AI□□□ □□□□□□□□□ □□□□□□□□□ □□□□□□
AWS □□□□□□□□□□□□ □□□□□□□□□

Contents

AI와 오믹스 연구를 위한 AWS 클라우드 컴퓨팅

안준용 (고려대학교 보건과학대학 바이오시스템의과학부)

유전체 분석에 AWS 클라우드를 쓰기 시작한 것은 2015년이었다. 그때부터 지금까지 SFARI 재단의 수만 명 규모 유전체 데이터를 분석하는 데 AWS를 활용해 왔고, 최근에는 AI 모델 개발과 바이브 코딩에도 같은 플랫폼을 쓰고 있다. 이 책은 그 10년의 경험을 연구실 신입 대학원생과 학부 연구생에게 체계적으로 넘겨 주기 위해 쓴 교육 자료에서 출발했다. 동시에 임상유전학, 집단유전학, 단일세포 생물학, 공간 전사체학, 생물정보학 교육을 담당하는 유관 분야의 연구자들이 자신의 연구실로 같은 플랫폼을 도입하려 할 때 참고할 수 있는 지침서가 되었으면 하는 바람으로 쓴 책이기도 하다.

유전체 연구에서 클라우드가 필요해진 이유는 단순히 데이터가 커졌기 때문이 아니다. 한 사람의 전장유전체를 읽으면 수백 기가바이트가 쌓이고, 여기에 RNA-seq, single-cell, 공간 전사체, 이미징 오믹스까지 더해지면 연구실은 곧바로 저장 공간 부족이 아니라 데이터 이동, 환경 관리, 재현 가능한 분석 구조의 문제에 부딪힌다. gnomAD가 S3에 자원을 올려 두고, NCBI SRA가 클라우드 안에서 직접 접근을 열어 두는 시대에는 `download → analyze`가 아니라 `bring compute to data`가 기본 운영 방식이 되었다. 이 책은 그 전환을 어떻게 연구실의 일상 운영으로 가져올지를 다룬다.

이 책의 독자

이 책은 AWS 서비스 매뉴얼이 아니다. 유전체 분석을 처음 배우는 학부 고학년과 대학원 초급 연구자가 AWS의 어떤 층이 어떤 문제를 푸는지 구조적으로 이해할 수 있게 쓰려고 했다. 임상유전학이나 집단유전학, 단일세포 전사체학을 연구하지만 클라우드는 처음 접하는 연구자가 자기 연구실의 데이터 운영을 어디서부터 설계해야 할지 감을 잡을 수 있게 쓰려고 했다. 기존에 온프레미스 서버 중심으로 연구해 온 PI가 "우리 연구실도 클라우드로 옮겨야 하는가"라는 질문에 구체적인 선택지를 가지고 답할 수 있게 쓰려고 했다. 기술 용어가 처음 등장할 때마다 풀어서 설명했고, 왜 그 서비스가 그 자리에 있는지 이유를 함께 적었다. 쉽게 쓴다는 것이 부정확하게 쓴다는 뜻이어서는 안 되기 때문이다.

책의 구성

이 책은 4개의 파트로 구성되어 있다.

Part 1은 클라우드와 오믹스 데이터의 기본기를 다룬다. 왜 오믹스 연구가 클라우드를 필요로 하게 되었는지, EC2와 S3와 EBS와 IAM Role이 각각 어떤 문제를 푸는지, gnomAD와 SRA 같은 공용 자원을 어떻게 해서 쓸지를 설명한다.

Part 2는 AWS에서 오믹스 파이프라인을 운영하는 법을 다룬다. EC2에서 시작하는 실전 분석 환경부터 Ansible, AWS Batch, Lambda를 이용한 자동화, Spot과 Fleet를 이용한 대규모 병렬 작업, EMR과 Hail로 하는 코호트 규모 WGS 분석, 그리고 AWS HealthOmics와 Nextflow 파이프라인까지 이어진다.

Part 3은 데이터 접근과 분석 도구를 다룬다. htslib의 S3 스트리밍, goofys와 s3fs 같은 마운트 도구, SageMaker 기반 오믹스 머신러닝, Zarr와 AnnData와 OME-Zarr와 TileDB-SOMA로 대표되는 클라우드 네이티브 single-cell과 spatial omics, 그리고 AWS에서 Claude와 Kiro로 하는 바이브 코딩을 다룬다.

Part 4는 실제 연구 워크플로우를 다룬다. 시퀀싱 데이터에서 변이 해석까지의 통합 예제, 비용과 보안과 재현성을 함께 설계하는 방법, 한국 연구실을 위한 운영 체크리스트, Bedrock으로 하는 유전체 질의와 변이 해석, 산업계 사례와 데이터 브로커 운영, 그리고 1PB 규모 WGS 운영에서 배우는 비용 누수 지점과 해결 전략을 다룬다.

각 장의 끝에는 해당 장에서 인용한 공식 문서와 논문 목록을 실었다. 관심 있는 독자가 원문을 직접 찾아볼 수 있도록 하기 위함이다.

서문

이 책은 "클라우드와 AI를 이용해 오믹스 연구를 실제로 어떻게 시작할 것인가"라는 질문에 답하기 위해 구성했다. 독자는 리눅스 명령어와 기본적인 생물정보학 분석 개념은 조금 알고 있지만, AWS는 처음 접하는 수준을 기본 가정으로 한다. 따라서 각 장에서는 서비스 이름을 소개하는 데 그치지 않고, 어떤 문제를 해결하기 위해 그 서비스가 등장했는지부터 설명한다. gnomAD의 공개 S3 데이터, SRA의 클라우드 전송, Hail과 EMR, AWS HealthOmics, SageMaker, 시퀀싱 센터 데이터 공유와 전처리, 대규모 작업 자동화, 재현성 검증, S3 직접 접근 도구, single-cell과 spatial omics의 chunk 기반 접근처럼 실제 오믹스 연구실에서 마주치는 사례를 중심으로 내용을 전개한다. 책의 후반부에서는 Claude와 Kiro를

이용한 코드 생성, 그리고 Bedrock 기반 변이 질의·해석 자동화처럼 AI가 오믹스 워크플로에 들어오는 최신 흐름도 함께 다룬다.

책 전체의 무게중심은 유전체 변이 분석이다. 데이터 크기와 운영 복잡성이 가장 먼저 드러나는 영역이고, 클라우드와 AI가 해결해 온 문제의 원형이 이곳에 있기 때문이다. Part 3 이후로는 single-cell, spatial omics, 이미징 오믹스로 시야를 넓히며, 데이터 모델이 파일에서 chunk 기반 원격 접근으로 바뀌는 흐름, 그리고 생성형 AI가 해석 계층으로 자리 잡는 흐름을 함께 설명한다.

이 책의 독자

- 클라우드와 AI를 처음 배우는 생명과학, 의생명과학, 생물정보학 전공 학생
- 로컬 서버 중심으로 분석해 왔지만 클라우드로 확장하고 싶은 오믹스 연구자
- 공개 유전체·single-cell·spatial 데이터를 AWS에서 직접 다루는 법을 알고 싶은 초급 분석가
- 연구실에 AI 기반 질의·해석 계층을 어떻게 도입할지 고민하는 PI와 대학원생

읽는 순서

- 처음 읽는 경우에는 1장부터 4장까지 순서대로 읽으며 클라우드와 오믹스 데이터의 기본 사고방식을 익힌다.
- 리눅스와 생물정보학 파이프라인에 익숙하다면 5장부터 실전 운영 파트로 넘어가도 된다.
- single-cell과 spatial omics에 관심이 있다면 13장을 중심으로, AI 활용 흐름에 관심이 있다면 12장, 14, 18을 연달아 읽으면 좋다.
- 특정 서비스가 필요하다면 EMR, HealthOmics, SageMaker, Bedrock 장을 개별적으로 참고할 수 있다.
- 책 말미의 용어집(Glossary)은 언제든지 되돌아볼 수 있는 참고 자료로 활용한다.

1장. 왜 오믹스 연구에 AWS를 쓰는가

학습 목표

- 오믹스 연구에서 클라우드를 쓰는 대표적인 이유를 설명할 수 있다.
- 로컬 서버와 클라우드의 장단점을 비교할 수 있다.
- 계산 자원, 저장 공간, 협업, 공개 데이터 접근의 관점에서 AWS의 의미를 이해할 수 있다.

핵심 질문

- 왜 대규모 오믹스 분석은 개인 워크스테이션만으로 감당하기 어려운가
- 클라우드는 단순히 “비싼 서버”가 아니라 무엇이 다른가
- 어떤 유형의 오믹스 프로젝트가 AWS와 잘 맞는가

오믹스 데이터가 클라우드를 필요로 하게 된 배경

유전체 분석이 클라우드를 필요로 하게 된 첫 번째 이유는 데이터의 크기와 구조가 동시에 바뀌었기 때문이다. 한 사람의 전장유전체서열 분석(whole-genome sequencing, WGS)도 원시 FASTQ와 정렬 결과, 변이 파일, 품질 관리 결과를 모두 합치면 쉽게 수십에서 수백 기가바이트 규모가 된다. 여기에 RNA-seq, single-cell RNA-seq, 공간전사체(spatial transcriptomics), 장기 보관용 원본 데이터까지 더해지면 문제는 단순한 저장 공간 부족을 넘어선다. 같은 데이터를 여러 번 복사하고, 서로 다른 서버에서 서로 다른 버전의 참조 유전체와 소프트웨어를 쓰고, 분석이 끝난 뒤에도 중간 산물을 정리하지 못하는 일이 연구실의 일상이 된다. 즉 오늘날의 병목은 계산 속도만이 아니라 데이터 이동, 환경 관리, 그리고 반복 가능한 분석 구조에 있다.

