시간 데이터에 대한 논의에서 배치, 마이크로 배치, 스트리밍 변환의 차이점을 제시했습니다. 여기서는 지역 시간이 충분히 낮을 때 마이크로 배치 변환이 효과적인 스트리밍 변환이 될 수 있다는 점만 알아두겠습니다.

1 정규화 및 비정규화에 대한 자세한 내용은 "[데이터 정규화 설명: 데이터 정규화 방법](#)"을 참조하세요. 및 "[데이터 비정규화: 전체 가이드](#)".

스트리밍 데이터 변환은 데이터가 생성되거나 수신될 때 실시간으로 데이터를 처리하는 것을 의미합니다. 진정한 스트리밍 변환(Beam, Flink, Kafka)은 대부분의 경우 준비되지 않은 수준의 복잡성을 가져올 수 있습니다. 꼭 필요한 경우가 아니라면 Spark와 같은 마이크로 배치 접근 방식을 권장합니다.

Apache Spark는 특히 PySpark 및 Spark SQL 구성 요소를 통해 마이크로 배치/스트리밍 변환을 제공함으로써 이 분야에서 탁월합니다. 이는 창과 같은 실시간 데이터 처리 제약으로 인해 더 복잡한 실제 단일 이벤트 변환에 비해 구현하기가 더 간단합니다.

Spark Structured Streaming은 종분 및 연속 업데이트를 효율적으로 처리하므로 개발자들 사이에서 인기 있는 스트리밍 애플리케이션입니다. 주요 기능으로는 스트리밍 집계, 이벤트 시간 창, 스트림-배치 조인 등이 있으며, 모두 다른 방법보다 더 간단한 방식으로 Spark의 Dataset API를 통해 촉진됩니다.

기술적으로는 마이크로 배치 처리 엔진을 사용하지만 Spark Structured Streaming은 정확히 1회의 내결합성을 통해 100밀리초만큼 낮은 지연 시간을 달성합니다. 이는 "스트리밍"으로 간주될 만큼 충분히 낮습니다. 또한 Spark 2.3에 도입된 연속 처리 기능은 지연 시간을 1밀리초까지 줄여 최소 한 번 보장하고 스트리밍 데이터 변환 기능을 더욱 향상시킵니다.

데이터 변환의 미래

현대 데이터 스택은 **두 번째 르네상스를 맞이하고 있습니다.** 기존 데이터 워크플로우의 단점을 해결하는 새로운 기술이 박차를 가하고 있습니다.

동시에 AI의 발전은 우리가 일하는 방식뿐만 아니라 우리가 구축하는 실제 시스템을 변화시키고 있습니다.

데이터 변환의 의미를 재정의하는 새로운 도구와 기술이 매우 등장하는 것 같습니다. 다행히도 이 장에서 제공되는 개념과 패턴은 시대를 초월합니다. 도구에 관계없이 조직에서 가치를 창출하기 위해 데이터를 관리하고 정리된 자산을 생성하기 위해 오랜 시간 테스트를 거친 전략을 준수해야 합니다.

데이터 변환의 미래는 오늘날의 변환과 비슷하면서도 다릅니다. 효율적인 시스템을 설계하는 데 필요한 전략, 기술, 커뮤니케이션이 비슷하다는 점입니다.

툴링이 강화되고 프로세스가 자동화된다는 점에서 변경되지 않고 다릅니다.
음, 바라건대.

강력한 도구와 자동화는 실무자에게 엄청난 힘을 가져다주기 때문에 축복이
자 저주가 될 수 있습니다.

그러나 큰 힘에는 큰 책임이 따른다는 말이 있습니다. 높은 가치 대비 비용 비
율로 잘 계획되고 실행되는 변환 시스템을 제공할 수 있는지 확인하는 것이 엔
지니어로서 우리의 임무입니다.

3 장

데이터 오케스트레이션

이미 수집(E, L) 및 변환(T)에 대해 논의했지만 ETL의 표면적인 부분만 살펴보았습니다. 데이터 파이프라인을 일련의 개별 단계로 보는 것과는 달리, 데이터 엔지니어링 기초에서 Matt Housley와 Joe Reis가 적절하게 "저류"라고 부르는 메타 수준에서 작동하는 중요한 메커니즘이 있습니다.

- 보안 • 데이
터 관리 • 데이터 운영
(DataOps)
- 데이터 아키텍처
- 데이터 조정
- 소프트웨어 공학

이 장에서는 종속성 관리와 파이프라인 오케스트레이션을 살펴보고 오늘날 특정 오케스트레이션 방법이 인기 있는 이유를 이해하는 데 중요한 오케스트레이터의 역사를 살펴보겠습니다. 자체 데이터 워크플로를 조율하고 오케스트레이션의 몇 가지 일반적인 디자인 패턴에 대해 논의할 수 있는 옵션 메뉴를 제시합니다.

전체적으로 "오케스트레이터"가 역사적으로 "변환" 도구와 어떻게 분리되었는지에 대해 논의할 것입니다. 이것이 사실인 이유와 미래에는 사실이 아닐 수도 있는 이유에 대해 다루겠습니다. 하지만 우리는 여전히 별도의 오케스트레이터가 선호되는 접근 방식이라고 믿습니다.

데이터 오케스트레이션이란 무엇입니까?

데이터가 있든 없든 모든 워크플로우에는 순차적인 단계가 필요합니다. 물을 가열하지 않고 프렌치 프레스를 사용하려고 시도하면 실망스러울 뿐이며, 순서가 잘못된 데이터 변환은 카페인이 부족한 아침보다 훨씬 더 쓰라린 폭풍을 일으킬 수 있습니다. 카페인을 제거한 제품을 하찮게 여겨서는 안 됩니다.) 데이터에서 이러한 '단계'는 작업 및 '워크플로' 또는 방향성 비순환 그래프(DAG)라고 불리는 경우가 많습니다. 이 용어에 대해서는 곧 자세히 살펴보겠습니다.

오케스트레이션은 자동화를 통해 촉진되는 종속성 관리 프로세스입니다. 데이터 오케스트레이터는 스케줄링, 트리거링, 모니터링은 물론 리소스 할당까지 관리합니다. 오케스트레이터는 오로지 cron 기반인 스케줄러와는 확실히 다릅니다.

반면 오케스트레이터는 이벤트, 웹후크, 일정은 물론 워크플로 내 종속성까지 트리거할 수 있습니다. 데이터 오케스트레이션은 다양한 소스의 대규모 데이터를 처리할 수 있는 체계적이고 자동화된 효율적인 방법을 제공합니다.

오케스트레이션은 무엇보다도 파이프라인이 정확하고 시기적절한 결과를 생성하도록 보장하는 것입니다. 훌륭한 오케스트레이터는 효율성, 확장성 및 속도에도 중점을 두어야 합니다. 하지만 곧 논의하겠지만 작업은 대부분 오케스트레이터 외부에서 이루어집니다.

왜 오케스트레이션해야 하는가?

오케스트레이션은 이러한 워크플로를 가능하게 하는 도구 역할을 하는 오케스트레이터를 통해 워크플로를 효율성과 기능성 방향으로 조정합니다.

일반적으로 오케스트레이터는 일정이나 특정 이벤트를 기반으로 파이프라인을 트리거합니다. 이벤트 기반 파이프라인은 예측할 수 없는 데이터나 리소스 집약적인 작업을 처리하는 데 유용합니다. 데이터 엔지니어링 툴킷에 오케스트레이터를 포함하면 얻을 수 있는 혜택은 다음과 같습니다.

워크플로 관리 오케스트레이터

는 워크플로를 효율적으로 정의, 예약 및 관리하는 데 도움을 주며 종속성을 관리하여 작업이 올바른 순서로 실행되도록 합니다.

오토메이션

엔지니어로서 우리는 가능한 한 많이 자동화해야 합니다. 오케스트레이터를 사용하면 일상적이고 반복적이며 복잡한 작업까지 자동화하여 시간을 절약하고 수동 개입 없이 작업이 예정대로 실행되도록 할 수 있습니다.

오류 처리 및 복구 Orchestrator에는

오류 처리 및 복구 메커니즘이 내장되어 있는 경우가 많습니다. 실패한 작업을 다시 시도하거나, 팀에 알리거나, 문제를 해결하기 위해 다른 작업을 트리거할 수 있습니다.

모니터링 및 경고 워크플로를 모니

터링하고 오류 또는 지연에 대한 경고를 받는 것은 안정적인 데이터 파이프라인을 유지하는데 중요합니다. Orchestrator는 이러한 기능을 제공합니다.

리소스 최적화 오케스트레이터

는 작업이 실행되는 시기와 장소를 관리함으로써 리소스 사용을 최적화하는 데 도움을 주며, 이는 리소스가 제한되거나 비용이 많이 드는 환경에서 매우 중요합니다.

관찰 가능성 및 디버깅 Orchestrator

는 워크플로 문제 해결 및 최적화에 매우 중요한 워크플로, 로그 관리 및 기타 디버깅 도구를 시각적으로 표현합니다.

규정 준수 및 감사

오케스트레이터는 모든 작업에 대한 감사 추적을 유지 관리하며 이는 데이터 거버넌스 및 기타 규제 요구 사항을 준수하는 데 중요합니다.

오케스트레이터를 고용하는 것은 강력하고 효율적이며 확장 가능한 데이터 파이프라인을 구축하기 위한 전략적 단계로, 데이터 엔지니어링 프로세스가 잘 조정되고 모니터링되고 관리되도록 보장합니다.

DAG

"방향성 비순환 그래프"는 아마도 데이터 엔지니어링에서 가장 불필요하게 복잡한 용어일 것입니다. (어떤 이유로든) 그래프 이론에서 차용되었으며, 단순히 데이터 실행의 "트리"를 설명하는 데 사용됩니다.

여기서 "작업"은 노드로 표시되고 "종속성"은 가장자리로 표시됩니다 ([그림 3-1](#)). 이 나무들은 다음과 같습니다:

감독

작업 실행은 트리의 한쪽 끝에서 다른 쪽 끝으로 특정 방향을 따르며 이는 그래프 내에 종속성이 있음을 나타냅니다.

비순환 트

리에는 순환이 없습니다. 즉, [그림 3-1](#)의 이미지에서 경로 $1 \rightarrow 2 \rightarrow 5$ 를 실행하면 동일한 파이프라인 내에서 $5 \rightarrow 1$ 을 실행하지 않습니다.

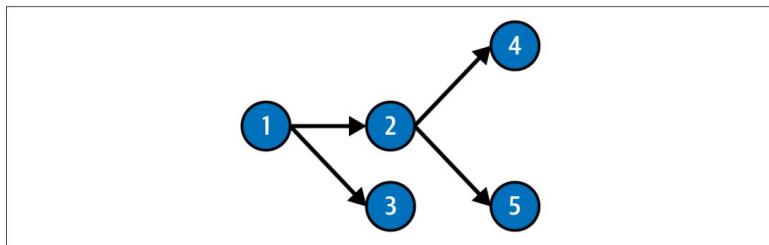


그림 3-1. 매우 간단한 DAG

DAG는 데이터 워크플로에 질서, 제어, 반복성을 제공합니다. 대조적으로, 나의 삶은 방향이 없는 순환 그래프입니다. 혼란스러운 사건은 운율이나 이유 없이 반복적으로 발생하는 것 같습니다. 따라서 저는 적어도 데이터 엔지니어링에 부과되는 표준 DAG에 감사드립니다.

DAG의 주요 목적은 종속성을 관리하는 것입니다. 종속성은 매우 빠르게 매우 복잡해질 수 있습니다. 당신이 그것을 알기도 전에 당신은 혼란이 뒤따를 것이기 때문에 적절한 분기 다음에 그림처럼 보이는 것으로 작업하게 될 것입니다.

데이터 엔지니어링 영역에서 DAG는 파이프라인을 조정하고 시각화하는 데 중추적인 역할을 합니다. 이는 일련의 작업을 매핑하여 구조화되고 예측 가능한 데이터 흐름을 보장하기 위한 청사진입니다. 이는 특히 팀이나 대규모 환경에서 복잡한 작업 흐름을 관리하는 데 없어서는 안 될 요소입니다.

데이터 수집, 정리, 변환 및 최종적으로 데이터베이스에 로드를 위한 작업이 지정된 데이터 파이프라인을 생각해 보세요. DAG는 이러한 작업의 순서를 정의하는 명확한 로드맵 역할을 합니다. 예를 들어 데이터가 정리 및 변환되기 전에 데이터베이스에 조기에 로드되지 않도록 보장합니다.

DAG는 DAG 렌즈를 통해 데이터 파이프라인의 생성, 예약 및 모니터링을 용이하게 하는 Apache Airflow와 같은 워크플로 조정 도구에 의해 대중화되었습니다.

데이터 조정 도구

데이터 조정 도구의 여정은 지난 수십 년 동안의 놀라운 발전에 대한 이야기를 반영합니다. 처음에 데이터 엔지니어는 맞춤형 Python 스크립트부터 적응형 CI/CD 도구에 이르기까지 데이터 워크로드를 조정하기 위해 레거시 또는 임시 변통 솔루션에 의존했습니다. 그러나 데이터 환경이 복잡해지면서 전문 도구에 대한 요구가 더욱 커져 Apache Airflow와 Luigi가 탄생하게 되었습니다.

이 분야의 궤적은 오픈 소스 커뮤니티와 빅 데이터의 요구에 따라 진화의 연속체에 있습니다. Apache Airflow가 지배적인 세력으로 부상했지만 Dagster, Prefect 및 Mage와 같은 대안도 많이 있습니다. 각 도구에는 고유한 장점과 단점이 있으므로 작업에 적합한 도구를 선택하는 것이 중요합니다. 데이터 조정의 미래는 각각 고유한 틈새 시장을 갖춘 다양한 도구 생태계가 될 가능성이 높습니다.

oke스트레이터 선택 지금쯤 알 수 있듯이 저

는 비유를 좋아합니다. OKE스트레이션은 말 그대로 OKE스트라 지휘자가 하는 일이다. 따라서 데이터 OKE스트레이터의 이름은 적절하지만 유사성은 더 깊습니다. 교향악단의 음악가가 데이터 스택의 구성원인 경우 이들이 조화롭게 연주하는지 확인하고 템포, 프레이징 또는 반복을 조정하여 그들의 비전을 팀에 전달하는 것은 지휘자입니다.

여기서 중요한 점은 지휘자가 악기도 연주하지 않는다는 것입니다. 어딘가에서 우리는 OKE스트레이션과 데이터 플랫폼을 혼동했습니다. 공평하게 말하면 이것은 저지르기 쉬운 실수입니다. 예를 들어, Airflow는 설치 및 학습이 충분히 간단하며(관리형 옵션이 제공되어 배포가 훨씬 더 쉽습니다), 일단 모든 것이 구성되면 작성할 수 있는 아름다운 빈 캔버스가 생깁니다. … 순수 Python.

오히려 OKE스트레이터를 목적에 따라 OKE스트레이션하는 것이 경쟁 우위를 가장 효과적으로 활용하는 것입니다. 외부 서비스(Fivetran, dbt, Databricks,

그런 다음 오케스트레이터를 사용하여 이러한 서비스를 적절한 순서로 트리거하고 모니터링합니다. 이후에는 트롬본을 효과적으로 지휘하고 연주할 수 없는데 왜 오케스트라 단원에게 그렇게 하라고 기대합니까?

이 가이드의 일관된 주제는 코드의 유연성과 소프트웨어 엔지니어링 모범 사례의 엄격함 및 효율성을 제공하기 위해 선언적 프레임워크와 명령형 프레임워크의 균형을 맞추는 하이브리드 도구의 필요성이었습니다. 우리는 오케스트레이터도 다르지 않다고 생각합니다. 이러한 프레임워크를 직접 구축하거나 공급업체/플랫폼에 비용을 지불하여 수행할 수 있습니다. 이는 주로 상황에 따라 다릅니다.

형질

위 사항을 염두에 두고 오케스트레이터 선택 시 고려 사항으로 넘어갈 수 있습니다. 명심해야 할 몇 가지 특징은 다음과 같습니다.

확장성 데이터

처리 요구 사항이 증가함에 따라 오케스트레이터는 증가된 로드를 처리할 수 있도록 워크플로를 확장하는 데 도움을 줍니다. 컨테이너화된 오케스트레이션의 이점에 대해 곧 논의하겠지만 오케스트레이터가 수직으로 확장할 수 있는지(병렬 작업 수 증가) 또는 수평으로 확장할 수 있는지(각 작업의 컴퓨팅 증가) 고려하세요.

앞서 언급했듯이 이상적으로 데이터를 조정하는 프로세스는 데이터 변환 프로세스와 분리되므로 확장은 반드시 계산이 아니라 복잡한 논리와 종속성을 처리하는 문제여야 합니다. DAG, 종속성, 작업이 10배 증가하면 워크플로가 어떤 모습일지 생각해 보세요. 관리가 가능한가요?

코드 및 구성 재사용성

보다 성숙한 소프트웨어 개발 영역과 달리 데이터 엔지니어링에는 확립된 모범 사례가 없는 것으로 악명 높습니다.

이는 존재하지 않는다는 의미가 아니라 구현을 보장하기 위해 정직한 노력(을바른 도구 선택, 프레임워크 존재 확인 등)을 하지 않으면 존재하지 않을 것이라는 의미입니다.

오케스트레이터는 코드 및 구성 재사용성을 촉진하여 유사한 서비스에 대한 액세스를 단순화하고 파이프라인 전체에서 공통 작업을 재사용해야 합니다. Airflow에서 이를 관리하는 방법에 대한 좋은 예는 Astro SDK를 참조하세요. 아무것

두 번 이상 수행된 작업은 함수나 클래스로 변환되어야 합니다.

사이

데이터 통합과 마찬가지로 오케스트레이션도 연결에 관한 것입니다. 무슨 뜻인가요? 워크플로의 상당 부분이 특정 플랫폼에서 발생하는 경우 오케스트레이터를 해당 플랫폼에 수용하는 것이 합리적입니다. 결과적으로 모든 주요 클라우드 제공업체는 Amazon(Apache Airflow용 관리 워크플로), Google(Cloud Composer), Azure(Azure Data Factory Managed Airflow) 등 Airflow의 호스팅 버전을 제공합니다.

Databricks Workflows와 같은 Airflow에 대한 플랫폼 내장형 대안은 Databricks 배포를 통해 고객에게 더 적합할 것입니다. 오케스트레이터는 외부 프로세스를 시작하는 역할을 하기 때문에 빈약한 웹후크 대신 광범위한 기본 연결 라이브러리를 보장하면 오케스트레이션 중인 작업에 대한 가시성이 가장 높아집니다.

지원 인기

있는 오케스트레이터는 강력한 커뮤니티 지원과 지속적인 개발을 통해 도구가 최신 기술과 모범 사례로 최신 상태를 유지하도록 보장합니다. 다른 도구와 마찬가지로 오케스트레이터를 조사할 때 얼마나 자주 업데이트를 기대할 수 있는지 자세히 조사해 보세요. 또한 도구의 성숙도를 평가하십시오. 귀하의 팀이 v1 이전 도구를 배포할 수 있습니까? 자주 업데이트하는 것이 가능합니까, 아니면 보다 안정적인 솔루션이 바람직합니까?

비공개 소스 또는 유료 솔루션을 선택하는 경우 지원도 마찬가지로 중요합니다. 팀의 구현을 돋는 솔루션 엔지니어가 있습니까? 문제 해결에 도움이 되는 지원을 받을 수 있습니까? 어떤 경우에는 유사한 구현을 가진 다른 조직에 대한 관점을 고려할 때 이러한 지원 팀이 컨설턴트만큼 가치가 있을 수 있습니다.

관찰 가능성 현대

데이터 스택의 영역은 종종 서로 섞입니다. 오케스트레이터가 반드시 관찰 도구일 필요는 없지만 변환 흐름에 대한 통찰력을 제공해야 합니다. 작업이 실패했나요? 재시도되었나요?

자신의 프로세스에 대해 생각해보십시오. 경보 시스템에 가장 적합한 것은 무엇입니까? Slack 메시지인가요, 이메일인가요, 손글씨인가요?

운송업체 비둘기가 배달하는 편지?¹ 오케스트레이터가 팀에 가장 적합한 방법을 지원하고 데이터 파이프라인 상태에 대해 필요한 가시성을 제공하는지 확인하세요.

오케스트레이터 옵션

현재 오케스트레이션 솔루션을 구현하려면 다음을 수행할 수 있습니다.

솔루션 구축 우리는 이

옵션이 이해되기 위해 필요한 규모가 Uber/Airbnb 수준이라는 단순한 이유 때문에 이 옵션을 옹호하지 않습니다.

기성 도구 구입 이는

DevOps 작업을 추상화하고 배포를 단순화하려는 팀에게 유용한 옵션이 될 수 있습니다.

오픈 소스 도구 자체 호스팅 자체

호스팅할 리소스가 있는 경우 이 옵션은 공급업체 잠금을 방지하는 좋은 방법이 될 수 있습니다. 그러나 자체 호스팅에는 완전히 다른 문제와 골칫거리가 따른다는 점에 유의해야 합니다.

클라우드 제공업체 또는 데이터 플랫폼에 포함된 도구를 사용하세요.

플랫폼에 크게 의존하는 경우 Azure Data Factory 또는 Databricks Workflows와 같은 솔루션은 워크플로를 조정하는 간단하고 효과적인 방법이 될 수 있습니다.

모든 접근 방식에는 장단점이 있습니다. 다양한 클라우드 네이티브 옵션(Amazon Managed Workflows, Google Cloud Composer, Azure Data Factory Managed Airflow)의 기반 역할을 하고 자체 카테고리인 Apache를 보장하는 도구부터 시작하여 몇 가지 옵션에 대해 논의하겠습니다. 기류:

Apache Airflow

Airbnb에서 개발한 Airflow는 활발한 커뮤니티, 대규모 챕택, 모든 주요 클라우드 제공업체의 지원을 통해 사실상의 오케스트레이션 도구로 발전했습니다. 챕택이 쉽고 배포가 간단하며 데이터 공간의 편재성 덕분에 여전히 인기 있는 선택입니다.

1 참고: 귀하가 요하네스 캐플러(Johannes Kepler)가 아니고서는 이 경고 방법을 옹호하지 않습니다. 올해는 1612년이다.

Airflow가 최신 데이터 스택 개발에 중요한 역할을 했을 뿐만 아니라 계속해서 오픈스토리레이션을 위한 효과적인 도구라는 점은 부인할 수 없습니다. 그러나 우리는 이 도구에 내재된 여러 가지 결함을 발견했습니다.

2010년대 초에 구상된 Airflow는 변환이나 수집이 아닌 조정을 위해 설계되었습니다. 천문학자의 칭찬할 만한 노력에도 불구하고, 변환 워크플로의 근본적인 장벽은 원래 청사진으로 인해 Airflow 내에서 지속됩니다.

Airflow는 믿을 수 없을 만큼 쉽게 가동할 수 있지만 대규모로 구축하고 프로덕션 환경에서 구축하는 것은 훨씬 더 어렵습니다. 제약이 없기 때문에 노련한 지침 없이도 종종 표면화되는 단순한 실수가 허용됩니다.

조정 및 변환 공간에서 최신 도구를 확인하여 무엇이 있는지 확인하는 것이 좋습니다. Airflow가 유일한 선택이라면 좋은 소식입니다. Airflow는 풍부한 커뮤니티 지원을 통해 여전히 뛰어난 조정 도구입니다.

기타 오픈 소스 도구 Airflow

를 포함한 많은 유명 오픈스토리레이터는 유료 호스팅 서비스 및 지원 옵션이 포함된 오픈 소스를 기반으로 구축되었습니다. Prefect와 Dagster가 자금 조달과 채택 분야의 선두주자로서 이 분야에서 혁신을 이루고 있는 다양한 최신 도구가 있습니다. Mage, Keboola 및 Kestra와 같은 최신 v1 이전 도구가 많이 증가하고 있습니다.

오픈 소스 옵션을 선택하는 것은 커뮤니티 지원, 소스 코드 직접 수정 기능 등 여러 가지 이유로 매력적일 수 있습니다. 물론, 오픈 소스 도구는 지속적인 개발을 위해 이러한 지원에 크게 의존합니다. 또한 v1 이전 소프트웨어를 선택하면 빠른 업데이트와 새로운 기능을 얻을 수 있지만 프로젝트가 포기되고 불안정해질 위험이 있습니다.

솔루션을 고려할 때 이력, 지원 및 안정성을 고려하십시오. 이를 현재와 미래의 팀 요구 사항과 비교해 보세요. 많은 오픈 소스 도구는 유료 호스팅 및 지원 옵션을 제공합니다. 귀하의 팀이 호스팅된 인스턴스를 관리하는 DevOps 작업을 오프로드할 수 있는 위치에 있다면 조사해 볼 가치가 있을 수 있습니다. 그렇지 않은 경우에는 선택한 도구의 지원 옵션(커뮤니티)과 배포의 복잡성을 이해해야 합니다.

유료 비공개 소스 도구

관리형 오픈 소스 옵션과 마찬가지로 유료 솔루션에는 응답 시간에 대한 사전 정의된 서비스 수준 계약(SLA) 지원, 가능한 아키텍처 컨설팅/자침, 배포 문제의 완전한 제거 등 다양한 이점이 있습니다.

가장 큰 단점은 독점 도구에 대한 공급업체 잠금입니다.

대부분의 데이터 팀은 수십 또는 수백 개의 데이터 파이프라인을 보유하고 있기 때문에 장기적으로 엄청난 비용이 소요될 수 있습니다. 평판, 기능 및 수명에 대해 유료 독점 솔루션을 철저히 조사하는 것이 좋습니다.

플랫폼 데이

터 플랫폼은 **Databricks Workflows** 와 같은 플랫폼의 다른 서비스와 긴밀하게 결합된 자체 오케스트레이션 솔루션을 제공합니다. 이러한 오케스트레이터는 특정 플랫폼에 읊인하고 다음을 제공하는 팀에 이상적입니다.

향상된 관찰 가능성 플랫폼에

직접 통합된 이러한 솔루션은 파이프라인 실행의 세부적인 세부 정보에 대한 최대 수준의 가시성을 제공합니다.

최대 호환성 앞서 설명한 것처럼

럼 오케스트레이터는 외부 서비스와 인터페이스하기 위해 존재합니다. 플랫폼 내장 옵션은 반드시 해당 플랫폼의 다른 도구와 호환됩니다.

지원 증가 다른 유료 서

비스와 마찬가지로 플랫폼 도구는 팀의 성공을 돋는 유료 지원 및 솔루션 엔지니어를 제공합니다. 이 서비스는 무시해서는 안 되며 구현 세부 사항에 대한 의사 컨설팅이 필요한 새로운 데이터 팀에게 매우 중요할 수 있습니다.

플랫폼별 오케스트레이터는 공급업체 잠금을 높이고 비용을 지불하지만 솔루션을 유지 관리하고 확장하는 데 필요한 작업을 대폭 줄일 수 있습니다. 그 가치를 무시해서는 안 됩니다. 이미 Databricks와 같은 플랫폼을 사용하고 있다면 해당 솔루션이 상당히 합리적입니다.

SQL 오케스트레이션 지

금까지 전체 데이터 시스템을 오케스트레이션하는 데 중점을 두었지만 자체적인 오케스트레이션 개념을 갖는 데이터 변환의 하위 집합인 관계형 데이터베이스가 있습니다. 데이터는 소스 시스템에 기록된 후 복잡한 SQL 쿼리 및 조작을 통해 순차적으로 변환되는 경우가 많습니다.

최신 데이터 스택 이전에는 팀이 맞춤형 인프라에 의존하여 이러한 변환을 단독으로 관리했습니다. 현재 우리는 dbt, Delta Live Tables, Dataform 또는 SQLMesh와 같은 도구를 사용합니다. 이러한 도구는 종속성과 조건을 평가한 다음 데이터베이스에 대해 명령을 최적화하고 실행하여 원하는 결과를 생성한다는 점에서 조정자입니다.

이러한 종복은 데이터 조정의 맥락에서 중요해집니다.
예를 들어 다음과 같은 Airflow DAG를 생각해 보세요.

1. 데이터를 단계로 수집합니다.
2. 데이터를 가볍게 변환합니다.
3. dbt 프로젝트를 트리거합니다.

흥미로운 점은 #3이 실제로 데이터 웨어하우스에 대해 실행되는 DAG라는 것입니다 ([그림 3-2](#)).

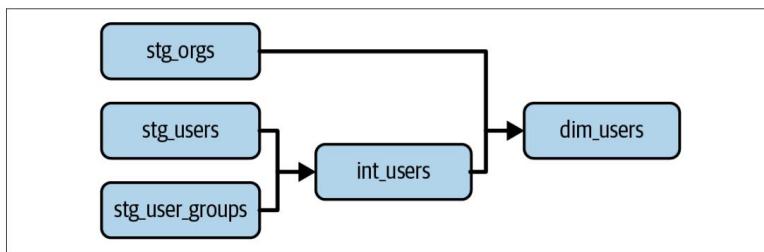


그림 3-2. dbt 파이프라인—익숙해 보이죠?

그러나 이 구조는 잠재적인 한계를 암시합니다. SQL 오케스트레이터 외부와 내부에 또 다른 계보가 존재하므로 수집에서 변환까지 이러한 계층 전체에서 데이터를 관찰하기 위한 메커니즘이 필요합니다.

이는 많은 데이터 팀이 직면하는 일반적인 함정으로, 종종 소스와 정리된 데이터 간의 연결이 끊어지는 결과를 낳습니다. 다운스트림 데이터에서 오류를 식별하는 것은 엄청난 과제가 됩니다. 분석가 또는 데이터 엔지니어에 의해 변칙이 도입되었는지 식별하는 것은 건초 더미에서 바늘을 찾는 것과 같습니다.

계보는 오페스트레이션 프로세스를 간소화할 뿐만 아니라 데이터 관찰 가능성에 대한 보다 포괄적인 관점을 조성하여 다양한 데이터 계층 간의 격차를 해소하고 보다 원활한 오류 감지 및 해결 프로세스를 보장합니다. 이것이 플랫폼별 데이터 오페스트레이터를 선택하는 한 가지 이유입니다. 데이터 워크플로우 간 및 데이터 워크플로우 내에서 가시성이 향상됩니다.

관측 가능성에 대해서는 4 장에서 더 자세히 논의하겠지만 지금은 SQL 기반 변환 프레임워크가 주로 오페스트레이터이며 앤드투엔드 관측 가능성을 제공하는 오페스트레이터를 찾는 것이 ETL 워크플로우에 매우 중요할 수 있다는 점만 언급하겠습니다.