예전에는 파일을 한 번 내려받아 연구실 서버에서 오래 붙잡고 분석하는 방식이 어느 정도 통했다. 그러나 코호트 규모가 커지고 공용 데이터 자원이 확대되면서 이 모델은 점점 비효율적이 되었다. gnomAD처럼 대규모 참조 자원을 객체 스토리지에 두고 공개하는 사례나, NCBI SRA가 클라우드 안에서 직접 접근하는 방식을 제공하는 사례는 공통된 메시지를 보여 준다. 핵심은 데이터를 사용자 컴퓨터로 모두 가져오는 것이 아니라, 데이터가 이미 존재하는 클라우드 환경으로 계산을 가져가는 것이다. 이른바 `bring compute to data`라는 사고방식은 단순한 유행어가 아니라, 대규모 오믹스학이 실제로 작동하기 위한 운영 원칙이 되었다 (gnomAD team 2020; NCBI 2024).

온프레미스 서버와 클라우드의 차이

온프레미스(on-premises) 서버는 연구실이나 기관이 직접 구매하고 운영하는 계산 자원이다. 이런 방식은 장기간 안정적으로 돌아가는 파이프라인, 엄격한 내부 보안 통제, 예측 가능한 연중 상시 workload에 강점이 있다. 반면 초기 구축 비용이 크고, 갑자기 수백 개 샘플이 몰리거나 특정 프로젝트에서만 매우 큰 메모리와 CPU가 필요할 때 유연하게 확장하기 어렵다. 반대로 AWS 같은 클라우드는 필요한 시점에만 인스턴스를 켜고, 일이 끝나면 끌 수 있다는 점에서 탄력성이 크다. 초보자가 흔히 오해하듯 클라우드는 "남의 서버를 빌리는 일"에 그치지 않고, 저장, 권한, 로그, 워크플로, 자동 확장, 공용 데이터 접근이 하나의 운영 체계로 묶인 환경이다 (AWS 2024a; NIH Cloud Lab 2024).

Table 1은 온프레미스와 AWS를 연구 운영 관점에서 단순화해 비교한 것이다. 여기서 중요한 점은 어느 한쪽이 항상 더 낫다는 결론이 아니라, 어떤 문제를 풀고 있는지에 따라 적합한 선택이 달라진다는 사실이다. 예를 들어 매주 비슷한 수의 샘플을 처리하는 임상 파이프라인은 온프레미스나 하이브리드 구성이 더 경제적일 수 있다. 반면 공개 데이터 자원을 바로 읽어야 하거나, 짧은 기간에 수백에서 수천 개의 작업을 병렬로 돌려야 하는 연구 프로젝트는 클라우드가 훨씬 자연스럽다. 실제 연구 현장에서는 둘 중 하나만 쓰기보다, 원본 보관과 일부 민감 데이터는 기관 내부에 두고 대규모 계산과 공개 데이터 활용은 클라우드에서 수행하는 혼합 구조가 자주 사용된다.

Table 1. 온프레미스 서버와 AWS의 비교

항목	온프레미스 서버	AWS 클라우드
초기 비용	서버, 스토리지, 네트워크를 먼저 구매해야 함	사용량 기반으로 시작 가능
확장성	증설에 시간과 구매 절차가 필요함	필요 시 즉시 확장 가능
공개 데이터 접근	외부 데이터 반입과 저장이 병목이 되기 쉬움	같은 클라우드의 공개 데이터에 직접 접근 가능
운영 부담	패치, 하드웨어, 장애 대응을 직접 담당	인프라 운영 일부를 관리형 서비스로 이전 가능
적합한 상황	예측 가능한 상시 workload, 내부 규제 중심 환경	변동성 큰 연구 workload, 대규모 병렬 처리, 공용 데이터 활용

저장, 계산, 협업, 재현성의 네 가지 관점

클라우드를 이해하는 가장 좋은 방법은 서비스 이름부터 외우는 것이 아니라, 저장과 계산과 협업과 재현성이라는 네 가지 질문으로 나누어 보는 것이다. 먼저 저장 관점에서 보면, 현대 오믹스 연구는 운영체제가 설치된 로컬 디스크와 장기 보관용 데이터 저장소를 분리해야 한다. Amazon S3 같은 객체 스토리지(object storage)는 FASTQ, BAM, CRAM, VCF, 보고서, 참조 데이터처럼 오래 보관하고 여러 서비스가 함께 읽어야 하는 자산에 적합하다. 반면 계산 노드에 직접 붙는 디스크는 정렬 중간 파일이나 임시 작업 공간처럼 짧은 시간 동안 높은 I/O가 필요한 작업에 더 어울린다. 학생이 처음 AWS를 배울 때 와 를 구분하는 습관을 들이는 것이 중요하다.

계산 관점에서는 더 명확한 변화가 보인다. 로컬 워크스테이션에서는 사용 가능한 CPU 수와 메모리가 고정되어 있고, 동시에 여러 사람이 같은 자원을 쓰면 병목이 즉시 드러난다. AWS에서는 Amazon EC2로 직접 서버를 띄울 수도 있고, AWS Batch로 샘플 단위 작업을 큐에 넣을 수도 있으며, Amazon EMR이나 AWS HealthOmics처럼 더 높은 수준의 관리형 분석 계층을 선택할 수도 있다. 이 차이는 단순히 "더 큰 컴퓨터"를 얻는 데 있지 않다. 오히려 계산 자원을 프로젝트의 길이와 크기에 맞추어 켰다 끄는 운영 방식, 그리고 같은 파이프라인을 다른 규모로 반복 실행할 수 있는 구조가 핵심이다. 따라서 클라우드는 계산 능력 자체보다 계산을 조직하는 방법을 바꾸는 도구라고 이해하는 편이 더 정확하다 (AWS 2024a; Data Carpentry 2024).

협업과 재현성의 의미도 달라진다. 예전에는 각 연구실 구성원이 자기 서버에 파일을 복사하고, 조금씩 다른 경로와 버전의 스크립트를 사용하며, 나중에 분석을 재현하지 못하는 일이 흔했다. 클라우드에서는 같은 S3 경로, 같은 컨테이너 이미지, 같은 입력 manifest, 같은 워크플로 정의를 공유함으로써 출발점을 맞추기 쉬워진다. 특히 권한을 IAM Role처럼 실행 주체에 붙이고, 로그와 메타데이터를 중앙에서 관리하면 "누가 언제 어떤 입력으로 어떤 분석을 돌렸는가"를 훨씬 명확하게 추적할 수 있다. 재현성(reproducibility)은 논문을 쓸 때만 필요한 덕목이 아니라, 협업 규모가 커질수록 비용과 시간을 줄여 주는 실용적 운영 기술이라는 점을 초반부터 이해할 필요가 있다 (AWS 2024b).

AWS 서비스 지형도 – storage, workflow, query, AI

초보자는 AWS 서비스 이름이 너무 많아서 어디서부터 시작해야 할지 막막해한다. 이때 중요한 것은 모든 서비스를 다 배우는 것이 아니라, 어떤 층(layer)의 문제를 푸는 서비스인지 이해하는 것이다. 유전체 분석에서 자주 만나는 AWS 서비스는 크게 데이터 저장, 계산 실행, 워크플로 자동화, 질의와 해석, 시각화와 개발 보조의 다섯 층으로 나눌 수 있다. 이 가운데 AWS HealthOmics는 유전체와 오믹스 workflow를 위해 설계된 purpose-built service이고, 나머지 다수는 범용 AWS 서비스를 유전체 문제에 맞게 조합하는 방식이다. 즉 유전체 분석을 AWS에서 한다는 말은 "유전체 전용 서비스만 쓴다"는 뜻이 아니라, 범용 클라우드 도구와 생명과학 특화 서비스를 함께 엮어 쓴다는 뜻이다.

Table 2는 이 책에서 반복해서 등장할 서비스를 교육용으로 단순화한 지도다. 처음부터 모든 세부 기능을 외울 필요는 없다. 다만 어떤 서비스가 데이터 저장 계층이고, 어떤 서비스가 실행 계층이며, 어떤 서비스가 질의와 해석 계층인지만 알아도 책의 뒤쪽 장을 훨씬 쉽게 따라갈 수 있다. 예를 들어 S3, EC2, IAM Role만 이해해도 가장 기초적인 AWS 기반 유전체 분석은 시작할 수 있다. 그 위에 Batch, EMR, HealthOmics, Athena, SageMaker, Bedrock 같은 계층을 차례로 얹는다고 생각하면 전체 구조가 훨씬 단순해진다.