디자인 패턴 및 모범 사례

디자인 패턴을 현명하게 사용하면 현대 데이터 엔지니어링 내에서 데이터 조정 프로세스의 효율성, 신뢰성 및 유지 관리 가능성을 크게 향상시킬 수 있습니다.

환경.

이러한 패턴은 "오페스트레이터 선택"에서 언급한 고려 사항 외부에 존재합니다. 대부분의 경우 오페스트레이션 선택과 관계없이 수행할 수 있기 때문입니다. 그러나 일부 오페스트레이션 솔루션은 이러한 패턴을 훨씬 더 쉽게 만들어준다는 점에 유의해야 합니다. 이는 선택 시 중요한 고려 사항입니다.

백필 데이

터 시스템을 구축하는 경우 구현 이전에도 데이터가 존재했을 가능성이 높습니다. 이는 새 데이터 기록을 시작하기 전에 이전 데이터를 다시 채워야 함을 의미합니다. 많은 사람들이 "일회성 사고"에 빠져 복제할 수 없는 해킹된 솔루션을 만들 것입니다.

가장 좋은 방법은 한 번 수행된 작업은 반복 가능해야 한다는 것입니다. 오페스트레이션에서는 파일프라인 자체에 백필 논리를 구축할 수 있는 좋은 기회가 있습니다. 가장 좋은 방법은 "이 데이터가 내일 사라지면 다시 만들 수 있을까요?"라고 묻는 것입니다.

정확한 시나리오가 발생하지 않을 수도 있지만 앞으로 열을 추가하거나 몇 가지 추가 날짜를 선택해야 할 가능성이 높습니다.

Airflow와 같은 도구에서는 과거의 `start_date`를 간단히 설정하여 기록 데이터를 다시 생성하는 실행을 트리거할 수 있는 DAG를 구축하는 것이 좋습니다. 이것은 때로는 실행보다 밀하기가 더 쉬우며 그렇게 될 것입니다.

멱등성 파이프라인이 필요합니다.

멱등성 우리는 2

장에서 멱등성을 논의했지만 다시 강조할 가치가 있습니다. 멱등성, 즉 어떤 작업을 여러 번 수행하여 일관된 결과를 얻을 수 있도록 보장하는 것은 신뢰할 수 있는 데이터 엔지니어링에 필수적입니다.

백필을 실행할 때 특히 중요합니다. 실수로 날짜가 겹치는 경우 데이터가 중복됩니까? 미래 날짜 외에 과거 날짜에 대해서도 파이프라인을 실행할 수 있나요?

이벤트 기반 오케스트레이션

이벤트 기반 오케스트레이션을 통해 데이터 또는 시스템 상태 변경에 대한 반응성을 높일 수 있습니다. 이벤트를 트리거한다는 것은 데이터가 가능한 한 최신 상태로 유지되거나 적절한 시간에 발생한다는 것을 의미합니다.

예를 들어 데이터 웨어하우스가 Fivetran 수집 작업에 크게 의존하는 경우 작업이 완료되면 이를 트리거할 수 있습니다.

이렇게 하면 이상한 cron 구문이나 장기 실행 작업이 중복되어 문제를 일으킬 가능성을 피할 수 있습니다.

이벤트 기반 오케스트레이션을 사용하여 비용을 낮출 수도 있습니다. 소스 데이터가 자주 발생하지 않는 부담스러운 파이프라인이 있습니까? 소스가 업데이트될 때만 해당 파이프라인을 트리거하도록 시스템을 구축할 수 있습니다.

조건부 논리 특히 데이터

와 ETL의 흥미로운 점은 입력이 항상 변경된다는 것입니다. 문제는 “깨질까요?”가 아닙니다. 하지만 “언제, 얼마나 자주 깨지나요?” 따라서 오케스트레이터는 조건부 논리("True이면 X, False이면 Y")를 처리하여 데이터 워크플로를 지시할 수 있어야 합니다.

조건부 논리는 팀이 ETL 프로세스를 더욱 자동화할 수 있음을 의미합니다. 파이프라인에서 일반적인 예외가 발생하는 경우 조건부 논리를 추가하면 오류, 실패 또는 잘못된 데이터를 설명할 수 있으므로 팀이 예외 처리 지역에서 벗어날 수 있습니다.

동시성 이 책의 존

재를 고려하면 우리는 ETL이 어렵다(또는 적어도 그리 쉽지는 않음)는 것을 알고 있습니다.

하나 이상의 파이프라인이 대량의 소규모 파이프라인으로 인해 병목 현상이 발생할 가능성 이 높습니다.

개별 작업 예를 들어 수천 개의 작은 압축 CSV를 수집해야 할 수도 있습니다. 특히 잘 못된 실행 체계를 사용하면 각각을 반복하는 데 몇 시간이 걸립니다.

대신 동적 작업이라고도 하는 동시성을 사용하여 시간을 절약할 수 있습니다. 오케스트레이터는 분명히 동시 파이프라인을 실행할 수 있지만 대부분은 동시에 실행될 작업을 펜아웃할 수 있습니다. 그림 3-3은 Apache Airflow에서 실행되는 동시 동적 파이프라인의 예를 보여줍니다.

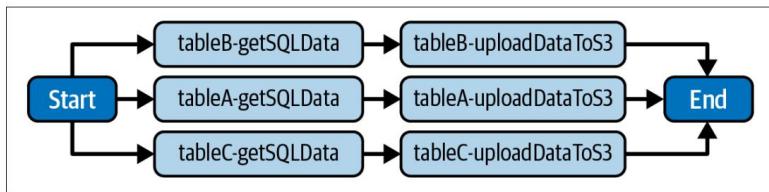


그림 3-3. 동적 작업 매핑은 Apache Airflow에서 병렬로 실행되는 작업을 프로그래밍 방식으로 생성하는 한 가지 방법입니다.

빠른 피드백 루프 오케스

트레이터는 본질적으로 복잡하며 종종 테스트하기 어려운 작업을 트리거합니다. 이로 인해 지역적으로 개발하기가 매우 어렵습니다. 우리는 "로컬 Airflow 인스턴스를 중지하는 가장 쉬운 방법은 내 컴퓨터를 재부팅하는 것"이라는 말을 들었습니다. 이 말은 과장이긴 하지만 일리가 있는 말입니다.

빠르게 실패하고 오류를 빠르게 식별하고, 클라우드 배포 인스턴스와 패리티를 달성하고, 개발자 친화적이고 빠르게 가동할 수 있는 로컬 환경을 갖추는데 도움이 되는 도구를 찾는 데 상당한 시간과 에너지를 투자하는 것이 좋습니다.

컨테이너화 덕분에 인프라를 쉽게 배포할 수 있으므로 유지 관리 및 확장이 엄청나게 번거로운 것을 배포하는 것도 가능합니다. 만약 당신이 이 함정에 빠지게 된다면, 그것은 매우 고통스러운 경험이 될 수 있습니다.

재시도 및 대체 논리 파이프라

인 차우기를 준비해야 하는 것과 마찬가지로 재시도 및 실패는 불가피합니다. 복잡한 데이터 스택에서 오류를 잘 처리하면 데이터 무결성과 시스템 안정성이 보장되어 결과적으로 원활한 데이터 운영이 촉진되고 가동 중지 시간이 줄어듭니다.

멱등성 파이프라인의 일부는 데이터를 생략하거나 복제하지 않고 작업을 다시 시도하거나 건너뛰고 적절한 당사자에게 경고하는 시나리오를 설정하는 방식으로 오류를 처리하는 것입니다. 대부분의 오케스트레이터에서는 작업 및 파이프라인 수준 모두에서 "재시도" 매개변수를 직접 설정할 수 있습니다.

조건부 논리를 재시도/대체 논리와 결합하여 오류를 적절하게 처리하는 시나리오를 생성함으로써 오류를 분류하는 데 소요되는 시간과 데이터 팀 전체의 스트레스 수준을 낮출 수 있습니다.

매개변수화된 실행

재시도 및 조건부 논리와 함께 매개변수화된 실행을 통해 오케스트레이션의 유연성이 향상됩니다. 매개변수를 추가한다는 것은 단순히 오케스트레이터가 변수를 허용하도록 허용하는 것을 의미합니다.

이는 다양한 사례를 처리할 뿐만 아니라 여러 목적으로 파이프라인을 재사용하는 귀중한 방법이 될 수 있습니다. 예를 들어, 매개변수화된 실행은 백필을 촉진하는 데 매우 중요할 수 있습니다.

예를 들어 API에서 데이터를 가져오기 위한 매개변수로 날짜를 허용하는 DAG를 설계하면 백필 할 날짜 목록을 순차적으로 실행하여 파이프라인을 백필하는 간단한 구조를 만들 수 있습니다.

리니지

(Lineage) 리니지는 데이터의 수명주기 동안 데이터가 이동하는 경로를 나타냅니다. DAG의 시각적 특성을 고려할 때 데이터에 대해 수행되는 다양한 작업을 이해하기에 좋은 장소이기도 합니다. 계보는 문제를 디버깅하고 파이프라인을 확장하는데 중요한 역할을 하므로 계보 솔루션이 강력한지 확인하세요. 이상적으로는 열 수준 계보, 즉 수집부터 분석까지 데이터 이동 과정을 열 수준에서 볼 수 있도록 노력하는 것이 좋습니다.

열 수준 계보는 변환 파이프라인을 통해 데이터 경로를 밝히고 추적 가능성과 디버깅 능력을 증폭시킵니다. 이러한 세분성은 SQL 오케스트레이션에서 업계 표준이 될 준비가 되어 있습니다. 나는 새로운 구현에서 열 수준 계보가 테이블 스테이크가 되어야 한다고 주장합니다. 고급 계보 기능은 Databricks Unity Catalog 및 Delta Live Tables와 같은 플랫폼 통합 오케스트레이션 솔루션의 확실한 이점 중 하나입니다.

파이프라인 분해 깔끔하

고 읽기 쉬운 파이프라인을 보장하기 위한 좋은 전략은 더 나은 모니터링, 오류 처리 및 확장성을 촉진하는 더 작고 관리하기 쉬운 작업으로 파이프라인을 나누는 것입니다 ([그림 3-4](#)).

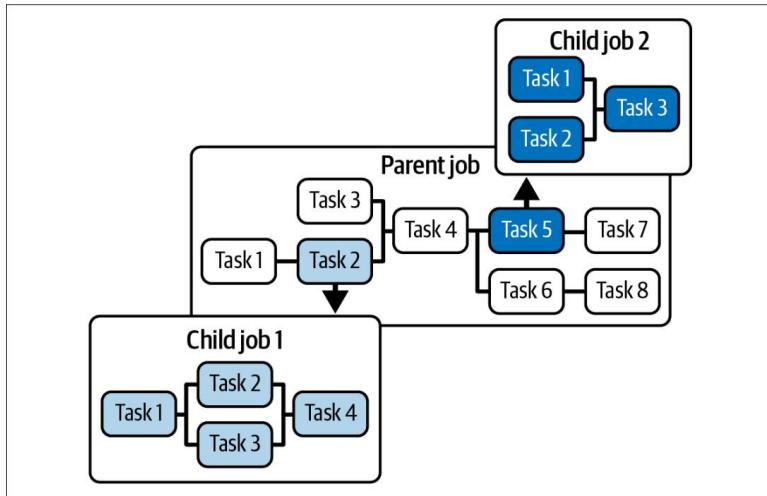


그림 3-4. 파이프라인 분해의 예 - 파이프라인을 마이크로서비스로 나누고 메타-DAG로 조정

소프트웨어 엔지니어링에서 차용하여 파이프라인을 마이크로서비스로 생각합니다. 마이크로서비스의 핵심 기능은 자율성과 서로 독립적으로 작동하는 능력입니다. 종속성과 심각한 오류를 완화하기 위해 병렬로 실행할 수 있는 자율 DAG를 구축하는 것을 목표로 합니다. [모듈형 디자인](#) 또한 워크플로를 더 쉽게 구축하고 디버깅할 수 있습니다.

데이터 오케스트레이션의 미래

이 가이드가 왜 최신 데이터 스택의 역사와 미래에 그토록 중점을 두고 있는지 궁금하십니까? 빠르게 발전하는 산업에서는 트렌드를 파악하는 것이 중요합니다. 이는 몇 년 안에 쓸모 없게 될 도구를 채택하는 것과 더 많은 이점을 얻을 수 있는 도구를 채택하는 것의 차이일 수 있습니다. 견인.

시간과 리소스 절약 측면에서 측정된 올바른 도구는 잘 이끄는 데이터 팀을 다른 팀보다 훨씬 앞서게 할 수 있습니다.

이 장에서는 오케스트레이션과 변환 도구를 병합하는 것에 대해 의구심을 표명했지만, 2 장에서 논의한 변환 도구의 발전으로 인해 이야기가 바뀔 수도 있습니다.

복잡한 빅 데이터 도구(예: MotherDuck, PyArrow, 최신 오케스트레이터)를 제거하기 위한 새로운 유형의 도구와 함께 수직 및 수평 자동 확장을 모두 촉진하는 컨테이너화된 인프라는 강력한 혁신 솔루션에 대한 수요를 줄일 수 있습니다. Prefect 및 Dagster와 같은 하이브리드 변환 도구는 진지한 고려가 필요합니다.

SQL 오케스트레이션의 발전은 데이터 웨어하우스(또는 레이크하우스) 내에서 관찰 가능성과 모니터링이 향상될 것을 예고합니다. SQLMesh와 같은 신흥 도구는 dbt와 같은 기존 플레이어에 도전할 준비가 되어 있지만 dbt는 계속해서 플랫폼을 혁신하고 정기적으로 업데이트하고 있습니다.

플랫폼이 성숙해짐에 따라 Databricks Workflows와 같은 솔루션은 플러그 앤 플레이 관점에서 점점 더 매력적으로 변하고 있습니다.

이러한 솔루션이 해당 플랫폼에 기본적으로 제공된다는 사실은 기능과 원활한 통합에 대한 자신감을 심어줍니다.

급변하는 환경에서 최신 정보를 유지하는 것은 매우 힘든 일이지만, 이것이 바로 제가 이와 같은 가이드를 작성하는 직업을 갖게 된 이유입니다.

그럼에도 불구하고, 이러한 발전은 우리가 업무를 보다 효과적으로 수행하고 우리가 원하는 것, 즉 적시에 강력한 방식으로 고품질 데이터를 제공할 수 있게 해주기 때문에 흥미로울 것입니다.

제4장

파이프라인 문제 및 문제 해결

앞서 언급했듯이 ETL의 기본 목표는 비즈니스 가치를 제공하는 것입니다. 그러나 이러한 목표는 복구 시간이 길어지는 취약한 시스템으로 인해 방해를 받는 경우가 많으며, 이로 인해 혁신과 가치 창출보다는 문제 해결에 과도한 리소스 할당이 발생합니다.

따라서 잘 설계된 ETL 파이프라인은 강력하고 유지 관리가 가능해야 하며 높은 데이터 품질, 효율적인 오류 처리, 효과적인 문제 식별과 같은 주요 특성을 구현해야 합니다. 이는 모두 시스템 관찰 가능성의 필수 구성 요소입니다. 귀하의 엔지니어링 능력에 관계없이 귀하의 시스템은 실패할 것입니다. 모든 것이 우리의 통제 안에 있는 것은 아닙니다!

실패에 대한 계획은 다음과 같은 엔지니어링 파이프라인을 의미합니다.

- 유지 관리 및 확장이 용이합니다. 즉, 빠른 오류가 허용됩니다.
분류 및 새로운 기능 개발
- 실시간 오류 처리 및 장애 복구를 위한 자동화된 방법 제공
- 학습과 경험을 바탕으로 개선을 위한 프레임워크를 통합합니다.

이 접근 방식은 유지 관리성을 보장하고 ETL 시스템의 또 다른 중요한 측면인 확장성과 원활하게 연결됩니다.

더 깊이 탐구하면서 시스템 유지 관리가 시스템 확장과 어떻게 밀접하게 연관되어 ETL 파이프라인이 변화하는 비즈니스 요구에 탄력적이고 적응할 수 있는지 확인하기 위한 과제와 전략을 해결하는 방법을 살펴보겠습니다.

유지 관리성

문제 및 문제 해결에 관한 장에서는 유지 관리 가능성, 즉 문자 그대로 이미 구축한 항목을 유지 관리하는 능력이라는 주제를 주로 다루고 있습니다.

엔지니어로서 우리는 때때로 솔루션에 대해서만 목표를 추구합니다.

우리는 최소 실행 가능 제품과 80/20 원칙을 크게 옹호하지만, 비즈니스 가치를 창출하기 위해 시스템에 대해 협력할 때는 해당 시스템을 유지 관리하고 확장하는 능력(5 장의 주제)이 필수적입니다. 파이프라인을 유지 관리할 수 없으면 팀에 직간접적인 비용이 발생합니다. 전자가 분명히 더 눈에 띄는 반면, 후자가 전함을 침몰시킬 가능성이 더 높습니다.

직접비

대부분은 직접 비용에 대해 잘 알고 있습니다. 값비싼 제품을 원한다면 일반적으로 비용을 지불하고 그에 대한 좋은 사례를 만들어야 합니다. 이자율이 거의 0에 가까웠고 리오 그란데처럼 벤처 캐피탈 자금이 흘러갔을 때 많은 사람들이 클라우드 컴퓨팅, 스토리지, DataOps와 같은 비용을 간과했습니다. 효율성은 패턴 디자인에서 시작됩니다. 나쁜 패턴은 조직 전체의 효율성이 낮다는 것을 의미합니다.

유지 관리 용이성이 낮은 ETL 시스템은 비효율적인 운영과 긴 런타임의 형태로 높은 직접 비용을 발생시킵니다. 이로 인해 선택한 공급자로부터 높은 비용이 청구될 뿐만 아니라 작업 속도가 느려지고 팀의 데이터가 지연될 수도 있습니다.

간접비

간접 비용은 직접 비용보다 훨씬 클 수 있습니다. 자주 실패하고 지속적인 심사가 필요한 시스템을 만드는 것은 리소스 할당에 있어서 암울할 수 있습니다. 팀의 시간과 에너지는 당신이 가진 가장 귀중한 자산입니다.

DAG를 수정하고, 경고에 응답하고, 화재를 진압하는 데 대부분의 근무 시간을 소비하면 바쁜 것처럼 보일 수 있지만 일반적으로 지속 불가능하고 비생산적인 것으로 나타납니다. 승리하는 팀은 이를 가능하게 하는 효율적인 시스템을 구축합니다.

기능 개발과 데이터 민주화에 집중합니다.

비효율적인 시스템은 더 많은 소방 훈련, 더 많은 잠 못 이루는 밤, 중요한 것을 구축하는 데 소요되는 시간 감소를 의미합니다.

SaaS의 주요 판매 포인트는 비용, 즉 사내 솔루션을 개발하고 유지 관리하는 비용보다 이점이 더 크다는 것입니다. 리소스나 경험이 제한된 팀에 속해 있다면 유지 관리가 가장 쉬운 솔루션은 구입하는 솔루션일 수 있습니다.

물론 이러한 중단은 데이터 엔지니어에게만 영향을 미치는 것이 아니라 다운스트림에도 영향을 미칩니다. 이는 이해관계자의 신뢰 상실, 고객의 수익 상실, 심지어 일반 대중의 평판 상실을 의미합니다. 데이터 오류, 특히 PII가 노출된 데이터 오류로 인한 간접적인 비용은 치명적일 수 있습니다.

그렇다면 유지보수성에 중점을 두는 이유는 무엇입니까? 승리하는 팀은 유지 관리 가능한 시스템을 기반으로 구축됩니다.

적시에 효율적인 데이터를 사용하여 조직을 발전시키는 것이 목표라면 문제와 문제 해결을 최소화하고 직간접적인 운영 비용을 줄여야 합니다. 이 장의 나머지 부분에서는 관찰 가능성, 데이터 품질, 오류 처리 및 향상된 워크플로를 통해 이를 수행할 수 있는 방법을 정확하게 논의하겠습니다.

모니터링 및 벤치마킹

여기서는 관찰 가능성과 모니터링/벤치마킹을 구별합니다. 벤치마킹 및 모니터링 시스템은 필수적입니다. 이는 관찰 가능성의 하위 집합입니다. 그러나 관찰 가능성은 문제 해결 및 유지 관리에만 국한되지 않습니다. 파이프라인 문제를 최소화하고 문제 해결 노력을 가속화하려면 시스템을 모니터링하고 벤치마킹해야 합니다.

엄격하게 모니터링되고 깔끔하게 벤치마킹된 시스템은 "관찰 가능"하도록 매우 훌륭하게 설정되어 있으며 데이터 시스템의 유지 관리성을 더 쉽게 개선하고 손상된 데이터와 관련된 비용을 낮출 수 있습니다. 적절한 모니터링과 경고가 없으면 팀은 문제가 발생했다는 사실조차 알지 못할 수도 있습니다. 우리가 원하는 것은 이해관계자가 데이터 오류를 발견하는 사람이 되는 것입니다. 더 나쁜 것은 오류가 발견되지 않고 잘못된 비즈니스 결정이 초래되는 것입니다.

우리는 수집, 변환, 저장 전반에 걸쳐 데이터를 관찰하고, 오류가 발생할 때마다 처리하고(가능하다면 적절하게) 문제가 발생하면 팀에 알려야 합니다.

그러나 관찰 가능성은 문제 해결만을 위한 것이 아닙니다. 또한 5 장에서 논의 하겠지만 확장에도 도움이 됩니다.

측정항목

다음은 조직 내 데이터의 신뢰성과 유용성을 평가하는 데 사용되는 몇 가지 필수 측정 방법입니다. 이러한 지표는 수집 및 처리되는 데이터가 특정 표준을 충족하고 의도된 목적을 효과적으로 제공하는지 확인하는 데 도움이 됩니다.

신선도 신선도

는 시스템 내 데이터의 적시성과 관련성을 나타냅니다. 이는 데이터가 나타내는 실제 이벤트와 비교하여 데이터가 얼마나 최신 상태인지를 측정하는 것입니다. 데이터 최신성을 유지하는 것은 분석, 의사결정 및 기타 데이터 기반 프로세스가 정확하고 최신 정보를 기반으로 하도록 보장하는데 중요합니다. 데이터 엔지니어는 데이터 업데이트의 대기 시간을 최소화하는 시스템을 설계하고 구현하여 이해관계자가 분석 및 운영을 위해 최신 데이터에 액세스할 수 있도록 노력합니다.

데이터세트의 일반적인 최신성 측정항목은 다음과 같습니다.

- 가장 최근 타임스탬프 사이의 길이(데이터) 및 현재 타임스탬프
- 소스 데이터와 데이터 세트 간의 자연 • 새로 고침 빈도(예: 분, 시간, 일)
- 대기 시간(데이터가 수집된 시점부터 사용 가능한 시점까지의 총 시간)

볼륨 볼륨

은 시스템 내에서 처리, 저장 및 관리해야 하는 데이터의 양을 의미합니다. 이는 데이터 처리의 기본 측면입니다. 대규모 데이터를 처리하려면 효율적인 저장, 빠른 검색, 처리 속도 등의 문제가 발생합니다. 높은 데이터 볼륨에는 분산 컴퓨팅, 병렬 처리, 데이터 압축과 같은 특수 인프라와 기술이 필요합니다.

블룸 측정항목에는 다음이 포함됩니다.

- 데이터 레이크의 크기(기가바이트, 테라바이트, 페타바이트)
- 데이터베이스의 행 수
- 시스템의 일일 거래량

품질 품질

에는 데이터의 수명주기 전반에 걸쳐 데이터가 정확하고 일관되며 신뢰할 수 있음을 보장하는 것이 포함됩니다. 데이터 품질은 정확성, 일관성, 신뢰성, 적시성, 완전성, 보안, 문서화 및 모니터링을 중심으로 이루어집니다. 이러한 측면을 해결하면 정보에 입각한 의사 결정 및 분석을 위한 고품질 데이터가 보장됩니다.

다음은 몇 가지 샘플 데이터 품질 지표입니다.

- 고유성: 데이터세트에 중복된 행이 있습니까? • 완전성: Null이 몇 개나 존재합니까? 예상되는가?
- 유효성: 데이터 형식이 일관되게 지정되어 있습니까? 적절한 범위(예: 0보다 큰 범위)에 존재합니까?

행동 양식

모니터링은 스택 전체에서 발생하는 모든 것을 확인하고 적시에 오류를 감지하는 능력을 의미합니다. 데이터 품질은 관찰된 데이터의 품질을 향상시키는 엄격한 조치를 구현하는 것을 의미합니다.

다음은 모니터링 중인 데이터의 품질을 직접적으로 개선하여 오류를 줄이고 가동 중지 시간을 줄이기 위해 적용할 수 있는 패턴과 기술입니다.

로깅 및 모니터링 디버깅의 첫 번

째 단계는 로그를 확인하는 것입니다. 하지만 로그를 확인하려면 확인할 로그가 있어야 합니다! 시스템이 데이터를 자세하게 기록하고 있는지 확인하십시오. 구축하는 시스템의 경우 로깅을 필수로 설정하세요. 라이브러리 및 기록할 항목을 포함하여 로깅 모범 사례를 정의하고 성문화합니다. 당신은 뛰어난 데이터 배관공이어야 할 뿐만 아니라 우리가 훨씬 더 좋게 생각하는 용어인 데이터 나무꾼이어야 합니다.

계보 개념

직으로는 단순한 계보(라이프사이클 동안 데이터가 이동하는 경로)는 데이터를 관찰하는 가장 중요한 방법 중 하나입니다. 파이프라인과 시스템의 시각적 및 코드 표현을 갖는 것은 일상적인 실행에 중요하며 문제 분류 및 디버깅에 소요되는 시간을 무한히 절약할 수 있습니다.

계보가 유용하려면 완전하고 세부적이어야 합니다. 시스템 간의 상호 연결성을 포함하여 모든 시스템이 관찰된다는 점에서 완전합니다. 이상적으로는 귀하의 계보가 가장 세부적인 수준에 있을 것입니다. 테이블 형식 데이터의 경우 이는 열 수준입니다. 열 수준 메타데이터는 가장 세부적인 통찰력을 제공합니다. 이는 팀의 오류 분류 능력을 향상시키고, 워크플로를 단순화하며, 생산성과 데이터 작업 경험을 향상시킵니다. 그림 4-1에서 볼 수 있듯이 계보 연습은 열부터 시작됩니다.

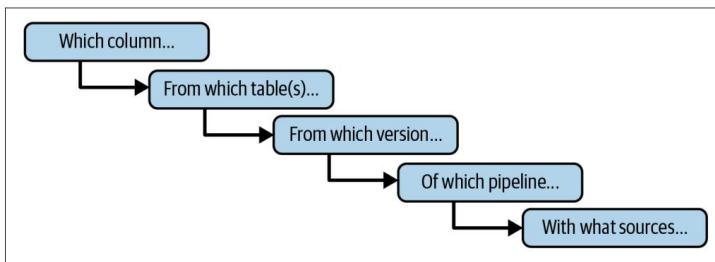


그림 4-1. 데이터 출처에 대한 첫 번째 원칙 평가를 통해 계보의 복잡성이 드러납니다.

좋은 시작은 독립적인 계보 솔루션을 갖춘 시스템을 구현하는 것입니다. 보다 완전한 계보 시스템은 모든 데이터 프로세스를 관찰할 수 있습니다. 관리형 플랫폼은 일반적으로 독립형이므로 이에 적합합니다. Monte Carlo와 같은 다른 도구는 전체 스택에 대한 통찰력을 제공할 수 있습니다.

이상 탐지 데이터에서

가장 까다로운 점은 모든 것을 볼 수 없다는 것입니다. 대부분의 오류는 이해관계자나 분석가가 예상치 못한 결과를 반환함으로써 발생합니다. 이는 "오늘 내 데이터에 어떤 그램린이 살아있나요?"라는 질문을 불러일으킵니다.

변칙 검색은 특정 임계값 내에서 변칙 데이터가 존재하지 않음을 알 수 있는 한 가지 방법입니다. 이상 탐지 시스템은 기본 통계 예측을 통해 시계열 데이터를 분석하고 일부 신뢰 구간을 벗어난 데이터를 반환합니다.

이상 탐지는 시스템 외부에서 발생할 수 있는 오류를 포착하는 좋은 방법이 될 수 있습니다. 예를 들어 유럽에서 구매를 과소보고하는 버그를 도입하는 결제 처리 팀이 있습니다. 경보는 울리지 않지만 오른쪽 테이블에 있는 이상 탐지 시스템이 버그를 잡아냅니다.

데이터 차이

점 데이터 엔지니어링에는 소프트웨어 엔지니어링에 포함되지 않은 추가 차원인 데이터 품질이 포함됩니다. 코드를 변경하면 이해하기 어려운 방식으로 출력이 변경됩니다. 데이터에서 가장 어려운 점은 무엇입니까?

데이터 차이점은 코드 변경으로 인한 데이터 변경 사항을 보고하는 시스템입니다. 이러한 도구는 행 및 열 변경 사항, 기본 키 수 및 보다 구체적인 효과를 알려줍니다. 이들의 주요 목적은 정확한 시스템이 정확하게 유지되도록 하는 것입니다.

데이터 비교 솔루션의 경우 Datafold와 같은 도구를 권장합니다.

SQLMesh와 같은 최신 SQL 조정자에도 데이터 비교 기능이 있습니다. 데이터 비교는 CI/CD와 밀접하게 연결되어 있으며 주장과 테스트를 통해 잘 강조됩니다. 위의 내용을 모두 곧 논의하겠습니다.

어설션 어설션은

소스 데이터의 유효성을 검사하기 위해 데이터 출력에 적용되는 제약 조건입니다. 이상 탐지와 달리 어설션은 훨씬 간단합니다. 예를 들어, "users.purchasesplans.pricing 테이블의 가격만 포함하는지 확인"이라고 말할 수 있습니다. 가격에 존재하지 않는 값이 구매 항목에 표시되면 (a) 새로운 가격이 도입되었거나 (b) 시스템에 오류가 있음을 알 수 있습니다.

본질적으로 수동적이지만(어설션에는 일부 비즈니스 컨텍스트와 무엇이 되어야 하는지에 대한 이해가 필요함), 어설션은 (적어도 우리가 지정하는 방식으로) 데이터에 오류가 없다고 완전히 확신할 수 있는 몇 가지 방법 중 하나입니다. 어설션 솔루션의 경우 Great Expectations와 같은 라이브러리를 확인하거나 어설션을 정의하는 기능이 내장된 시스템을 찾으세요.