Table 2. 오믹스 입문자를 위한 AWS 서비스 지도

층	대표 서비스	책에서의 역할
저장	Amazon S3, S3 Glacier storage classes, Amazon EBS	원본 데이터, 결과 파일, 장기 보관, 임시 작업 디스크
데이터 이동	AWS DataSync	시퀀싱 센터, 병원, 로컬 HPC, 다른 클라우드에서 데이터 반입
직접 계산	Amazon EC2	분석 서버, 노트북 환경, 맞춤형 도구 실행
배치와 오케스트레이션	AWS Batch, Amazon EKS, AWS ParallelCluster	대규모 병렬 처리, Kubernetes 운영, Slurm/HPC 스타일 클러스터
분산 분석	Amazon EMR	Spark 기반 cohort-scale 데이터 처리
전용 오믹스 실행	AWS HealthOmics	Nextflow, WDL, CWL 기반 관리형 오믹스 workflow
질의와 요약	Amazon Athena, Amazon Quick	S3 위 데이터 SQL 질의, cohort 요약과 시각화
ML/AI	Amazon SageMaker, Amazon Bedrock	모델 학습, 추론, 생성형 AI 기반 해석
개발 보조	Kiro	코드 작성과 workflow 설계 보조

초보 연구자가 AWS를 배울 때 가장 먼저 버려야 할 오해

초보자가 가장 먼저 버려야 할 오해는 클라우드가 무조건 비싸다는 생각이다. 클라우드는 잘못 쓰면 분명 비싸질 수 있지만, 반대로 몇 달에 한 번 있는 대형 분석을 위해 상시 서버를 미리 사 두는 방식도 결코 싸지 않다. 비용은 자원 자체보다 운영 방식에서 갈리는 경우가 많다. 필요한 동안만 계산 자원을 켜고, 중간 파일을 정리하고, 같은 리전에 데이터를 두고, 적절한 인스턴스를 선택하면 클라우드는 오히려 비용 구조를 더 투명하게 만든다. 이 책의 후반부에서 다루겠지만, 실제 대규모 WGS 프로젝트의 비용 누수는 "고성능 컴퓨터를 썼기 때문"보다 incomplete upload, request cost, intermediate data explosion, 불필요한 데이터 이동에서 더 자주 발생한다.

두 번째 오해는 AWS를 쓰려면 모든 데이터를 반드시 내 버킷으로 옮겨야 한다는 생각이다. 실제로는 공용 자원과 직접 연결하는 방식이 점점 더 중요해지고 있다. gnomAD, SRA, AWS Open Data 같은 사례는 클라우드 시대의 분석이 `download -> analyze`보다 `query -> access in place -> compute`에 가깝다는 점을 보여 준다. 세 번째 오해는 AWS가 대기업이나 전문 엔지니어만의 도구라는 생각이다. 실제로는 EC2 한 대와 S3 한 버킷, 그리고 적절한 IAM Role만으로도 매우 많은 교육용 실습과 중소 규모 연구를 시작할 수 있다. 마지막으로 버려야 할 오해는 모든 서비스를 다 알아야만 시작할 수 있다는 생각이다. 처음에는 EC2, S3, IAM Role이라는 세 가지 축만 제대로 이해하고, 이후 필요에 따라 Batch, EMR, HealthOmics, SageMaker로 확장해 가면 충분하다.

AWS는 오믹스 연구를 자동으로 더 똑똑하게 만들어 주는 마법 도구가 아니다. 대신 데이터가 커지고, 협업이 복잡해지고, 공용 자원이 클라우드에 존재하는 시대에 맞는 연구 운영 방식과 분석 구조를 제공한다. 따라서 AWS를 배운다는 것은

서비스 이름을 많이 외우는 일이 아니라, 어떤 데이터를 어디에 두고, 어떤 계산을 언제 켜고, 누구와 어떤 방식으로 같은 출발점을 공유할 것인지 설계하는 일에 가깝다. 이런 관점으로 접근하면 뒤이어 나올 EC2, S3, gnomAD, SRA, EMR, HealthOmics 장이 서로 분리된 서비스 설명이 아니라 하나의 연구 인프라 이야기로 연결되기 시작한다.

핵심 개념 정리

- 현대 유전체학에서 클라우드의 핵심 가치는 “큰 서버를 빌리는 것”이 아니라 “데이터가 있는 곳에서 계산하고 협업하는 것”이다.
- 저장과 계산은 분리해서 생각해야 하며, 객체 스토리지와 실행 노드는 서로 다른 역할을 가진다.
- 재현성은 논문 작성 단계의 문제가 아니라, 분석 운영 전체를 안정화하는 실용적 원칙이다.
- AWS는 범용 서비스와 유전체 특화 서비스를 함께 조합하는 구조로 이해해야 한다.

복습 질문

1. 왜 대규모 오믹스 연구에서는 `download -> analyze`보다 `bring compute to data`가 더 중요해졌는가?
2. 온프레미스 서버와 AWS는 각각 어떤 연구 상황에서 더 적합한가?
3. AWS 입문자가 처음부터 모든 서비스를 배우기보다 EC2, S3, IAM Role에 먼저 집중해야 하는 이유는 무엇인가?

Further Reading

- Langit L. AWS for Bioinformatics.
- Data Carpentry. Cloud Genomics.
- NIH Cloud Lab. AWS Jumpstart.

References

- AWS. 2024a. Genomics Data Transfer, Analytics, and Machine Learning on AWS.
- AWS. 2024b. Genomics on AWS Best Practices.
- Data Carpentry. 2024. Cloud Genomics.
- gnomAD team. 2020. Open access to gnomAD data on multiple cloud providers.
- NCBI. 2024. SRA in the Cloud.
- NIH Cloud Lab. 2024. AWS Jumpstart.

2장. EC2, S3, EBS, IAM Role 이해하기

학습 목표

- EC2 인스턴스, S3, EBS의 역할 차이를 구분할 수 있다.
- 블록 스토리지와 객체 스토리지의 차이를 유전체 데이터 사례로 설명할 수 있다.
- IAM User와 IAM Role의 차이를 이해하고 기본 보안 원칙을 설명할 수 있다.

핵심 질문

- EC2는 무엇을 위한 서비스인가
- S3와 EBS는 각각 언제 쓰는가
- 왜 액세스 키를 파일에 박아 넣는 대신 IAM Role을 써야 하는가

리전, 가용 영역, 계정이라는 기본 단위

AWS를 처음 배울 때 가장 먼저 익혀야 하는 것은 개별 서비스보다 공간적 단위와 관리 단위다. 리전(region)은 물리적으로 떨어진 지리적 영역이고, 가용 영역(availability zone, AZ)은 한 리전 안에서 전력과 네트워크 측면에서 어느 정도 분리된 데이터센터 묶음이라고 이해하면 된다. 계정(account)은 이런 자원을 누구의 소유와 과금 체계 아래에서 운영할 것인지를

결정하는 관리 단위다. 오믹스 분석에서는 계산 자원과 저장 자원이 같은 리전에 있는지가 매우 중요하다. 같은 AWS 안에 있어도 리전이 달라지면 데이터 전송 비용과 지연 시간이 커질 수 있고, 워크플로 전체의 단순성도 떨어진다.

이 개념이 중요한 이유는 유전체 데이터가 크기 때문이다. 몇 메가바이트짜리 로그 파일을 옮길 때는 잘 드러나지 않지만, 수백 기가바이트 FASTQ나 여러 샘플의 BAM, CRAM을 옮기기 시작하면 위치가 비용과 성능을 직접 좌우한다. 따라서 초급자도 "어느 리전에 버킷을 만들 것인가", "어느 리전에서 EC2를 띄울 것인가", "공개 데이터가 어느 리전에 있는가"를 먼저 확인하는 습관을 가져야 한다. 이후 배우게 될 S3, EC2, Batch, HealthOmics, EMR 모두 이 기본 단위 위에서 움직인다. 서비스 이름을 모를 때도 , 를 먼저 묻는다면 실수를 크게 줄일 수 있다.

EC2 인스턴스 이름 읽기 – `m`, `c`, `r`, `i`, `f`, `hpc`

Amazon EC2는 AWS에서 가장 기본적인 계산 서비스다. 가장 단순하게 말하면, 필요할 때 원하는 크기의 가상 서버를 켜는 서비스다. 오믹스 분석에서는 정렬, 변이 호출, QC, 노트북 실습, 맞춤형 파이프라인 실행처럼 "직접 소프트웨어를 설치하고 돌려야 하는 계산"이 주로 EC2 위에서 돌아간다. 초보자에게는 인스턴스 종류가 너무 많아 보이지만, 실제로는 이름을 읽는 규칙만 이해해도 전체 지형이 빠르게 정리된다. 핵심은 어떤 인스턴스가 더 비싼지가 아니라, 어떤 병목에 맞추어 설계되었는지를 읽는 것이다.

AWS 인스턴스 이름에서 가장 먼저 볼 것은 패밀리(family)다. `m`은 general purpose, `c`는 compute optimized, `r`은 memory optimized, `i`나 `d` 계열은 storage optimized, `f`는 FPGA 같은 가속기를 쓰는 accelerated computing, `hpc`는 긴밀한 병렬 통신이 필요한 high-performance computing 계열로 이해할 수 있다. 그 뒤의 숫자는 세대(generation), 마지막 크기 표시는 인스턴스 크기(size)를 뜻한다. 따라서 `r7i.4xlarge` 같은 이름을 보면, 최신 세대의 메모리 중심 인스턴스라는 사실부터 파악하면 된다. 이 책에서 중요한 것은 모든 이름을 외우는 일이 아니라, 이름에서 의도를 읽는 감각을 기르는 일이다 (AWS 2024a).

세대와 접미사까지 함께 읽으면 정보가 훨씬 풍부해진다. `c5`는 Xeon 기반의 5세대 compute optimized, `c6i`는 Intel Ice Lake 기반의 6세대, `c6a`는 AMD EPYC 기반의 6세대, `c6g`는 AWS Graviton2 기반, `c7g`는 Graviton3, `c8g`는 Graviton4 기반 최신 세대를 뜻한다. 접미사 `i`는 Intel, `a`는 AMD, `g`는 Graviton이라는 프로세서 종류를 가리키고, 여기에 다시 붙는 `d`는 NVMe 로컬 SSD(instance store)를, `n`은 네트워크 대역폭 강화를, `e`는 메모리 확장을 의미한다. 따라서 `c8gd.8xlarge`는 `Graviton4 compute optimized, NVMe , 8xlarge` 라고 한 문장으로 풀어 읽을 수 있다. 같은 논리로 `r7iz`는 `Ice Lake (z) `, `m7a.xlarge`는 `AMD 7 general purpose`로 읽힌다.