오류

이제 관찰과 오류 예방에 대해 논의했으므로 간단한 진실에 이르렀습니다. 솔루션이 얼마나 강력하더라도 여전히 오류가 있을 수 있습니다! 따라서 오류 처리는 귀하와 팀 모두에게 매우 중요합니다.

오류 처리와 별개로 데이터 손실이든 가동 중지 시간이든 그 영향으로부터 복구하는 것입니다. 복구의 중요한 부분은 회고, 사후 분석, 향후 유사한 오류를 방지하는 방법에 대한 성찰입니다.

오류 처리 오류 처리

는 데이터 시스템의 기능을 유지하거나 적시에 신중하게 팀에 경고하기 위해 오류 응답이나 경계 조건을 자동화하는 방법입니다. 다음 접근 방식에서는 오류를 적절하고 효율적으로 처리하는 몇 가지 방법을 자세히 설명합니다.

조건부 논리 데이

터 파이프라인을 구축할 때 조건부 논리는 신뢰할 수 없거나 일관성이 없는 소스에 유용할 수 있습니다. "X라면 A, 그렇지 않으면 Y이면 B"라고 말할 수 있는 기능은 오케스트레이션 및 변환 도구에 강력한 구성 요소를 추가합니다. 조건부 논리를 하용하는 솔루션을 찾으세요.

재시도 메커니즘 시스템

은 잘 구축되었더라도 실패합니다. 예상치 못한 오류로 인해 일회성 API 시간 초과나 기타 이상한 현상이 발생할 수 있습니다. 따라서 제대로 작동하는 코드라도 오류가 발생할 수 있습니다. 재시도 논리는 모든 오케스트레이션 도구에서 중요합니다. 물론 이러한 기능이 작동하려면 합리적인 재시도 설정을 구성해야 합니다. 데이터의 이상한 점이나 미묘한 차이를 처리할 수 있지만 낭비가 아니며 끝없는 실행을 초래하지 않는 재시도 전략을 통합해야 합니다.

파이프라인 분해 이전에

3 장에서 모듈화 개념을 언급했지만 파이프라인을 "마이크로서비스"로 나누는 것은 오류의 영향을 억제하는 효과적인 방법입니다. 꼭 필요한 테이블과 연결만 필요한 DAG 및 시스템 구축을 고려해보세요.

정상적인 성능 저하 및 오류 격리 오류 격리는

파이프라인 분해를 통해 가능합니다. 가능하면 시스템은 억제된 방식으로 오류가 발생하도록 설계해야 합니다. 예를 들어 제품 사용 데이터가 재무 보고 파이프라인의 업스트림에 게시되는 것을 원하지 않을 것입니다. 그렇지 않으면 관련 없는 오류로 인해 CFO가 늦은 메트릭을 보드에 반환하게 될 수 있습니다.

아무도.

정상적인 성능 저하란 시스템의 일부에 장애가 발생하더라도 제한된 기능을 유지하는 능력입니다. 오류를 격리하고 파이프라인을 분해함으로써 우리는 효과적으로 정상적인 성능 저하를 가능하게 합니다. 시스템의 나머지 부분이 너무 잘 작동하기 때문에 비즈니스의 한 부분에서만 인지되는 오류가 발생할 수 있습니다. 우리를 믿으십시오. 이것은 모두가 알아차리는 오류보다 훨씬 낫습니다.

경고 경고

는 최후의 방어선이 되어야 합니다. 즉, 경고 수신은 반드시 사후 조치를 취해야 합니다. 우리는 뭔가 나쁜 일이 일어났다는 것을 알고 그것을 고치기 위해 모든 것을 버립니다. 일반적인 오류를 사전에 해결하는 것이 가장 좋지만 결국 예상치 못한 오류가 발생하기 마련입니다.

알림은 이메일이나 Slack 메시지의 형태로 제공될 수 있습니다. Slack은 눈에 잘 띄고 팀 구성원이 유용한 맥락이 포함된 댓글을 추가할 수 있기 때문에 Slack을 선호합니다.

주의를 기울일 때 주의 피로도를 인지하세요. 경고 피로는 향후 경고의 중요성을 감소시키는 압도적인 수의 알림을 의미합니다. 사려 깊은 알림과 함께 오류를 격리하고 정상적으로 성능이 저하되는 시스템을 구축하는 것은 경보 피로를 줄이고 팀을 위한 좋은 개발자 경험을 만드는 강력한 메커니즘이 될 수 있습니다.

복구 이제 우리

는 시스템에 대한 통찰력을 갖게 되었고, 품질 제약 조건을 적극적으로 시행하고 있으며, 오류를 처리하기 위한 전용 방법을 갖게 되었습니다. 마지막 부분은 데이터 손실이 포함될 수 있는 재해 복구를 위한 시스템을 구축하는 것입니다. 다음은 피할 수 없는 실패 이후 회복하는 데 도움이 되는 몇 가지 방법과 개념입니다.

스테이징

지금까지 스테이징에 대해 꽤 많이 언급했지만 스테이징된 데이터의 또 다른 이점은 재해 복구입니다. Delta Lake와 같은 Parquet 기반 형식과 메달리온 아키텍처와 같은 패턴을 사용하면 시간 여행을 통해 데이터를 특정 지점까지 복원 할 수 있습니다. 준비된 데이터는 임시 데이터로 처리되어야 하지만 중복성을 위한 중요한 패턴입니다.

백필 (a) 파

아프라인 실행을 시작하기 전 일정 기간의 데이터가 필요하거나 (b) 백필이 필요한 데이터가 손실되었을 가능성이 높습니다. 채우기는 완전한 데이터 세트를 생성하기 위해 파이프라인의 기록 실행을 시뮬레이션하는 방법입니다. 예를 들어 Airflow에는 **백필 명령어** 가 있습니다. 두 날짜 사이의 모든 날짜에 대해 파이프라인을 실행합니다.

시스템을 구축할 때 백필을 염두에 두십시오. 채우기가 쉬운 시스템은 문제가 발생했을 때 상당한 시간을 절약해 줍니다. 채우기 논리는 매우 복잡하고 빠르게 진행될 수 있으므로 간단한 채우기를 지원하는 도구를 찾으십시오.

멱등성 파이프라인은 또한 여러분의 삶을 더 쉽게 만들어줄 것입니다. 기본적으로 백필 기능을 사용하려면 선택한 오케스트레이터를 확인하세요.

워크플로 개선

프로세스 개선이 전적으로 우리의 통제 범위 내에 있기를 바라지만 그렇지 않습니다. 우리의 업무는 본질적으로 협력적입니다. 속담에서 "배관공"으로 이해관계자에게 내부 및 외부 데이터에 대한 파이프라인을 제공합니다. 이는 우리의 업무를 본질적으로 협력적으로 만듭니다.

간략하게 언급했지만 데이터 엔지니어링은 실제로 문제가 발생하는지 여부가 아니라 언제 문제가 발생하는지에 대한 문제입니다. 최고의 시스템에서도 예외가 발생하고 실수가 발생하며 모든 것이 무너집니다. 타이타닉은 이유가 있는 교훈입니다. 그렇죠?

결정적인 요인은 문제에 적응하고 해결하는 능력입니다. 이것이 바로 이 장의 요점입니다! 문제 해결, 적응성 및 복구를 우선시하는 시스템으로 시작하는 것은 골치 아픈 문제를 줄이는 좋은 방법입니다. 이 섹션에서는 프로세스를 지속적으로 개선할 수 있는 몇 가지 방법을 제공합니다.

관계로 시작 다음 예를 고려하

십시오. 반정기적으로 소프트웨어 팀이 경고 없이 프로덕션 스키마를 변경합니다. 이로 인해 프로덕션 데이터세트의 일일 ETL 작업이 중단 될 뿐만 아니라 내보내기 방법에 CDC가 없기 때문에 데이터가 손실 될 위험도 있습니다.

당연히 이로 인해 팀은 상당히 좌절감을 느끼게 됩니다. 이러한 문제를 해결하면 시간과 자원이 낭비될 뿐만 아니라 완전히 피할 수 있는 것처럼 보입니다. 가혹하게 반응하기 전에 "소프트웨어 팀이 전체 생산성을 줄이고 더 나쁜 결과를 초래하려고 하는가?"라는 질문을 자문해 보십시오. 대답이 '예'라면 구직을 권장하지만, 99.999%의 경우 대답은 '아니요, 물론 아닙니다.'입니다.

실제로 해당 팀이 정확히 무엇을 하려고 하는지 스스로에게 묻는다면 아마도 당신도 마찬가지일 것입니다. 주어진 자원에서 가능한 한 최선을 다하는 것입니다. 좋습니다. 이제 그들의 동기에 공감하게 되었습니다.

다음은 그들의 작업 흐름을 이해하고 불만 사항을 전달하는 것입니다. 여기에서 효율성을 향상시키는 프로세스를 만들기 시작할 수 있습니다. 체계적이고 실용적인 접근 방식을 통해 건전한 관계를 보장할 수 있는 몇 가지 방법은 다음과 같습니다.

SLA 서

비스 수준 계약(SLA)은 런타임 및 가동 시간 보장을 위해 공급자 공간에서 일반적이지만 팀 내에서나 팀 간에도 효과적으로 사용할 수 있습니다. 데이터 품질 문제로 어려움을 겪고 있는 경우 모든 사람이 수신 데이터에 필요한 것이 무엇인지 명확하게 이해할 수 있도록 성능 지표, 책임, 응답 및 해결 시간, 에스컬레이션 절차 등을 공식적으로 정의하는 SLA를 고려하세요.

요구 사항을 명확하게 전달하고 소유권을 할당함으로써 SLA는 몇 가지 계약서를 종이에 작성하는 것만으로도 통제할 수 없는 데이터의 품질을 향상시키는 놀랄도록 효과적인 방법이 될 수 있습니다.

데이터 계약

데이터 계약은 외부 소스에서 수집된 데이터를 관리하는 효과적인 방법으로 지난 몇 년 동안 주목을 받았습니다.

dbt에 의해 널리 알려진 계약은 ETL 파이프라인의 일부 또는 전부를 실행하기 전에 메타데이터(일반적으로 열 이름 및 유형)를 확인하는 어설션 유형입니다.

우리는 SLA와 같이 두 당사자 간의 합의를 의미하기 때문에 "계약"이라는 용어를 좋아합니다. 단순한 계약이더라도 먼저 SLA를 정의한 다음 외부 자산에 대한 데이터 계약 형태로 해당 SLA를 구현하는 것이 좋습니다. 계약이 오류를 반환하는 경우(언제) SLA는 해당 오류를 해결하는 책임이 누구에게 있는지, 그리고 오류가 얼마나 빨리 발생해야 하는지를 정확하게 지정합니다.

아피스

API는 계약 및 SLA를 시행하는 보다 공식적인 방법이 될 수 있습니다. 어떤 의미에서 SQL 자체는 API이지만 API를 통해 내부 데이터를 가져오거나 제공하지 못할 이유가 없습니다. API는 예상되는 데이터 세트를 전송하는 방법일 뿐이지만 올바르게 구현되면 소스에 추가적인 표준화 및 일관성 계층을 제공합니다. API에는 규정 준수에 유용할 수 있는 보다 세부적인 액세스 제어, 확장성 이점 및 버전 관리 기능도 제공됩니다.

연민과 공감 dbt가 대문자로 시작되고 변

환 도구가 아닌 다른 것을 가리키는 텍스트에 나오는 이러한 용어에 익숙할 수도 있지만, 이 용어는 심리학에서와 마찬가지로 공학에서도 중요합니다. 동료(및 파트너)와 그들의 동기, 문제점 및 작업 흐름을 이해하면 귀하의 우려 사항을 효과적으로 전달하고 그들의 인센티브에 호소할 수 있습니다.

이 디지털 시대에는 특히 모호한 화상 회의와 간결한 텍스트 조각에서 적대적인 접근 방식을 취하거나 악의적인 의도를 가정하기가 너무 쉽습니다. 가능하다면 일대일, 장문 서면 의사소통, 대면 회의 등을 통해 동료를 이해하기 위해 더욱 노력하는 것이 좋습니다.

인센티브 조정 의미 있는 발

전을 촉진하려면 인센티브와 결과를 조정해야 합니다. 1 팀의 유일한 임무가 "많은 것을 구축"하는 것이라면 해당 요소를 탄력적이고 강력하게 만드는 데 시간이 거의 소요되지 않습니다.

1 이 섹션은 Barr Moses의 Monte Carlo 가이드, “[12 데이터 품질 실제로 중요한 측정항목](#)”.

일반적인 사고 관리 지표를 중심으로 핵심 성과 지표(KPI)를 설정하면 업무를 올바르게 수행하는 데 소요되는 시간과 에너지를 정당화하는 데 도움이 될 수 있습니다.

사고 건수(N)

특정 기간 동안의 사건 수를 계산하면 부정확하거나 불완전하거나 누락된 데이터의 빈도를 파악할 수 있습니다.

감지 시간(TTD)

TTD는 사고가 감지되는 데 걸리는 평균 시간을 나타냅니다.

해결 시간(TTR)

이 지표는 중단 후 시스템이 정상 작동을 재개할 수 있는 신속성을 측정합니다. 이는 시스템의 복원력과 복구 기능을 직접적으로 측정하는 것입니다.

데이터 다운타임($N \times [TTD + TTR]$)

위의 세 가지 지표를 사용하여 평균 "데이터 가동 중지 시간"에 도달할 수 있습니다. 이 요약 지표는 중단의 심각도와 시스템 상태를 이해하는 데 도움이 될 수 있습니다.

비용

가동 중지 시간을 통해 이러한 오류로 인한 비용을 계산할 수 있습니다.

비용은 팀과 조직에 따라 매우 다릅니다. 배포의 세부 사항을 고려하여 맞춤형 비용 계산을 권장합니다.

결과 개선 Thomas

Edison의 유명한 인용문인 "나는 실패하지 않았습니다. 단지 작동하지 않는 10,000가지 방법을 찾았을 뿐입니다."는 탁월한 데이터 파이프라인과 이를 지원하는 프레임워크를 구축하는 프로세스에 적절하게 적용됩니다.

이 분야에서 우수성을 향한 여정은 일련의 교육받은 추측, 실험, 그리고 결정적으로 도전에 직면했을 때 방향을 조정하고 수정하는 능력으로 표시됩니다.

지금까지 따라오셨다면 훌륭한 시스템이 훌륭한 프레임워크 위에 구축된다는 사실을 아실 것입니다. 실패 후 프로세스를 반복 및 개선하고 좋은 파이프라인을 조정하여 훌륭하게 만들 수 있는 몇 가지 방법은 다음과 같습니다.

선적 서류 비치

데이터가 준비되어 있고 백필이 가능한 경우 남은 것은 수정 방법을 아는 것뿐입니다! 시스템 프로세스와 코드를 문서화할 때는 지루하고 현명하게 행동하세요. 자질구레한 일처럼 느껴질 수도 있지만, 실패하는 동안 코드를 리엔지니어링하는 것보다 시간이 덜 걸리고 스트레스가 덜할 것이라고 보장할 수 있습니다.

사후 조사

대규모 이벤트나 중단이 발생한 경우 사후 분석은 오류를 분석하는 데 유용할 수 있습니다. 사후 분석에는 무엇이 잘못되었는지 반성하고 그 이유를 이해하기 위한 분석이 포함됩니다.

사후 분석과 일반적으로 성찰은 배우고, 교육하고, 성장하는 훌륭한 방법입니다. 이상적으로는 사후 분석을 통해 애초에 복구가 필요한 이벤트가 줄어들 것입니다.

단위 테스트 단

위 테스트는 작은 코드 조각(시스템의 구성 요소)을 검증하여 예상대로 결과가 나오는지 확인하는 프로세스입니다. 다른 엔지니어링 시스템과 마찬가지로 데이터 엔지니어링 시스템의 코드도 단위 테스트를 거쳐야 합니다. 이는 귀하가 생성하는 모든 사용자 정의 코드나 맞춤형 시스템이 원하는 결과를 생성하는지 확인하기 위한 테스트를 거쳐야 함을 의미합니다.

단위 테스트는 출력 데이터가 아닌 기본 코드를 확인하므로 이는 어설션과 특히 다릅니다. 코드에 단위 테스트를 구축하는 것은 향후 오류를 최소화하는 훌륭한 예방 방법입니다.

많은 플랫폼이 단위 테스트/어설션을 천천히 채택하고 있지만, 단위 테스트/어설션 없이 운영되는 데이터 팀이 늘라울 정도로 많습니다. 우리는 모든 데이터 시스템에서 이를 채택할 것을 응호합니다.

CI/CD

CI/CD(지속적 통합/지속적 배포)는 코드를 통합하고 배포하는 방법, 즉 변경 사항(예: 풀 요청)을 동화하고 테스트하고 코드 베이스에 블아웃하는 방법을 나타내는 용어입니다.

실제로 데이터 엔지니어링을 위한 CI/CD에는 이미 논의한 내용과 아직 다루지 않은 내용이 많이 포함될 수 있습니다. 단위 테스트, 어설션, Linting 및 데이터 차이점은 잘 작동하는 일관된 코드 기반을 보장하고 다른 사람들과 원활하게 협력하여 멋진 것을 대규모로 구축할 수 있도록 해줍니다.

단순화 및 추상화 도구라기보다는 디자

인 패턴에 더 가까운 엔지니어로서 여러분은 코드에서 최대한 복잡한 논리를 단순화하고 추상화하려고 노력해야 합니다. 이렇게 하면 공동 작업이 더 쉬워질 뿐만 아니라 오류가 발생할 가능성도 줄어듭니다.

코드를 작성할 때 "6개월 동안 보지 않으면 이해하기가 얼마나 어려울까?"라고 스스로 생각하십시오. 6개월 안에 문제가 발생하면 어떤 형태로든 압박감을 느낄 가능성이 높으므로 이를 염두에 두세요.

코드로서의 데이터 시스템

데이터 차이는 우리가 아직 논의하지 않은 개념, 즉 코드로서의 데이터 시스템 구축에 영향을 미칩니다. 모든 시스템의 버전을 관리하고 코드화하면 변경 사항을 룰백하고 이전 상태로 되돌릴 수 있습니다. 어떤 의미에서 메달리온 아키텍처를 갖춘 스테이징 시스템을 사용하면 시간 여행과 유사한 작업을 수행할 수 있습니다.

소프트웨어 엔지니어링 모범 사례를 사용하여 데이터 시스템을 구축하고 버전 제어 코드로 논리를 구현하면 관찰 가능성, 재해 복구 및 협업이 크게 향상됩니다. JSON 또는 YAML 구성만 있는 경우에도 버전 제어가 어렵거나 불가능한 도구 또는 코드를 통한 조작에 주의하세요.

책임이 정의되고, 인센티브가 조정되고, 전체 모니터링/문제 해결 툴킷이 제공되면 데이터 워크플로우 자동화 및 최적화를 시작할 준비가 된 것입니다. 일부 작업은 자동화할 수 없고 극단적인 경우가 항상 존재한다는 점을 인식하여 데이터 엔지니어링 기술은 자동화와 실용성의 균형을 맞추는 것입니다.

이제 강력하고 탄력적인 시스템을 통해 확장할 준비가 되었습니다.

제5장

효율성과 확장성

이 마지막 장에서는 우리가 개발한 데이터 파이프라인을 최적화하고 확장하는 중요한 측면에 중점을 둡니다. 논의의 경계를 설정하기 위해 "효율성"과 "확장성"이 무엇을 의미하는지 정의하는 것부터 시작합니다.

우리의 여정은 우리의 운영 환경에 대한 철저한 이해에 달려 있는 자원 할당에서 시작됩니다. 이러한 이해를 통해 우리는 프로세스를 효과적으로 최적화할 수 있습니다.

이 장은 이중 초점을 맞춘 토론으로 마무리됩니다. 먼저, 협업 프로세스, 특히 팀 규모와 기술 세트 측면에서 효과적으로 확장하는 방법을 탐구합니다. 둘째, 효율적인 데이터 파이프라인 관리의 핵심 요소인 최적의 개발자 경험을 창출하는 방법을 탐구합니다.

이 장 전반에 걸쳐 도구 사용 및 플랫폼 고려 사항, 관리형 솔루션과 맞춤형 솔루션의 장단점, 우수한 ETL 시스템을 제작하기 위한 아키텍처 전략과 같은 지속적인 주제를 다루었습니다. 이러한 논의의 목표는 효율적이고 확장 가능한 데이터 시스템을 구축하고 유지 관리하는 포괄적인 관점을 제공하는 것입니다.

효율성과 확장성의 정의

효율성은 데이터를 통해 비즈니스 가치를 제공하기 위해 워크플로를 최적화하는 것입니다. 이는 코드, 서비스 및 팀워크 측면을 포괄하는 자원을 활용하여 영향력 있는 결과를 생성하는 능력을 측정합니다. 효율성의 궁극적인 척도는 사용된 유한한 자원에 비해 발생하는 영향입니다.

확장성이란 점점 늘어나는 작업량을 처리할 수 있는 시스템, 네트워크 또는 프로세스의 능력이나 이러한 성장을 수용하기 위해 확장될 수 있는 잠재력을 말합니다. 데이터 변환의 맥락에서 이는 성능이나 효율성을 저하시키지 않고 증가하는 데이터 볼륨과 더욱 복잡한 변환 작업을 처리할 수 있는 데이터 처리 시스템의 능력에 관한 것입니다.

소규모 전자상거래 웹사이트의 데이터를 초기에 처리하는 데이터 변환 시스템을 상상해 보십시오. 비즈니스가 성장함에 따라 수천 명에서 수백만 명에 이르는 고객, 거래, 제품 상호 작용 등의 데이터도 성장합니다. 이러한 맥락에서 확장성은 시스템이 데이터 처리 기능을 확장하여 이러한 성장을 관리할 수 있음을 의미합니다.

데이터 팀은 데이터 변환 시스템을 확장할 때 가격과 성능 간의 상충 관계에 자주 직면합니다. 더 강력한 컴퓨팅, 정교한 솔루션 또는 확장된 클라우드 스토리지를 통해 성능이 향상되면 비용도 높아지는 경우가 많습니다.

인생과 마찬가지로 데이터에서도 균형이 중요합니다. 말처럼 쉽지는 않지만, 우리의 목표는 지속 가능한 가격으로 딱 맞는 성능을 제공하는 솔루션을 찾는 것입니다. 예를 들어, 사용량 기반 가격 책정은 서비스 평가를 시작하는 좋은 방법이 될 수 있지만 일반적으로 어느 시점에서는 구독이나 균일 가격 책정으로 인해 가려집니다. 데이터 팀은 확장 결정의 비용 효율성을 평가하여 투자에 대한 최적의 성능을 얻고 불필요한 용량에 과도한 비용을 지출하지 않도록 해야 합니다.

그렇다면 효율성, 확장성, 비용 효율성의 균형을 어떻게 맞출 수 있을까요?

4 장에서 논의한 것처럼 데이터 품질, 최신성 및 양에 대한 명확한 지표를 설정하는 것이 중요합니다. 이러한 지표는 최적화 또는 컴퓨팅 리소스 증가에 대한 투자를 정당화할 수 있습니다. 예를 들어, 팀이 매시간 데이터를 제공해야 하는데 현재 시스템이 너무 느리거나 비용이 많이 드는 경우 이러한 측정항목을 사용하면 최적화나 예산 조정을 더 쉽게 주장할 수 있습니다.

효율성과 확장성에 관한 지표도 유용할 수 있습니다. 다음은 효율성/확장성에 관한 주요 결과(KR)의 몇 가지 예입니다.

- 평균 처리 속도 25% 증가 • 스트리밍 작업에서 200ms 미만의 자연 시간 유지 • 데이터 저장 비용 15% 감소 • 성능 저하 없이 스팟 인스턴스 사용량 30% 증가

KR은 자신의 성과와 상관관계가 있는 KPI를 설정하여 얻을 수 있습니다.
결과:

처리 시간
주어진 양의 데이터를 처리하는 데 걸리는 시간

처리량
특정 기간 동안 처리된 데이터의 양

시스템 가동 시간
시스템이 사용 가능하고 작동 가능한 시간 비율

오류율
데이터 파이프라인의 오류 빈도

팀 OKR 및 KPI를 이해관계자 기대치에 맞추면 확장성과 최적화 노력에 리소스가 적절하게 할당됩니다.

데이터 최신성이나 품질과 같은 제약 조건과 함께 이 장에서 논의한 사용 가능한 도구와 리소스를 이해하면 최적의 성능과 비용을 위해 워크플로를 미세 조정할 수 있습니다.

환경 이해

데이터 엔지니어링의 확장은 기본적으로 아키텍처에 뿌리를 두고 있습니다.
효과적인 확장을 위해서는 올바른 프레임워크와 기반을 구축하는 것이 중요합니다. 데이터 발전으로 인해 구축을 더 쉽게 시작할 수 있게 되었지만, 이러한 용이성으로 인해 처음에는 작동하지만 유지 관리 부담이 되거나 확장에 어려움을 겪는 시스템을 만드는 위험도 발생합니다.

속담처럼, 사후 판단은 20/20이며, 모든 구현은 필연적으로 학습과 최적화 기회로 이어질 것입니다. 성공적인 확장의 핵심은 환경을 깊이 이해하여 데이터 시스템을 위한 강력하고 지속적인 기반을 구축하는 것입니다.

프레임워크

시스템 최적화는 시스템에 대한 철저한 이해에서 시작됩니다.

따라서 리소스 할당, 효율성 향상, 워크플로 최적화에 대해 알아보기 전에 우리가 운영 중인 프레임워크를 이해해야 합니다.

실행 메커니즘을 더 잘 이해할수록 작업을 더 효과적으로 최적화할 수 있습니다. 예를 들어 Spark에서 Pandas를 사용하는 경우 모범 사례를 읽어보세요. 문서는 언제나 [시작하기 좋은 곳입니다](#). BigQuery에 대해 SQL을 실행하는 경우 쿼리 계획과 실행 순서를 이해하세요. 다시 한 번, 문서가 [구세주가 될 수 있습니다](#).

고려해야 할 확장성에 대한 다른 기본 개념에는 OLAP 및 OLTP 데이터베이스 구별, Spark의 논리적 및 물리적 계획 이해, 스팟 인스턴스와 주문형 인스턴스 구별, 선택한 분산 컴퓨팅 시스템(Databricks, Kubernetes 또는 심지어 AWS Lambda)도 리소스와 규모를 할당합니다.

자원 할당

견고한 아키텍처 기반을 갖춘 데이터 시스템의 확장성은 효과적인 리소스 할당에 크게 좌우됩니다. 올바른 도구를 갖는 것은 방정식의 일부일뿐입니다. 진정으로 중요한 것은 이를 활용하는 방법입니다. 핵심은 가장 적절한 상황에서 강점을 활용하는 것입니다. 그렇다면 자원배분에 있어 어떤 요소를 고려해야 할까요? 다음 섹션에서는 명심해야 할 몇 가지 중요한 측면에 대해 설명합니다.

병렬화 및 동시성 데이터 워크플로

를 위해 완전히 관리되는 서비스 플랫폼(가능한 옵션)을 활용하지 않는 경우 일 반적으로 ETL 파이프라인 실행 방법을 어느 정도 제어할 수 있습니다. 현재 환경에서는 분산 컴퓨팅이

클러스터와 노드의 성능은 주로 전 세계 데이터 센터에 위치한 공유 가상 머신에서 수행됩니다.

설정에는 Kubernetes와 같은 자체 호스팅 실행 환경이나 Databricks와 같은 공유 환경이 포함될 수 있습니다. 각 옵션은 리소스 할당, 확장성 및 유지 관리 측면에서 고유한 고려 사항을 제시합니다.

클러스터

ETL 워크플로우를 관리할 때 직면하게 될 중요한 결정은 클러스터 구성과 관련됩니다. 클러스터는 통합 시스템으로 함께 작동하는 서버 모음이며, 각 서버를 "노드"라고 합니다. 이러한 노드는 워크플로를 실행하고 모니터링하는 역할을 담당합니다.

클러스터는 다목적이거나 특정 작업일 수 있습니다. 즉, 임시 쿼리와 일반 분석 또는 데이터 파이프라인의 특정 작업에 사용할 수 있습니다. 클러스터 구성 프로세스에는 노드 수와 유형에 대한 결정이 포함됩니다. 이러한 선택은 메모리 및 처리 능력과 같은 주요 요소에 영향을 미치며 컴퓨팅 용량 및 관련 비용에 직접적인 영향을 미칩니다.

클러스터 구성에 대한 미묘한 이해를 통해 워크플로를 다양한 설정으로 사용자 정의하거나 확장 및 구성을 자동화할 수도 있습니다. 이를 통해 비용 효율성과 성능 사이에서 최적의 균형을 유지할 수 있습니다. 다음 단락에서는 클러스터 실행을 관리하기 위한 몇 가지 효과적인 기술을 살펴보겠습니다.

스팟과 온디맨드 클러스터 인스턴스 스팟 인스턴스

스는 일반적으로 온디맨드 인스턴스보다 훨씬 낮은 가격으로 비용 효율적인 솔루션을 제공하므로 예산에 민감한 프로젝트에 실용적인 선택이 됩니다. 즉각적인 결과가 필요하지 않은 일괄 처리 또는 데이터 분석과 같이 유연성을 수용하고 중단을 허용할 수 있는 작업에 특히 효과적입니다. 그러나 가용성은 시장 수요에 따라 가변적이며 공급자는 경고 없이 이를 회수할 수 있어 잠재적으로 진행 중인 작업이 중단될 수 있습니다.

이와 대조적으로 온디맨드 인스턴스는 지속적인 서비스를 보장하여 안정성과 일관된 성능을 제공합니다. 이는 운영에 중요하고 일관된 가용성이 중요한 실시간 처리가 필요한 작업에 이상적으로 적합합니다. 그러나 이

안정성과 일관성 수준은 스팟 인스턴스에 비해 더 높은 비용으로 제공됩니다.

스팟 또는 온디맨드 인스턴스 사용 결정은 각 작업의 특정 요구 사항을 기반으로 해야 합니다. 여기에는 온디맨드 인스턴스의 안정성 및 중단 없는 성능에 대한 필요성과 비교하여 비용 효율성의 중요성과 스팟 인스턴스의 잠재적 중단 처리 능력과 같은 요소를 고려하는 것이 포함됩니다.

풀링 풀

온 리소스를 반복적으로 생성하고 파괴하는 비용이 많이 드는 프로세스를 피하기 위해 데이터 엔지니어링에 활용됩니다. 특히 풀은 유휴 상태로 유지되고 사용할 준비가 된 클러스터 그룹을 나타냅니다.