크기 표시도 규칙적이다. `large`, `xlarge`, `2xlarge`, `4xlarge`, `8xlarge`, `12xlarge`, `16xlarge`, `24xlarge`, `48xlarge`, `metal` 순으로 vCPU와 메모리가 대체로 두 배씩 늘어난다. 같은 `c7g` 안에서 `c7g.large`(2 vCPU / 4 GiB)부터 `c7g.16xlarge`(64 vCPU / 128 GiB)까지 스펙이 선형적으로 확장되므로, 우선 패밀리와 세대를 고르고 나서 크기만 필요에 맞게 고르면 된다. 오믹스 분석에서는 인스턴스를 고를 때 구체적인 이름이 곧 예산과 병목을 결정하므로, 다음 Table 1-1처럼 최근 세대의 대표 선택지를 머릿속에 하나씩 갖고 있는 편이 실용적이다.

Table 1-1. 2026년 기준 compute optimized 계열 대표 예시

이름	프로세서	로컬 SSD	대표 크기	오믹스에서 자주 쓰는 맥락
`c5.9xlarge`	Intel Xeon (5세대)	없음	36 vCPU / 72 GiB	기존 파이프라인 호환, 레거시 워크로드
`c6i.8xlarge`	Intel Ice Lake	없음	32 vCPU / 64 GiB	표준 variant calling, bwa, GATK
`c6a.16xlarge`	AMD EPYC	없음	64 vCPU / 128 GiB	CPU 집약 Batch, 가격 대비 CPU 시간
`c7g.16xlarge`	AWS Graviton3	없음	64 vCPU / 128 GiB	ARM 빌드 가능한 최신 툴 (samtools, bcftools)
`c7gd.8xlarge`	Graviton3	NVMe SSD 2×1.9 TB	32 vCPU / 64 GiB	정렬 sort, shard 기반 scratch
`c8g.24xlarge`	Graviton4	없음	96 vCPU / 192 GiB	최신 대규모 CPU 병렬 작업

이름	프로세서	로컬 SSD	대표 크기	오믹스에서 자주 쓰는 맥락
c8gd.16xlarge	Graviton4	로컬 NVMe 포함	64 vCPU / 128 GiB	최신 scratch-heavy 정렬·sort

메모리형(r6i, r7g, r8g, r7iz)이나 storage-optimized(i4i, i7ie, i8g, im4gn, is4gen), 범용(m6i, m7g, m8g)도 같은 규칙으로 읽으면 된다. 예를 들어 Hail joint genotyping처럼 메모리 압박이 큰 작업에는 r7iz.32xlarge(고주파 Xeon, 128 vCPU / 1,024 GiB)가, sort 중간 파일이 많은 정렬 워크플로에는 로컬 NVMe가 붙은 i7ie.6xlarge나 c8gd.8xlarge가 자연스럽다. 반대로 "뭘 골라야 할지 모르겠다"는 상황에서는 m 계열 최신 세대(m8g.xlarge 같은 것)를 기본값으로 두고, 실제 실행 로그를 보면서 병목을 확인한 뒤 c·r·i로 바꿔 나가는 편이 안전하다. 실제 구체 인스턴스 종류와 온디맨드 가격은 AWS EC2 Instance Types 페이지에서 최신 목록을 확인할 수 있다 (AWS 2024a).

오믹스 분석용 컴퓨트 선택 – 메모리형, CPU형, IO형, 가속형

오믹스 분석에서 인스턴스를 고를 때 초급자가 가장 흔히 하는 실수는 "일단 큰 서버면 좋다"라고 생각하는 것이다. 실제로는 CPU가 병목인지, 메모리가 병목인지, 로컬 디스크 I/O가 병목인지, 아니면 특정 가속기를 쓸 수 있는지에 따라 선택이 달라져야 한다. 예를 들어 정렬과 일반적인 variant calling은 CPU 사용량이 높고 샘플 단위 병렬화가 쉬우므로 c 계열이 잘 맞는 경우가 많다. 반대로 Hail aggregation, joint genotyping, 대형 annotation join처럼 메모리 압박이 큰 단계는 r 계열이 더 자연스럽다. Notebook 기반 탐색이나 가벼운 분석 서버는 m 계열로도 충분한 경우가 많다.

로컬 scratch 공간을 많이 쓰는 작업은 따로 생각해야 한다. 정렬 중간 산물, shard 파일, 대형 sort, 일시적인 matrix 변환처럼 짧은 시간 동안 많은 읽기·쓰기를 반복하는 작업은 i 또는 d 계열이 유리할 수 있다. 반면 FPGA 기반 가속 소프트웨어를 쓸 수 있을 때만 f 계열이 의미가 있다. 예를 들어 DRAGEN처럼 하드웨어 가속을 전제로 설계된 파이프라인은 일반 CPU 인스턴스와 다른 성능·비용 구조를 보일 수 있지만, 모든 분석이 자동으로 빨라지는 것은 아니다. 즉 accelerated instance는 "범용 고성능"이 아니라, 해당 가속기를 이해하는 소프트웨어와 함께 쓸 때만 강한 특수 선택지라고 보는 편이 정확하다.

Table 1은 오믹스 분석에서 자주 만나는 workload와 인스턴스 패밀리를 교육용으로 정리한 것이다. 실제 선택에서는 데이터 크기, 사용 도구, 병렬화 방식, 비용 제약까지 함께 봐야 한다. 그럼에도 이 정도의 첫 지도가 있으면 초급자도 최소한 를 설명할 수 있다. 좋은 선택은 늘 가장 비싼 인스턴스가 아니라, 병목에 가장 잘 맞는 인스턴스다. 이 판단 습관은 이후 Spot, Batch, EMR, HealthOmics를 사용할 때도 그대로 이어진다.

Table 1. 오믹스 분석에서의 대표 인스턴스 선택

인스턴스 패밀리	대표 병목	유전체 예시
m	균형형	노트북 서버, 일반 분석 환경, 소규모 테스트
c	CPU	정렬, 일반 batch 작업, 샘플 단위 variant calling
r	메모리	joint genotyping, 대형 annotation join, Hail 집계
i / d	로컬 I/O	sort, shard 기반 중간 파일 처리, scratch-heavy workflow
f	가속기	DRAGEN 같은 FPGA 기반 파이프라인
hpc	긴밀한 병렬 통신	EFA가 필요한 tightly coupled 계산

S3, EBS, FSx – 데이터가 머무는 자리

EC2가 계산을 위한 서버라면, EBS와 S3는 데이터를 두는 서로 다른 방식이다. Amazon EBS는 인스턴스에 붙는 블록 스토리지(block storage)로, 운영체제 디스크나 작업용 파일시스템처럼 "서버에 연결된 디스크"에 가깝다. 파일을

마운트해서 사용하고, 정렬 중간 파일이나 임시 작업 공간처럼 빠른 랜덤 I/O가 필요한 작업에 잘 맞는다. 반면 Amazon S3는 객체 스토리지(object storage)다. 폴더처럼 보이지만 실제로는 경로가 아니라 객체 키(key)를 가진 파일 저장소이며, 주로 URI 형태로 접근한다. FASTQ, BAM, CRAM, VCF, reference, 결과 파일, 공유 데이터셋처럼 오래 보관하고 여러 서비스가 함께 읽는 자산은 S3에 두는 편이 자연스럽다.

초보자에게는 EBS , S3 라는 비유가 꽤 유용하다. 작업대 위에는 지금 손으로 만지며 처리할 물건을 올려 두고, 호수에는 여러 사람이 함께 참고할 자료와 오래 보관할 자산을 둔다고 생각하면 된다. 이 둘 사이의 차이는 오믹스 분석에서 매우 실용적이다. 예를 들어 BAM 정렬 과정에서는 임시 파일과 sort 작업이 많으므로 EBS나 로컬 scratch가 중요하지만, 완성된 결과와 원본 데이터는 S3에 보관하는 편이 낫다. 즉 어디에 저장할지는 단순한 취향이 아니라, 접근 패턴과 워크플로 단계의 문제다.

FSx for Lustre는 이 둘 사이를 연결하는 특수한 선택지로 짧게 이해하면 좋다. 이는 POSIX 파일시스템이 필요하고, 여러 계산 노드가 빠르게 같은 파일 트리를 읽고 써야 할 때 유용한 고성능 공유 스토리지다. S3와 연동해 대규모 원본을 가져와 scratch처럼 처리하고 다시 결과를 S3로 돌려보내는 방식이 대표적이다. 모든 연구실이 처음부터 FSx를 배울 필요는 없지만, S3 가 있다는 사실을 아는 것은 중요하다. 특히 구조적 변이 분석이나 대규모 intermediate 파일을 다루는 워크플로에서는 이런 계층이 큰 차이를 만들 수 있다 (AWS 2024d).

Table 2. 유전체 데이터 유형별 저장 위치의 첫 선택

데이터 유형	우선 저장 위치	이유
원시 FASTQ	S3	장기 보관, 공유, 여러 서비스의 공통 입력
정렬 중간 파일	EBS 또는 로컬 scratch	높은 I/O와 짧은 수명
최종 BAM/CRAM, VCF	S3	재사용과 공유가 쉬움
운영체제와 도구 설치	EBS	인스턴스에 붙는 시스템 디스크
다수 노드가 함께 읽는 고성능 작업 데이터	FSx for Lustre	공유 파일시스템과 높은 처리량

S3 Standard, Intelligent-Tiering, Glacier 계층과 비용 감각

S3를 이해할 때 또 하나 중요한 것은 모든 객체가 같은 비용 구조를 갖지 않는다는 점이다. 가장 단순한 S3 Standard는 자주 읽고 쓰는 데이터에 적합한 기본 계층이다. 접근 패턴이 불규칙하거나 예측하기 어려운 경우에는 S3 Intelligent-Tiering이 유용할 수 있다. 반면 Standard-IA나 Glacier 계열은 저장 단가를 낮추는 대신, 읽어올 때의 retrieval fee나 최소 저장 기간(minimum storage duration), 복원 지연을 고려해야 한다. 따라서 초보자는 "가장 싼 클래스가 항상 가장 경제적이다"라는 생각부터 버리는 편이 좋다.