풀 사용을 지원하는 플랫폼이나 시스템을 선택하면 솔루션의 비용 효율성을 크게 향상시킬 수 있습니다. 클러스터를 쉽게 사용할 수 있게 함으로써 풀은 클러스터를 시작하고 자동 크기 조정하는 데 걸리는 시간을 줄일 수 있습니다.

예를 들어 관리형 플랫폼에서 Spark로 작업할 때 Databricks 풀과 같은 기능을 활용하는 것이 비용 효율적인 전략이 될 수 있습니다.

이러한 풀에 스팟 인스턴스를 통합하면 비용을 더욱 줄일 수 있습니다. [그림 5-1](#)은 Databricks 풀을 사용하여 비용을 효과적으로 절감하는 방법을 보여줍니다.

동시에 더 빠른 실행 시간이 필요하거나 엄격한 요구 사항이 있는 작업에 온디맨드 인스턴스를 사용하면 성능 속도를 향상시킬 수 있습니다. 이 접근 방식을 사용하면 리소스를 균형 있게 사용하여 데이터 엔지니어링 워크플로에서 비용과 효율성을 모두 최적화할 수 있습니다.

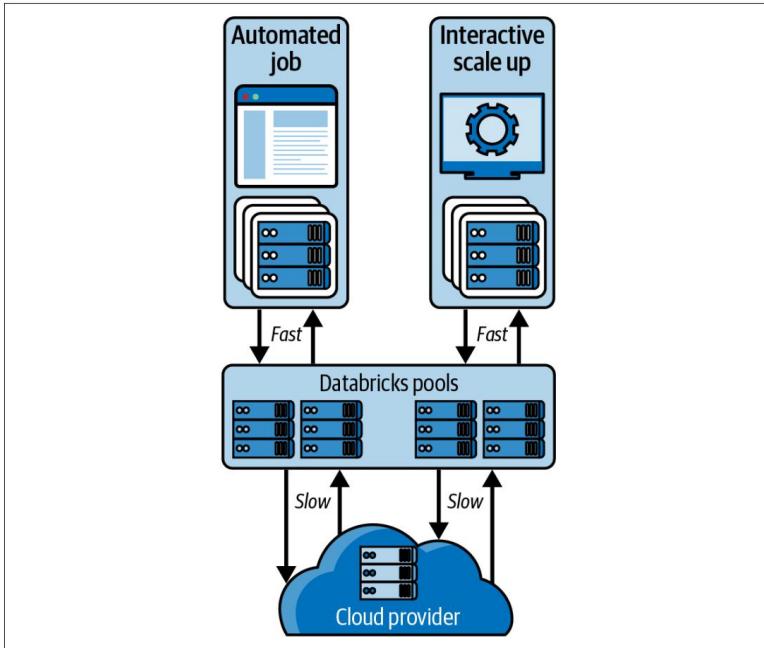


그림 5-1. Databricks 풀 성능 저하 없이 가격을 낮추는 데 사용할 수 있음

클러스터 공유

유공유 클러스터는 데이터 엔지니어링에서 리소스 할당에 대한 비용 효율적인 접근 방식을 제공합니다. 여러 사용자가 동일한 컴퓨팅 리소스에 동시에 워크로드를 연결하고 실행할 수 있도록 함으로써 공유 클러스터는 눈에 띄는 비용 절감 효과를 가져옵니다. 이는 클러스터 관리를 단순화하고 정확한 액세스 제어 조치를 포함한 포괄적인 데이터 거버넌스를 촉진합니다. "공유"라는 용어의 의미에도 불구하고 이러한 클러스터는 효율적인 리소스 활용을 통해 비용을 절감하는 안전한 방법을 제공합니다.

자동 크기 조정

정 크기 조정에는 두 가지 유형이 있습니다. 수평적 확장은 작업이 실행되는 노드 또는 기계의 수를 늘리는 것을 의미합니다. 수직적 확장은 기존 리소스의 크기나 성능을 높이는 것을 의미합니다. 예를 들어 데이터베이스의 스토리지가 부족한 경우 스토리지 용량을 늘려 수직으로 확장할 수 있습니다. 또는 대규모 리소스 집약적 작업의 성능을 향상시키기 위해 더 많은 머신을 추가하여 수평적 확장을 선택할 수도 있습니다.

그림 5-2는 시각적 표현을 제공합니다.

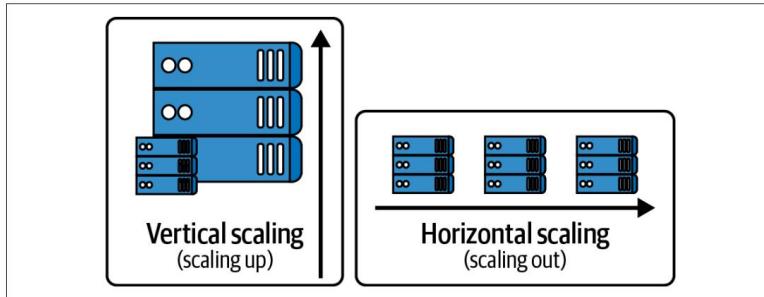


그림 5-2. 수직 대 수평 확장 - 둘 다 적절한 시간과 장소가 있습니다.

자동 확장에는 변동하는 워크로드에 맞게 리소스를 자동으로 조정하는 도구와 기술이 포함됩니다. 전형적인 예는 웹 트래픽이 크게 급증하는 블랙 프라이데이에 미국 소매 웹사이트를 들 수 있습니다. 일년 내내 높은 수준의 컴퓨팅 리소스를 유지하는 것은 엄청나게 많은 비용이 듭니다.

대신, 예상치 못한 경우에도 자동 확장을 사용하여 이러한 트래픽 급증을 효율적으로 처리할 수 있습니다.

요즘에는 데이터 웨어하우스부터 Spark 실행 프로그램에 이르기까지 자동 확장 기능을 제공하는 클라우드 서비스가 점점 늘어나고 있습니다. 관리형 서비스로 자동 확장하면 상당한 시간과 리소스를 절약하여 DevOps 작업 부담을 줄일 수 있습니다. 그러나 이러한 편리함에는 비용이 따르며 관리형 자동 크기 조정 서비스를 활용하는 정도는 팀의 특정 요구 사항과 개발 단계에 따라 결정되어야 합니다.

서비스

서비스 컴퓨팅은 미리 정해진 할당이 아닌 실제 사용량을 기반으로 백엔드 서비스를 제공하는 모델입니다.

자원의. 서비스는 여전히 이 모델에 필수적이지만, 구별되는 측면은 가격 구조입니다. 서비스 컴퓨팅 서비스를 사용하는 회사는 고정 대역폭이나 설정된 서버 수가 아닌 사용량을 기준으로 비용을 청구합니다.

BigQuery 및 Databricks SQL로 예시되는 서비스 솔루션은 변동하는 워크로드에 따라 자동 확장하는 기능을 제공합니다.

이러한 적응성은 리소스를 세심하게 조정해야 할 필요성을 크게 줄이거나 제거하는 경우도 있습니다. 서비스 컴퓨팅은 맞춤형 가격 구조를 갖는 경우가 많지만 특정 유지 관리가 없기 때문에 많은 사람들에게 매력적인 옵션이 됩니다.

데이터 처리 기술 리소스 할당 방법에 관계없이 최

적의 효율성을 위해서는 여전히 데이터를 처리해야 합니다. 웨어하우스든 데이터 레이크든 파이프라인을 구축할 때 고려해야 할 몇 가지 개념은 다음과 같습니다.

증분 처리 전체 데이터 세트를

처리하는 데는 특히 데이터 양이 증가함에 따라 엄청나게 비용이 많이 들 수 있습니다. 잘라내기 및 다시 로드는 간단한 접근 방식이지만 실행 불가능하거나 규모에 따라 비용이 지나치게 많이 드는 지점에 도달합니다. 새 데이터만 추가 (INSERT)하거나 새 데이터를 삽입하는 동안 기존 데이터를 업데이트 (UPSERT) 하는 증분 처리는 보다 확장 가능한 대안을 제시합니다.

증분 처리 로직은 빠르게 복잡해질 수 있습니다. [dbt 증분 모델](#)과 같이 증분 처리를 위해 사전 구축된 패턴이 있는 도구를 찾는 것이 좋습니다. 변경 데이터 캡처 또는 [Databricks MERGE](#).

열 기반 데이터 저장소

데이터 및 분석의 경우 열 기반 형식과 데이터베이스는 상당한 성능 향상을 제공합니다. 데이터 분석에는 열 단위 읽기가 포함되는 경우가 많기 때문에 데이터를 열 형식으로 저장하는 것이 본질적으로 더 효율적입니다. 이와 대조적으로 Postgres 와 같은 기존 OLTP 데이터베이스는 행 중심이므로 프로덕션 환경에서처럼 데이터가 한 번에 한 행씩 자주 쓰거나 읽는 시나리오에 적합합니다.

오늘날 데이터 및 분석 팀은 열 기반 저장소를 선호합니다. 최적의 분석 성능을 얻으려면 Parquet 및 Avro와 같은 데이터 형식과 BigQuery 및 DuckDB와 같은 데이터베이스를 활용하는 것이 좋습니다. 두 데이터의 시각화는 [그림 5-3을 참조하세요.](#)

백화점.

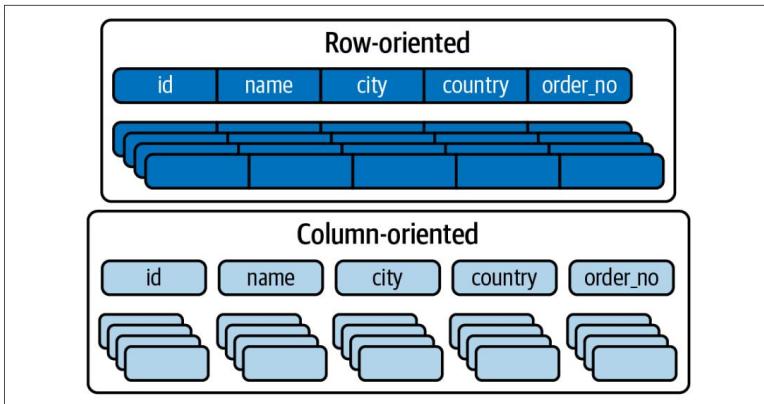


그림 5-3. 행 및 열 기반 데이터베이스는 데이터 및 앱 개발에서 중추적인 역할을 하지만 사용 사례는 매우 다릅니다.

이러한 열 기반 저장 방법이 파티셔닝(또는 데이터를 분할하는 다른 방법)과 결합되면 데이터 처리를 위한 매우 강력하고 효율적인 패턴이 생성됩니다. Databricks SQL과 같은 데이터베이스는 Parquet 기반 형식(Delta Lake)에 직접 구축되어 [유동 클러스터링과 같은 비용 최적화 및 성능 향상을 실현합니다.](#)

데이터 분할 데이터

를 "분할"한다는 것은 단순히 데이터를 여러 부분으로 나누는 것을 의미합니다. 파티셔닝은 읽기/쓰기 성능에 매우 중요하므로 간과해서는 안 됩니다. Parquet와 같은 형식과 PyArrow 와 같은 라이브러리를 사용하여, 데이터를 분할하는 것은 간단할 수 있습니다.

분할의 효율성은 열 형식 데이터세트에서 특히 두드러집니다. 10개 열에 100일 동안의 데이터가 포함된 데이터세트라는 일반적인 시나리오를 가정해 보겠습니다. 다음 쿼리를 고려해보세요.

날짜 = '2024-01-01' 인 데이터에서 user_id를 선택하세요 .

분할되지 않은 행 기반 데이터 세트에서 이 쿼리는 모든 행과 열을 검색하므로 실행 시간이 느려지고 컴퓨팅 비용이 증가합니다.

그러나 날짜별로 분할된 컬럼 중심의 데이터베이스에서는 쿼리가 필요한 데이터만 효율적으로 처리합니다. 여러 날에 걸쳐 데이터가 균등하게 분포된다고 가정하면 분할된 열 기반 데이터베이스에서 하루 동안 한 열의 데이터에 액세스하는 것이 1000배 더 효율적입니다. 하나의 쿼리에서는 큰 차이가 없을 수도 있지만 이를 수백 개의 대시보드, 쿼리 및 파이프라인에 쌓으면 빠르게 이점을 얻을 수 있습니다. 그림 5-4는 분할이 데이터 세트를 분할하고 성능을 향상시키기 위해 어떻게 작동하는지 보여줍니다.

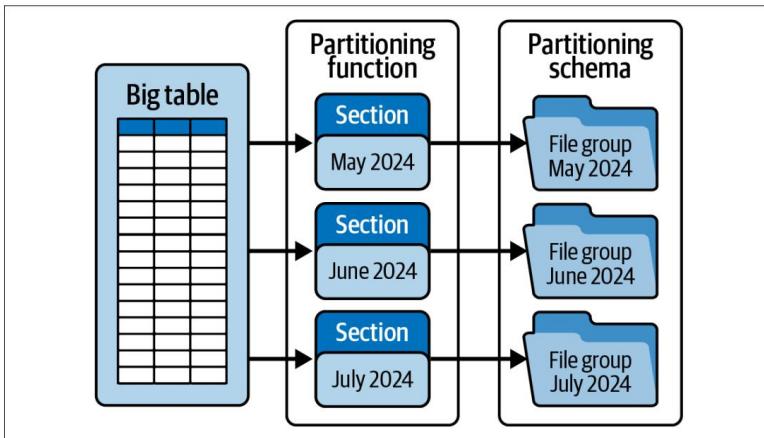


그림 5-4. 데이터 분할은 쿼리 성능과 스토리지 효율성을 모두 향상시킵니다.

파티션을 생성하고 유지 관리할 필요가 없는 관리형 서비스도 살펴볼 가치가 있습니다. 파티셔닝은 생성 및 유지 관리를 위해 도메인 지식과 사려 깊은 엔지니어링이 필요한 어려운 프로세스일 수 있습니다. 데이터 세트를 자동으로 분할, 정렬 및 최적화하는 시스템은 그 만한 가치가 있습니다.

비용.

구체화

구체화란 물리적 데이터 저장소, 즉 테이블을 생성하는 것을 의미합니다. "물리적"이 문자 그대로의 의미보다 더 개념적이기 때문에 이 용어는 디지털 시대에 약간 오해를 불러일으킬 수 있습니다. 최신 데이터베이스는 특정 쿼리에서 파생된 데이터를 기본적으로 가상으로 표현하는 "뷰"를 지원하는 경우가 많습니다. 예를 들어 `SELECT * FROM users WHERE status = 'active'` 와 같은 쿼리를 실행하고 이를 `active_users` 뷰로 정의할 수 있습니다. 그러면 이 보기는 다음과 같이 작동합니다.

소스 데이터에서 최소한의 변화가 필요한 분석 준비 데이터 세트에 액세스하는 편리한 방법을 제공하는 테이블이었습니다.

"외부 뷰"는 외부에 저장된 데이터에 대해 생성된 뷔를 의미합니다.

마지막으로, 예를 들어 데이터 레이크에서 그렇습니다. 반구조화된 데이터를 데이터 레이크에 저장하고 이를 외부 뷔로 직접 쿼리하는 추세가 점점 늘어나고 있습니다. 이 접근 방식은 데이터 변화의 대부분이 데이터 웨어하우스로 이동되는 데이터 엔지니어링의 "우측 이동"을 나타냅니다. 이를 통해 분석가 및 기타 이해관계자는 반구조화된 데이터와 직접 상호 작용하여 프로세스를 간소화할 수 있습니다.