유전체 데이터는 생애주기(lifecycle)가 뚜렷하므로 storage class 사고방식이 특히 중요하다. 분석 직후의 FASTQ와 결과물은 재분석과 검증이 자주 일어나므로 hot 데이터에 가깝고, 이 단계에서는 S3 Standard가 자연스럽다. 몇 주나 몇 달이 지나 접근 빈도가 줄어들면 Intelligent-Tiering이나 Standard-IA가 적절할 수 있다. 장기 보관 의무가 있는 원시 데이터나 거의 꺼내지 않는 보관본은 Glacier Instant Retrieval, Glacier Flexible Retrieval, Glacier Deep Archive 같은 colder tier를 검토할 수 있다. 하지만 재분석이 잦은 데이터를 너무 빨리 Deep Archive로 밀어 넣으면 저장비는 줄어도 복원 지연과 retrieval 비용 때문에 프로젝트 전체 비용은 오히려 올라갈 수 있다 (AWS 2024b).

실전에서는 lifecycle rule을 통해 데이터를 단계적으로 이동시키는 패턴을 많이 쓴다. 예를 들어 업로드 직후에는 S3 Standard에 두고, 일정 기간이 지나면 Intelligent-Tiering이나 Standard-IA로 옮기고, 장기 보관 대상만 Glacier 계열로 내리는 식이다. 이 구조는 학생들에게 단순 암기보다 훨씬 유익하다. 중요한 것은 storage class 이름 자체가 아니라, 어떤 데이터가 아직 활발히 사용 중인지, 어떤 데이터가 보관 단계로 들어갔는지를 지속적으로 구분하는 습관이다. 비용 감각은 청구서를 본 뒤에 배우는 것이 아니라, 저장소를 설계할 때부터 함께 배워야 한다.

IAM Role과 최소 권한 원칙

IAM(Identity and Access Management)은 AWS에서 누가 무엇에 접근할 수 있는지를 결정하는 권한 체계다. 초보자는 보통 IAM User와 액세스 키부터 떠올리지만, 오믹스 분석 운영에서 더 중요한 개념은 IAM Role이다. IAM User는 사람이 로그인하거나 API를 호출할 때 사용하는 장기 자격 증명에 가깝고, IAM Role은 EC2, Batch, Lambda,

HealthOmics 같은 실행 주체에 임시 권한을 부여하는 방식이라고 이해하면 된다. 이 차이는 보안뿐 아니라 운영 편의성에서도 매우 크다. 액세스 키를 파일이나 스크립트에 박아 넣지 않고도, 인스턴스가 필요한 S3 버킷만 읽도록 안전하게 설정할 수 있기 때문이다.

최소 권한 원칙(least privilege)은 "필요한 일만 할 수 있도록 최소한의 권한만 준다"는 원칙이다. 예를 들어 어떤 EC2 인스턴스가 특정 입력 버킷을 읽고 결과 버킷에 쓰기만 하면 된다면, 전체 S3 전체 권한을 줄 이유가 없다. 이 원칙은 학생에게 추상적인 보안 문구로 설명하기보다, 실제 분석 흐름으로 보여 주는 편이 훨씬 잘 이해된다. `EC2 S3 Role` 는 한 문장을 중심으로 생각하면, 왜 액세스 키를 코드에 넣는 습관이 위험한지 바로 드러난다. 유전체 데이터는 민감 정보일 수 있기 때문에, 권한 관리의 기본기를 초반에 정확히 배우는 것이 이후 모든 장의 안전장치가 된다 (AWS 2024c; AWS 2024e).

초보자가 자주 하는 실수

입문자에게서 되풀이되는 실수는 크게 네 가지로 볼 수 있다. 첫째, EBS와 S3를 같은 종류의 저장소라고 생각하는 것이다. EBS는 서버에 붙는 디스크이고, S3는 여러 서비스가 공유하는 객체 저장소이므로, 둘을 혼동하면 데이터 배치와 비용 감각이 모두 흐려진다. 둘째, 인스턴스 크기만 보고 선택하는 것이다. 실제로는 메모리가 필요한 작업에 CPU형 인스턴스를 쓰거나, I/O 병목 작업에 큰 CPU만 늘리면 성능은 거의 오르지 않고 비용만 커진다.

셋째, 액세스 키를 설정 파일이나 노트북에 하드코딩하는 것이다. 처음에는 편해 보여도, 이 습관은 팀 협업과 장기 운영에서 큰 위험이 된다. IAM Role을 써야 할 자리에 장기 키를 넣어 두면 누가 어떤 권한을 언제 썼는지 통제하기 어려워지고, 비밀 유출 가능성도 커진다. 마지막으로 storage class를 무작정 가장 싼 쪽으로 선택하는 실수도 많다. 저장 단가만 보고 Glacier Deep Archive를 선택했다가, 나중에 데이터를 꺼내는 시간과 비용 때문에 연구 일정 전체가 꼬이는 일은 생각보다 흔하다.

EC2는 계산, EBS는 붙는 디스크, S3는 오래 두고 공유하는 객체 저장소, IAM Role은 실행 주체에 임시 권한을 주는 방식. 이렇게 네 축을 먼저 잡아 두면 충분하다. 그다음부터는 서비스 이름이 아니라 병목과 데이터 생애주기를 기준으로 선택하면 된다. 즉 좋은 AWS 운용은 화려한 서비스 조합보다, 적절한 계산 자원과 적절한 저장 위치와 적절한 권한을 고르는 기초에서 시작한다. 이 기본기가 잡혀 있어야 뒤에서 다룰 Batch, EMR, HealthOmics, Hail, Bedrock도 자연스럽게 이해할 수 있다.

핵심 개념 정리

- EC2는 계산 자원이고, 인스턴스 패밀리는 병목의 종류를 반영한다.
- EBS는 인스턴스에 붙는 블록 스토리지이고, S3는 장기 보관과 공유에 적합한 객체 스토리지다.
- FSx for Lustre는 여러 노드가 함께 쓰는 고성능 파일시스템이 필요할 때 고려하는 보조 계층이다.
- S3 storage class는 저장 단가만이 아니라 retrieval fee, minimum duration, 복원 지연까지 함께 봐야 한다.
- IAM Role은 액세스 키를 코드에 넣지 않고 실행 주체에 임시 권한을 주는 기본 보안 장치다.

복습 질문

1. `c` 계열과 `r` 계열 인스턴스는 각각 어떤 유전체 workload에 더 잘 맞는가?
2. 정렬 중간 파일과 최종 VCF를 같은 저장소에 두지 않는 이유는 무엇인가?
3. IAM User 대신 IAM Role을 우선 고려해야 하는 이유를 실제 EC2와 S3 예시로 설명해 보라.

Further Reading

- AWS Samples. IAM role creation example in `aws-genomics-workflows`.
- AWS. Using Amazon FSx for Lustre for genomics workflows on AWS.
- AWS. Secure your genomic workflows and data with AWS HealthOmics.

References

- AWS. 2024a. Amazon EC2 instance type names.
- AWS. 2024b. Amazon S3 storage classes overview.
- AWS. 2024c. IAM best practices.

- AWS. 2024d. Using Amazon FSx for Lustre for genomics workflows on AWS.
- AWS. 2024e. Secure your genomic workflows and data with AWS HealthOmics.

3장. 데이터는 옮기지 말고 연결하라 – gnomAD와 S3

학습 목표

- gnomAD가 S3 링크를 제공하는 이유를 설명할 수 있다.
- gnomAD의 Hail VDS가 왜 단순 압축 파일이 아니라 분석 친화적 데이터 표현인지 설명할 수 있다.
- 데이터 복사 중심 사고와 데이터 현지 분석(data-local analysis) 사고의 차이를 이해할 수 있다.
- 공개 유전체 데이터를 AWS에서 직접 참조하는 기본 전략을 설명할 수 있다.

핵심 질문

- 왜 예전에는 FTP나 Globus로 데이터를 옮겨야 했는가
- 왜 지금은 S3 링크를 연결하는 방식이 더 합리적인가
- gnomAD 같은 대규모 자원은 사용자에게 무엇을 바꾸게 했는가
- 왜 gnomAD의 VDS는 “큰 VCF 하나”와 다른 사고방식을 요구하는가

공개 데이터 배포 방식의 변화

대규모 유전체 공공 자원을 다루는 방식은 지난 몇 년 사이에 뚜렷하게 바뀌었다. 예전에는 연구자가 FTP나 Globus 같은 전송 경로를 통해 큰 파일을 자기 기관 서버로 내려받고, 그 복사본 위에서 필터링과 통계 분석을 수행하는 방식이 표준에 가까웠다. 이 방식은 데이터셋이 작고 갱신 주기가 느릴 때는 어느 정도 합리적이었다. 그러나 수십만 명에서 백만 명 규모의 코호트가 등장하면서 같은 데이터를 여러 기관이 반복해서 저장하고, 서로 다른 시점의 복사본을 유지하고, 매번 이동 비용을 감당하는 방식은 점점 비현실적이 되었다. 공개 데이터 자원이 클라우드 객체 스토리지에 놓이기 시작한 배경에는 바로 이 구조적 한계가 있다.

gnomAD는 이 변화를 설명하기에 가장 좋은 사례다. gnomAD 팀은 2020년 다중 클라우드 접근 공지에서, 데이터를 공개 클라우드에 두는 이유를 중복 복사 제거, 자원 절약, 다른 클라우드 데이터셋과의 통합, 그리고 더 넓은 접근성 확보라는 네 가지 관점에서 설명했다. 이 메시지는 단순히 다운로드 링크의 위치가 바뀌었다는 뜻이 아니다. 오히려 연구자가 대규모 참조 자원을 자기 서버로 가져오는 대신, 공용 객체 스토리지에 있는 자원을 직접 참조하고 그 근처에서 계산해야 한다는 새로운 분석 습관을 뜻한다. 이때 핵심 문장이 바로 `bring compute to data`이다. 즉 클라우드 시대의 유전체 분석은 파일을 먼저 옮기는 일보다, 어디에 있는 데이터를 어떤 방식으로 바로 읽을 것인가를 먼저 설계하는 일에 가깝다 (gnomAD team 2020; AWS Open Data Registry 2026).

gnomAD 데이터 포털과 S3 배포 구조

AWS Open Data Registry는 gnomAD를 공개 S3 자원으로 소개하고, 대표 버킷으로 `s3://gnomad-public-us-east-1/`를 안내한다. 초보자에게 중요한 점은 여기서 `S3 (bucket)`과 `(object path)`라는 개념을 함께 이해하는 것이다. 버킷은 데이터가 담기는 최상위 저장 공간이고, 개별 파일이나 테이블은 그 아래 객체 키를 통해 접근된다. 로컬 디스크의 폴더와 비슷하게 보일 수 있지만, 실제로는 POSIX 파일시스템이 아니라 객체 스토리지이므로 접근 방식과 비용 구조가 다르다. 오믹스 연구자 입장에서는 이것이 “어디에 있는 공개 자원을 어떻게 읽을 것인가”라는 실용적 질문으로 곧바로 연결된다.

gnomAD를 S3에서 제공한다는 사실은 두 가지 변화를 동시에 의미한다. 첫째, 사용자는 큰 참조 자원을 매번 개인 저장소에 복사하지 않아도 된다. 둘째, 같은 클라우드 안에서 EC2, EMR, Hail, Athena 같은 계산 계층이 공용 데이터를 직접 읽을 수 있으므로, 분석 출발점이 훨씬 일관되고 재현 가능해진다. 예를 들어 어떤 연구자가 같은 버전의 gnomAD release를 같은 S3 경로에서 읽는다면, 최소한 참조 자원의 위치와 버전에 대해서는 같은 출발선을 공유하게 된다. 따라서 S3 위의 gnomAD는 단순한 다운로드 저장소가 아니라, 협업용 공용 참조 레이어(shared reference layer)로 이해하는 편이 더 정확하다.

Table 1은 전통적 다운로드 방식과 S3 직접 참조 방식의 차이를 교육용으로 단순화한 것이다. 여기서 중요한 것은 “무조건 클라우드가 빠르다”가 아니라, 데이터 이동을 줄일수록 저장 중복과 운영 복잡성이 함께 줄어든다는 점이다. 특히 코호트가

커질수록 이동 비용과 정합성 관리 비용이 더 크게 작용하므로, 참조 자원은 가능한 한 공용 위치에 두고 계산을 옮기는 방식이 유리해진다. 이 점은 뒤에서 다룰 SRA, EMR, Hail, HealthOmics 장에도 반복해서 등장하는 공통 원리다.

Table 1. 공개 유전체 자원 접근 방식의 비교

항목	전통적 다운로드 방식	S3 직접 참조 방식
출발점	파일을 기관 서버로 먼저 복사	공용 객체 스토리지 경로를 직접 참조
저장 요구	기관별 중복 복사본이 쉽게 늘어남	공용 자원을 공유하고 로컬 복사를 줄임
갱신 관리	새 release가 나오면 다시 동기화 필요	버전된 경로를 기준으로 직접 참조 가능
협업	각자 다른 복사본을 쓸 가능성 큼	같은 경로와 같은 release를 공유하기 쉬움
대규모 코호트 적합성	이동과 저장 비용이 커짐	계산을 데이터 가까이에서 수행하기 쉬움

Hail VDS – 파일이 아니라 분산 데이터셋으로 보기

gnomAD를 이야기할 때 자주 등장하는 `VDS`는 여기서 주의 깊게 설명해야 하는 개념이다. 초보자는 VDS를 단순히 "VCF보다 더 잘 압축한 형식"으로 이해하기 쉽지만, 실제로는 그보다 훨씬 분석 지향적인 표현이다. Hail 문서에 따르면 VariantDataset(VDS)은 cohort-level genomic data를 표현하기 위한 희소(sparse), 분리(split) 구조이며, `reference_data`와 `variant_data`를 별도의 MatrixTable로 나누어 저장한다. 다시 말해 모든 샘플과 모든 위치를 빽빽하게 펼쳐 저장하는 방식이 아니라, 참조 블록과 실제 변이 정보를 나누어 저장함으로써 대규모 코호트에 더 잘 맞는 질의와 계산을 가능하게 한다. 따라서 VDS는 저장 형식인 동시에 계산 구조다 (Hail 2026).

이 차이는 gnomAD v4.0 release note에서 매우 인상적인 숫자로 드러난다. 2023년 11월 공개된 v4.0 공지에 따르면, gnomAD는 release용 730,947 exomes를 만들기 위해 실제로 955,213 samples 규모의 전체 데이터를 처리했으며, 이 전체 v4 dataset은 VDS 형식 덕분에 18 TB만 사용했다. 같은 데이터를 전통적 project VCF로 저장했다면 897 TB가 필요했을 것이라고 설명한다. 이 수치는 VDS가 단순한 파일 포맷 차이를 넘어, 대규모 코호트를 어떻게 표현할 것인가의 문제임을 잘 보여 준다. 즉 VDS는 저장 공간만 아끼는 기술이 아니라, 대규모 callset을 분석 가능한 구조로 유지하기 위한 데이터 모델이라고 보는 편이 더 정확하다 (gnomAD Production Team 2023).

또한 VDS는 무엇을 저장하느냐만 바꾸는 것이 아니라, 무엇을 질문할 수 있느냐도 바꾼다. gnomAD v4.1 공지는 VDS 덕분에 all callable sites에서 allele number를 유지할 수 있었고, 변이가 관찰되지 않은 위치까지 포함한 allele number 파일을 제공할 수 있었다고 설명한다. 이는 기존 callset 처리에서 변이가 없는 위치의 정보가 쉽게 사라지던 문제와 대비된다. 학생 입장에서는 이것을 "저장 효율이 좋아졌다"보다 "질문 가능한 범위가 넓어졌다"라고 이해하는 편이 훨씬 유익하다. 유전체 데이터 구조는 단순히 디스크 공간을 아끼는 문제가 아니라, 과학적 해석의 범위를 결정하는 문제이기 때문이다 (Chao et al. 2024).

공용 참조 레이어와 재현성

공용 클라우드 자원의 가장 큰 장점 가운데 하나는 재현성과 협업의 출발점을 맞추기 쉽다는 점이다. 전통적 방식에서는 같은 gnomAD release를 연구실마다 따로 보관하고, 어느 시점에 어떤 버전을 받았는지, 어떤 전처리를 추가했는지, 로컬에서 어떤 인덱스를 다시 만들었는지 추적하기 어려운 경우가 많았다. 반면 공용 객체 스토리지에 있는 versioned release를 직접 참조하면, 적어도 참조 자원의 위치와 버전에 대해서는 훨씬 명확하게 기록할 수 있다. 이후 동일한 Hail 버전, 동일한 container image, 동일한 workflow를 결합하면 분석 출발점이 더 단단해진다. 재현성이란 단지 논문의 마지막 단계에서 코드를 공개하는 일이 아니라, 분석에 들어가기 전부터 같은 공용 레이어를 공유하는 운영 습관이라는 점을 이 장에서 강조할 필요가 있다.

gnomAD가 2024년 8월 browser Hail Tables를 공개한 것도 같은 흐름으로 읽을 수 있다. 이 발표의 핵심은 웹 브라우저 뒤에서만 보이던 데이터 구조가, 이제 연구자가 재사용 가능한 Hail Table 자산으로 직접 접근 가능해졌다는 점이다. 학생들에게는 이것이 매우 중요한 전환이다. 브라우저에서 보는 요약 정보와 실제 분석 자원이 더 가까워질수록, "웹에서 본 내용을 다시 로컬에서 재구성해야 하는 불필요한 노동"이 줄어든다. 공개 참조 자원이 단순한 보기용 포털을 넘어서 분석 가능한 테이블과 라이브러리 형태로 확장될수록, 공용 참조 레이어라는 개념은 더 강해진다 (gnomAD browser 2024).

Hail, Spark, Athena, 도구 생태계와의 연결

gnomAD를 S3에서 읽는다고 해서 모든 분석이 자동으로 해결되는 것은 아니다. 중요한 것은 그 위에 어떤 질의 계층과 어떤 도구를 얹느냐이다. Hail은 대규모 희소 유전체 데이터를 분산 연산 환경에서 다루는 대표적 도구이고, Spark 기반 분석 계층과 함께 사용할 때 진가를 발휘한다. AWS 쪽에서는 Athena-ready lakehouse 자산이나 공개 테이블 자원이 추가되면서, 반드시 거대한 VCF를 모두 내려받지 않아도 특정 cohort filter, annotation query, 빈도 확인 같은 작업을 더 직접적으로 설계할 수 있게 되었다. 즉 같은 공용 S3 자원이라도 어떤 도구로 접근하느냐에 따라 사용 경험은 크게 달라진다.