데이터 변환을 계획할 때 테이블, 뷔 또는 구체화된 뷔를 사용할지 신중하게 고려하는 것이 중요합니다. 각 뷔는 데이터 시스템 아키텍처에 고유한 의미와 이점을 갖고 있기 때문입니다.

프로세스 효율성

제대로 작동하는 프로토타입이나 생산 솔루션을 갖추는 것이 중요합니다. 작동하는 시스템이 있으면 관찰 가능성이 핵심이 됩니다. 이는 솔루션의 성능을 모니터링하고 측정할 수 있는 능력이 있음을 의미합니다. 고려해야 할 질문은 다음과 같습니다. 솔루션이 리소스를 얼마나 잘 관리합니까? 병목 현상이 있나요?

실행 후에는 어떻게 되나요?

시스템 작동을 관찰하여 통찰력을 얻은 후에는 초기 가정과 결정을 재평가하는 것이 중요합니다.

엔지니어링은 본질적으로 계획 단계에서 명확하지 않았던 새로운 통찰력을 기반으로 지속적인 학습과 적응을 수반합니다. 최적화는 요구 사항을 재평가하고, 단순화를 추구하고, 자동화를 탐색하는 지속적인 프로세스입니다.

데이터(엔지니어링) 민주화 데이터 민주화는 데이터 팀을 넘

어 전체 조직에서 데이터 기반 의사 결정에 액세스할 수 있도록 하는 것을 의미합니다. 일반적으로 분석과 관련되어 이해관계자가 분석 팀에 의존하지 않고 독립적으로 데이터를 탐색할 수 있도록 하는 셀프 서비스 분석 도구 제공이 수반되는 경우가 많습니다.

데이터 엔지니어링 팀의 경우 데이터 민주화에는 리소스 생성과 ETL 작업 개발을 단순화하는 시스템과 프레임워크를 만드는 것이 포함됩니다. 예는 다음과 같습니다.

이해관계자는 새로운 맞춤형 API 데이터로 데이터를 강화해야 합니다.

소스(Fivetran과 같은 커넥터가 없음). 분석 팀이 데이터 소스에서 가져오고, 스테이지 테이블에 쓰고, 데이터 웨어하우스에 수집하기 위한 ETL 작업을 요청합니다.

이제 나와 함께 일한 많은 사람들은 "물론입니다. 지금부터 5번의 스프린트 안에 이를 맞출 수 있습니다."라고 말합니다. 이는 비즈니스 관점에서 보면 손실입니다. 데이터 엔지니어링 백로그에 추가되고 중요한 보충 데이터가 지연됩니다. 5번의 스프린트와 2개월 후, 회사는 새로운 KPI를 출시했고 이니셔티브는 강등되었으며 파이프라인은 사용되지 않았습니다. 지금까지 사용자(우리 이해관계자!)와 이전하면 EOD는 "12월 말"로 바뀌고 상황은 훨씬 덜 효율적이 됩니다.

데이터 엔지니어링의 민주화는 셀프 서비스 시스템과 플랫폼을 구축하는 것을 의미합니다. 이러한 시스템의 목표는 분석가 또는 기타 이해관계자가 필요한 작업에 대해 차단되지 않은 채 대부분의 제품 작업을 수행할 수 있도록 하는 것입니다. 시스템 구축은 다음 프로세스를 따릅니다.

1. 데이터 엔지니어가 일반적으로 요청하는 요청이나 패턴을 식별합니다.

자주 마주치는

2. 이러한 요청의 필요성 평가 3. 프로세스를 최대한 단순

화 4. 기술이 부족한 사용자도 사용할 수 있는 SaaS 제품 또 는 맞춤형 솔루션 탐색

5. 지속적인 단순화 및 최적화 6. 이점 활용

이러한 시스템에는 여전히 코드 검토나 풀 요청이 필요할 수 있지만 대부분 자동적으로 작동해야 합니다. 여기에는 분석자가 SQL 및 YAML을 제출하여 자동으로 Airflow 작업을 생성하는 저장소를 설정하거나 대화형 파이프라인 개발을 위해 하이브리드 GUI/코드 솔루션을 사용하는 것이 포함될 수 있습니다.

이 접근 방식을 용이하게 하는 도구가 점점 더 널리 보급되고 있습니다.

이러한 시스템을 구현하는 데이터 팀의 능력은 효과적으로 확장하는 팀과 끝없는 요청으로 인해 수령에 빠진 팀 간의 주요 차별화 요소로 떠오르고 있습니다. 이는 확장하는 팀과 내년에도 JIRA 보드가 잘 채워지는 팀 간의 차별화 요소입니다.

데이터 민주화는 확장성의 독특한 측면을 나타냅니다.
 로우 코드, 노 코드 또는 맞춤형 사내 플랫폼을 통해 셀프 서비스를 지원함으로써 데이터 팀의 병목 현상을 완화합니다.
 셀프 서비스 기능이 없으면 데이터 팀은 임시 요청으로 인해 끊임없이 압도당할 위험이 있습니다.

개발자 경험 ETL 시스템의 효율

성과 확장성은 코드 워크플로 최적화 및 확장에 중점을 두고 있지만, 이러한 시스템을 뒷받침하는 데이터 엔지니어링 팀의 경우 개발자 경험(DevEx)을 향상시키는 것이 중요합니다. Noda et al. DevEx의 "핵심 차원"을 **그림 5-5**에 표시된 흐름 상태, 피드백 루프 및 인지 부하로 정의합니다.

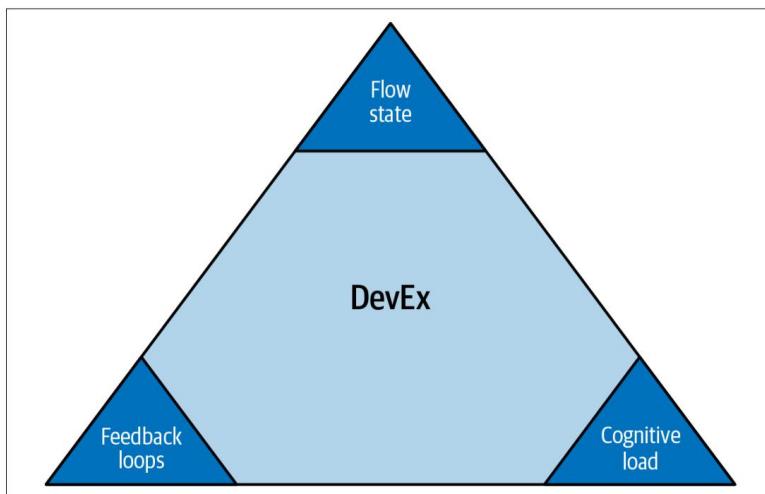


그림 5-5. 개발자 경험은 흐름 상태, 피드백 루프, 인지 부하의 세 가지 주요 영역으로 나눌 수 있습니다.

피드백 루프 피드

백 루프는 작업에 대한 응답 속도와 품질로 정의할 수 있습니다. Airflow에서 DAG를 개발하는 프로세스를 고려해보세요. 로컬에서 쉽게 개발할 수 있나요? 변경하면 해당 변경 사항이 귀하에게 어떤 영향을 미치는지 즉시 확인할 수 있습니까?

¹ Noda, A. 등. 2022. “DevEx: 실제로 생산성을 높이는 것은 무엇입니까? 개발자는-생산성 측정 및 개선에 대한 중심적 접근 방식입니다.” 소프트웨어 엔지니어링 기초(FSE)에 관한 2022 ACM SIGSOFT 심포지엄 진행 중(pp. 32–47). ACM. <https://dl.acm.org/doi/10.1145/3610285>.

체계? 귀하의 데이터는 무엇입니까? 우리는 데이터 엔지니어가 "프론트엔드 DevEx"를 목표로 삼아야 한다고 믿습니다. 프론트엔드 엔지니어는 한 장에서 코딩하고 다른 장에서 실시간 변경 사항을 확인한 다음 몇 가지 간단한 명령으로 변경 사항을 배포할 수 있습니다. 유사한 로컬 DevEx 를 생산하기 위해 시스템을 어떻게 구축할 수 있습니까?

인지 부하 인지 부하

는 작업에 필요한 정신적 처리의 양입니다. 로컬 개발 환경을 구성하는 것이 얼마나 어렵나요? 새로운 DBT 모델을 만들려면? 환경의 기존 테이블, 인프라 및 메타데이터를 이해하고 싶으십니까? 업무를 수행하기 위해 일반적으로 7가지 도구 사이를 전환하고 있습니까? 엔지니어로서 우리는 개발에 대한 인지적 부하를 단순화하여 최소화하도록 노력해야 합니다. 영향력 있는 작업에 집중할 수 있도록 구성과 구축을 최대한 단순화하는 시스템에 투자하세요.

흐름 상태 흐름

은 집중, 참여, 즐거움을 느끼는 몰입의 정신 상태입니다. 누구나 흐름 상태(flow state)에 대해 들어봤고 우리 모두는 그것을 추구합니다. 거기까지 어떻게 가나요? 짧은 피드백 루프와 낮은 인지 부하로 의미 있고 영향력 있는 작업을 수행합니다. 첫 번째 구성 요소는 주로 관심에 따라 결정되므로 여기서는 도움을 드릴 수 없습니다. 하지만 스트레스가 적은 환경에서 신속한 개발을 촉진하도록 시스템을 조정하면 흐름을 찾는 데 도움이 됩니다.

협동

마지막 섹션은 협업에 관한 것입니다. 정말 놀랍습니다. 혼자서 할 수 있는 것보다 다른 사람과 함께 할 때 더 많은 일을 해낼 수 있을 것입니다! 데이터 엔지니어링의 세계에서 이는 효과적일 뿐만 아니라 다른 사람들이 쉽게 이해하고, 확장하고, 참여할 수 있는 솔루션을 만들기 위해 노력하는 것을 의미합니다.

코드형 인프라 인프라의 모든 부

분은 버전이 제어되고 테스트 가능한 코드로 존재해야 합니다. 이를 통해 Git과 같은 기술과 GitHub와 같은 제공업체를 활용하여 엔지니어의 엄격함을 바탕으로 구축할 수 있습니다. 예, 잊어버린 "Git 명령 하나"를 Stack Overflow에서 찾아 해매게 될 것입니다. 그러나 무언가를 깨뜨리고 완전히 다시 빌드하는 것보다 훨씬 낫습니다.

이는 GUI 기반의 코드가 적거나 코드가 없는 도구의 경우 특히 중요합니다. 완전히 기록하지 말고 솔루션에 최소한 버전이 지정된 구성 파일과 설정을 백업/복원할 수 있는 방법이 있는지 확인하십시오. 마지막으로 원하는 것은 파이프라인을 재구축하기 위해 10번의 클릭이 필요한 도구입니다. 처음 실패를 경험하면 그 100개 파이프라인을 재구축하는 악몽은 즐겁지 않을 것입니다.

문서화 및 지식 공유 문서화는 흔히 자질구레한 일로 여겨

지지만 효과적인 위기를 전달하고 작성하는 데에는 기술이 있습니다. 당신이 하는 모든 일은 문서화하고 후손(그리고 당신 자신)을 위해 저장해야 합니다. 문제를 해결해 드리겠습니다. 작년에 구축한 것을 기억하려고 하거나 자원 없이 다른 사람의 코드를 파헤쳐 보는 것은 악몽이 될 수 있습니다. 작성한 코드를 문서화하세요. 데이터를 문서화하십시오. 노력할만한 가치가 있다는 것이 보장됩니다.

결론

데이터 엔지니어링의 효율성과 확장성은 가장 빠르고 성능이 뛰어난 파이프라인을 구축하는 영역을 훨씬 뛰어넘습니다. 여기에는 우리 모두가 필연적으로 직면하는 자원 제한뿐만 아니라 협업의 중요한 역할과 절충의 필요성에 대한 더 넓은 이해가 포함됩니다.

효율적인 알고리즘과 데이터 구조의 코딩과 구현에 대한 기술적 전문성도 중요하지만, 소프트 스킬도 마찬가지로 중요합니다. 효과적인 의사소통, 철저한 문서화, 숙련된 팀 관리는 모두 성공적인 데이터 엔지니어링 관행의 필수 구성 요소입니다. 이 장과 전체 가이드의 목표는 데이터 파이프라인을 효과적이고 신속하게 확장할 수 있는 포괄적인 기술 세트를 갖추는 것입니다. 비슷한 야심찬 가상의 우주비행사의 말을 인용하면 "무한대 이상"으로 확장할 수 있습니다.

결론

기술 가이드인 ETL 이해를 이용해 주셔서 감사합니다. 이 여정에서 우리는 데이터 수집의 복잡성을 탐구하고, 데이터 변환의 미묘한 차이를 깊이 파고들었으며, ETL 저류의 복잡성(오케스트레이션, 문제 해결, 확장성)을 해결했습니다. 그 과정에서 우리는 깨끗하고 신뢰할 수 있는 데이터 세트를 만들기 위한 지속적인 방법론에 중점을 두고 진화하는 데이터 엔지니어링 환경에 대해서도 조명했습니다.

우리는 지속적인 관련성을 지닌 개념을 다루는 것을 목표로 삼았지만, 데이터 엔지니어링 분야가 향후 몇 년간 크게 발전할 가능성이 있다는 점을 인식하는 것이 중요합니다. 그러나 소스에서 데이터를 추출하고 변환하여 타겟에 로드하는 ETL의 기본 프로세스는 향후 기술 발전이나 ETL 시스템 구축 방법의 변화와 관계없이 초석으로 남을 것으로 예상됩니다.

이 가이드는 데이터 엔지니어링에 대한 기본 개념을 제공하여 지식의 격차를 식별하고 메우는 데 도움을 줍니다.

데이터 환경은 방대하고 복잡할 수 있지만 데이터를 효과적으로 탐색하려면 주요 구성 요소와 더 큰 그림을 이해하는 것이 중요합니다.

기술적 측면 외에도 우리는 데이터 엔지니어링과 그 이상에서 중추적인 두 가지 모두 솔루션 아키텍처 및 개발자 경험 향상과 같은 주제를 엮었습니다. 동료, 사용자, 그리고 그들이 직면한 과제를 이해하는 것의 중요성을 강조하면 당신은 더욱 능숙한 엔지니어가 될 뿐만 아니라 더욱 정확하고 효과적인 솔루션으로 이어질 것입니다. 끊임없는 단순화와 신중한 의사 결정이라는 철학을 채택하면 작업에 변화를 가져올 수 있습니다.

마지막으로, 더 넓은 데이터 커뮤니티에 적극적으로 참여하시기 바랍니다. 오픈 소스 프로젝트에 기여하거나, 컨퍼런스에 참석하거나, 기사를 작성하거나, 소셜 미디어에서 통찰력을 공유하는 등 귀하의 참여는 엄청난 성취감을 줄 수 있습니다. 커뮤니티에 참여하면 개인의 성장과 학습이 촉진될 뿐만 아니라 데이터 엔지니어링 관행의 집단적 발전에도 기여합니다. 행운을 빕니다. 거기서 뵙기를 바랍니다!

저자 소개

Matt Palmer는 제품 분석 및 데이터 엔지니어링에 대한 배경 지식을 갖춘 Replit의 개발자 경험 엔지니어입니다. 여가 시간에는 글쓰기, 하이킹, 근력 운동을 즐깁니다. 그는 베이 지역에 거주하며 대부분의 주말을 캘리포니아를 탐험하며 보냅니다.