2025년 1월 공개된 gnomAD toolbox는 이 흐름을 현재형으로 잘 보여 준다. 이 공지는 v4 exomes와 genomes short variant release VCF 전체가 약 1.4 TB이므로, 모든 사용자가 이를 통째로 내려받아 처리하는 방식은 현실적이지 않다고 지적한다. 대신 toolbox를 통해 로컬 복사 없이 gnomAD data를 더 쉽게 질의하도록 돕는 방향을 제시한다. 이 사례는 "클라우드에 올려 두었다"는 사실만으로 충분하지 않고, 그 위에 query-friendly 도구와 테이블이 있어야 실제 접근성이 높아진다는 점을 잘 보여 준다. 즉 gnomAD는 공개 다운로드 자원이면서 동시에 공용 분석 플랫폼의 일부이고, S3는 그 플랫폼의 기반 저장 계층으로 작동한다.

핵심 개념 정리

- gnomAD가 S3 링크를 제공하는 이유는 데이터를 더 많이 복사하라는 뜻이 아니라, 있는 곳에서 바로 읽으라는 뜻이다.
- 객체 스토리지 위의 공개 자원은 협업용 공용 참조 레이어로 기능할 수 있다.
- Hail VDS는 단순 압축 포맷이 아니라, 대규모 코호트를 희소하고 분산 분석 친화적으로 표현하는 구조다.
- 공개 데이터의 가치가 커질수록 `download -> analyze`보다 `query -> access in place -> compute` 패턴이 더 중요해진다.

복습 질문

1. gnomAD가 S3를 통해 데이터를 공개하는 이유를 저장, 협업, 계산 관점에서 설명해 보라.
2. VDS가 전통적 project VCF와 다른 점은 무엇이며, 왜 이것이 대규모 코호트 분석에 중요한가?
3. 공용 객체 스토리지 위의 참조 자원이 재현성과 협업을 어떻게 바꾸는가?

Further Reading

- AWS Open Data Registry. Genome Aggregation Database (gnomAD).
- gnomAD browser. gnomAD v4.0.
- gnomAD browser. gnomAD toolbox.

References

- AWS Open Data Registry. 2026. Genome Aggregation Database (gnomAD).
- Chao K, Wilson M, Goodrich J, gnomAD Production Team. 2024. gnomAD v4.1.
- gnomAD browser. 2024. Release gnomAD browser tables.
- gnomAD browser. 2025. gnomAD toolbox.
- gnomAD browser. 2026. gnomAD v4.1.1.
- gnomAD Production Team. 2023. gnomAD v4.0.
- gnomAD team. 2020. Open access to gnomAD data on multiple cloud providers.
- Hail. 2026. VariantDataset.

4장. 공용 오믹스 데이터를 AWS로 가져오기 – SRA와 Open Data

학습 목표

- SRA 데이터를 AWS에서 다루는 기본 흐름을 설명할 수 있다.
- AWS Open Data 프로그램이 연구자에게 주는 의미를 이해할 수 있다.

- 공개 데이터 반입과 직접 참조의 차이를 구분할 수 있다.

핵심 질문

- SRA 데이터는 AWS에서 어떻게 접근하는가
- 모든 공개 데이터가 S3에 있는 것은 아닌데, 어떤 전략으로 가져와야 하는가
- AWS Open Data는 단순 저장소가 아니라 어떤 생태계를 만드는가

"다운로드"에서 "직접 접근"으로 바뀐 패러다임

공용 유전체 데이터를 AWS에서 다룬다고 할 때 초보자가 가장 먼저 바로잡아야 할 표현은 라는 말이다. 특히 NCBI SRA(Sequence Read Archive)의 경우, AWS에서 데이터를 쓴다는 것은 많은 경우 NCBI 서버에서 무엇인가를 새로 요청해 받는 일이 아니라, 이미 S3에 공개되어 있는 데이터를 직접 접근(access)하는 일에 가깝다. NCBI의 `SRA in the Cloud`와 `Download SRA sequence data using AWS` 문서는 이 점을 분명하게 설명한다. 즉 public SRA data는 AWS Registry of Open Data를 통해 S3와 HTTPS 경로로 제공되며, 연구자는 이를 EC2나 다른 AWS 계산 계층에서 바로 읽을 수 있다. 이 구조가 의미하는 바는 크다. 유전체 데이터 접근의 출발점이 `download -> analyze`에서 `query -> access in place -> compute in cloud`로 이동하고 있기 때문이다 (NCBI 2024a; NCBI 2024b).

이 변화는 단순한 편의성 이상의 의미를 가진다. 예전에는 원하는 run accession을 찾은 다음, 이를 FTP나 다른 경로로 길게 내려받고, 로컬 스토리지에 저장한 뒤, 다시 분석 환경으로 옮기는 단계가 필수에 가까웠다. 그러나 공용 데이터가 이미 S3 안에 있다면, 같은 클라우드 안의 계산 자원이 이를 직접 읽어 올 수 있다. 따라서 병목은 전송 자체보다 메타데이터를 어떻게 검색하고, 어떤 accession만 선택하고, 어떤 형식으로 바로 후속 파이프라인에 연결할 것인가로 이동한다. 이 장에서 SRA는 단순한 다운로드 실습이 아니라, 공용 유전체 데이터 접근 패러다임이 어떻게 바뀌고 있는지를 보여 주는 대표 사례로 읽어야 한다.

S3 direct access와 `--no-sign-request`

SRA 데이터를 AWS에서 다루는 가장 클라우드 네이티브한 방식은 direct S3 access이다. NCBI 문서는 `s3://sra-pub-run-odp/`, `s3://sra-pub-src-2/`, `s3://sra-pub-sars-cov2/` 같은 공개 버킷을 예시로 들며, `aws s3 ls s3://sra-pub-run-odp/ --no-sign-request`와 같은 명령으로 일부 데이터를 계정 없이도 조회할 수 있다고 설명한다. 이 예시는 교육적으로 매우 중요하다. 많은 초보자가 "클라우드 접근은 복잡한 인증과 비용이 따라붙는다"고 생각하지만, 공개 SRA 자산의 상당 부분은 적어도 탐색 단계에서 매우 직접적으로 접근할 수 있기 때문이다. 즉 공용 데이터에 대해 AWS는 때로 로그인한 조직 사용자만의 환경이 아니라, 공개 연구 인프라의 일부로 기능한다.

직접 접근 방식의 장점은 데이터를 다른 저장소로 다시 옮기지 않아도 된다는 데 있다. 예를 들어 EC2, AWS Batch, EMR, HealthOmics 같은 계산 계층이 같은 클라우드 안에서 public SRA bucket을 바로 읽는다면, 중간 복사본과 불필요한 저장 비용을 줄일 수 있다. NCBI 문서는 S3 URL을 통한 접근이 자유롭고, S3 URL의 경우 inter-region data transfer fee가 없다고 설명한다. 물론 사용자가 자기 결과를 저장하거나 별도 계산 자원을 운영하는 비용은 여전히 발생할 수 있다. 하지만 핵심은 공용 원본에 접근하는 출발점이 훨씬 가벼워졌다는 점이다 (NCBI 2024b).

SRA Toolkit – format conversion과 표준 접근 경로

그렇다고 해서 direct S3 access가 항상 유일한 길은 아니다. `sra-tools`는 여전히 SRA 생태계에서 매우 중요한 표준 도구이며, 특히 `prefetch`, `fastq-dump`, `fasterq-dump` 같은 유틸리티는 많은 워크플로에서 계속 사용된다. direct S3 access가 "공용 객체에 바로 닿는 경로"라면, SRA Toolkit은 "SRA 포맷을 사용자가 원하는 형식으로 가져오고 변환하는 경로"라고 이해할 수 있다. 따라서 이 장에서는 둘을 경쟁 관계로 설명하기보다, 서로 다른 목적의 도구로 설명하는 편이 더 정확하다. 즉 이미 알고 있는 run accession을 빠르게 cloud-native하게 다루고 싶다면 direct S3 access가 더 자연스러울 수 있고, SRA 고유 포맷을 표준 도구로 처리하거나 특정 변환 단계를 거치고 싶다면 Toolkit이 적합할 수 있다.

실제 교육에서는 이 두 경로를 대비해 보여 주는 것이 좋다. 같은 accession을 대상으로 할 때 `aws s3 cp`는 빠른 직접 복사 경로를 보여 주고, `prefetch`와 `fasterq-dump`는 표준화된 포맷 변환 경로를 보여 준다. 학생은 여기서 "어떤 도구가 더 정답인가"를 외우기보다, 어떤 상황에서 어떤 접근이 더 자연스러운가를 배워야 한다. SRA Toolkit의 존재는

클라우드 시대에도 파일 형식과 변환 단계가 여전히 중요하다는 사실을 상기시킨다. 즉 클라우드는 데이터 이동을 줄여 주지만, 형식과 해석의 문제를 사라지게 하지는 않는다.

Athena metadata query와 cohort selection

공용 데이터가 커질수록 더 중요한 것은 원본 파일을 만지는 일보다 먼저 메타데이터를 좁히는 일이다. NCBI는 AWS Athena를 통한 cloud-native metadata search를 공식적으로 안내하고 있으며, `Get Started in Athena`와 예제 쿼리 문서는 SRA metadata를 SQL로 검색하는 흐름을 제공한다. 이 구조는 교육적으로 매우 강력하다. 학생은 먼저 organism, assay, platform, project, accession 범위를 Athena로 좁히고, 그 다음 필요한 run만 direct access나 Toolkit으로 넘기는 흐름을 배울 수 있다. 즉 대규모 공개 데이터 시대의 첫 단계는 파일 다운로드가 아니라 metadata query다 (NCBI 2024c).

이 점은 연구 설계에도 직접 연결된다. 예전에는 관심 있는 데이터셋을 대충 골라 먼저 내려받은 뒤, 로컬에서 정리하는 경우가 많았다. 반면 Athena 기반 접근은 어떤 cohort를 만들고 싶은지, 어떤 속성으로 샘플을 필터링할 것인지, 어떤 조건을 만족하는 run만 계산에 태울 것인지부터 명시하게 만든다. 이는 단순한 기술 선택이 아니라 분석 사고방식의 변화다. Table 1처럼 `-> accession ->` `->` 는 흐름을 체화하면, 학생은 더 큰 공개 데이터 환경에서도 같은 원리로 확장할 수 있다.

Table 1. SRA on AWS의 세 가지 기본 접근 경로

경로	언제 적합한가	대표 예시
Public direct access	accession을 이미 알고 있고, public data를 바로 읽고 싶을 때	`aws s3 ls s3://sra-pub-run-odp/ --no-sign-request`
Toolkit-based access	SRA 포맷 변환이나 표준 도구 흐름이 필요할 때	`prefetch`, `fasterq-dump`
Cloud Data Delivery	Toolkit으로 직접 받을 수 없는 original file 또는 특정 restricted data가 필요할 때	사용자 S3 bucket으로 직접 전달

Cloud Data Delivery – original file과 restricted data 전달

사용자가 흔히 떠올리는 `request` 개념에 가장 가까운 것은 Cloud Data Delivery Service다. NCBI 공식 문서는 SRA Toolkit이 모든 original submitted file을 직접 제공할 수는 없기 때문에, cloud data delivery를 통해 source file이나 기타 특정 파일을 사용자 bucket으로 전달한다고 설명한다. 여기서 중요한 것은 이것이 일반적인 의미의 "내 컴퓨터로 다운로드"가 아니라는 점이다. 데이터는 사용자의 AWS 또는 GCP bucket으로 직접 전달되며, AWS의 경우 목적 bucket이 `us-east-1` 리전에 있어야 한다는 제약이 있다. 따라서 학생은 public data의 direct access와, 별도의 delivery 요청이 필요한 경우를 명확히 구분해 배워야 한다 (NCBI 2024d).

이 distinction은 운영상 매우 중요하다. public data는 공용 bucket을 바로 읽는 방식이 자연스럽고, restricted data나 original file 일부는 사용자의 cloud bucket으로 controlled delivery하는 방식이 더 적합하다. 즉 모든 것을 같은 방식으로 처리하지 않는다는 점이 핵심이다. 실제 연구에서는 공개 데이터와 승인 기반 데이터가 함께 등장할 수 있으므로, 어떤 자산이 direct access 대상이고 어떤 자산이 delivery 대상인지를 구분하는 능력이 필요하다. 이 장에서 SRA는 단지 데이터 접근 기술이 아니라, 데이터 거버넌스와 접근권한이 클라우드 안에서 어떻게 구현되는지를 배우는 출발점이 된다.

DataSync와 하이브리드 데이터 반입

SRA 같은 공개 데이터 자원은 직접 참조가 중요하지만, 우리 연구실이 직접 생산한 데이터는 여전히 반입해야 하는 경우가 많다. 시퀀싱 센터, 병원, 온프레미스 HPC, 다른 클라우드에 있는 FASTQ, BAM, CRAM을 AWS로 옮겨야 할 때는 AWS DataSync가 매우 실용적인 도구가 된다. DataSync는 NFS, SMB, HDFS, object storage와 AWS 스토리지 사이의 대용량 전송, 무결성 검증, 자동화된 동기화를 지원한다. 따라서 이 장에서는 , 는 대비를 분명히 해 두는 것이 좋다. 이 둘을 같은 문제로 보지 않을수록 전체 데이터 전략이 더 명확해진다.

DataSync를 함께 소개하는 이유는 학생이 클라우드 활용을 공개 데이터 읽기에만 한정해서 생각하지 않게 하기 위해서다. 실제 연구 운영에서는 public open data, controlled data, 자체 생성 데이터가 함께 존재한다. SRA가 보여 주는 것은 공용 데이터의 direct access 모델이고, DataSync가 보여 주는 것은 하이브리드 환경에서 자체 데이터를 반입하는 운영 모델이다. 결국 AWS를 잘 쓴다는 말은 모든 데이터를 무조건 복사하는 것도 아니고, 모든 데이터를 무조건 현지 참조하는 것도 아니다. 데이터의 성격에 따라 , , 을 구분하는 것이 핵심이다.

비용과 운영 주의점

공용 데이터 접근이 쉬워졌다고 해서 비용을 완전히 잊어도 되는 것은 아니다. NCBI 문서는 public SRA data의 자유 접근과 무료 access를 강조하지만, 사용자가 EC2를 운영하거나 Athena query를 돌리거나 결과를 자기 버킷에 저장하는 비용은 여전히 발생할 수 있다. 또한 AWS의 일반적인 S3 비용 모델에는 Requester Pays 같은 개념도 존재하므로, 학생은 "공용 데이터 자유 접근"과 "모든 S3 bucket이 동일한 비용 규칙을 갖는다"를 혼동하지 않도록 배워야 한다. 특히 Athena는 메타데이터 탐색에 매우 강력하지만, 쿼리 결과 저장과 스캔 비용을 고려해야 하므로 무제한 무료 도구처럼 가르치면 안 된다. 비용 감각은 direct access의 장점을 과장하지 않고, 어떤 단계에서 누구의 비용이 발생하는지를 구분하는 태도에서 출발한다.

현대 유전체학에서 공용 데이터 접근은 더 이상 "먼저 받아서 저장"이 아니라, "먼저 찾고, 필요한 것만 정하고, 가능한 한 데이터가 있는 곳에서 계산"하는 방향으로 바뀌고 있다. SRA는 이 전환을 가장 잘 보여 주는 사례다. direct S3 access, Toolkit-based conversion, Cloud Data Delivery, DataSync 기반 반입은 서로 대체 관계가 아니라, 서로 다른 종류의 데이터와 다른 운영 문제를 푸는 경로들이다. 학생이 이 네 가지를 구분할 수 있게 되면, 이후 어떤 공개 유전체 자원을 AWS에서 보더라도 같은 원리로 접근 전략을 설계할 수 있게 된다.

핵심 개념 정리

- SRA를 AWS에서 다룬다는 것은 종종 보다 에 가깝다.
- public data, toolkit conversion, cloud delivery는 서로 다른 목적의 접근 경로다.
- 대규모 공개 데이터 시대에는 원본 파일보다 메타데이터를 먼저 질의하는 습관이 중요하다.
- 공개 데이터 직접 참조와 자체 데이터 반입은 다른 문제이며, DataSync는 후자에 더 가깝다.

복습 질문

1. public SRA data를 AWS에서 다룰 때 `download`보다 `direct access`라는 표현이 더 정확한 이유는 무엇인가?
2. direct S3 access와 SRA Toolkit은 각각 어떤 상황에서 더 적합한가?
3. Cloud Data Delivery와 DataSync는 어떤 점에서 서로 다른 문제를 해결하는가?

Further Reading

- NCBI. SRA in the Cloud.
- NCBI. Download SRA sequence data using Amazon Web Services (AWS).
- AWS Open Data Registry. NIH NCBI Sequence Read Archive (SRA) on AWS.

References

- Amazon Science. 2020. AWS democratizes access to the largest genomic sequences repository — NIH's Sequence Read Archive.
- AWS Open Data Registry. 2026. NIH NCBI Sequence Read Archive (SRA) on AWS.
- NCBI. 2024a. SRA in the Cloud.
- NCBI. 2024b. Download SRA sequence data using Amazon Web Services (AWS).
- NCBI. 2024c. Get Started in Athena.
- NCBI. 2024d. Cloud Data Delivery Service.
- NCBI. 2024e. Search in Athena.

5장. EC2에서 시작하는 실전 분석 환경

학습 목표

- 오믹스 분석용 EC2 인스턴스를 선택하는 기본 기준을 설명할 수 있다.
- 스토리지와 네트워크를 고려한 실전 환경 구성을 이해할 수 있다.
- SSH, 키 페어, 보안 그룹, 스크래치 디스크 운용의 기본을 설명할 수 있다.

핵심 질문

- 작은 실습과 대형 WGS 분석은 어떤 인스턴스가 다른가
- 임시 작업 디스크와 영구 저장 공간은 어떻게 나누는가
- 분석이 끝난 뒤 어떤 자원은 끄고 어떤 자원은 남겨야 하는가

첫 번째 분석용 EC2 인스턴스 만들기

EC2를 처음 다룰 때 초보자가 가장 자주 하는 실수는 "일단 큰 서버를 하나 띄워 보자"라고 생각하는 것이다. 그러나 실전에서는 인스턴스 크기보다 먼저 분석의 모양을 정리해야 한다. 지금 하려는 일이 유전체 정렬처럼 CPU를 오래 잡아먹는 작업인지, Hail이나 대형 annotation join처럼 메모리를 많이 쓰는 작업인지, 아니면 Jupyter와 소규모 QC처럼 균형형 자원으로 충분한 작업인지에 따라 출발점이 달라지기 때문이다. 따라서 첫 번째 EC2 인스턴스를 만든다는 말은 콘솔에서 버튼을 누르는 일이 아니라, , , , 돌릴지를 먼저 결정하는 일에 가깝다.

오믹스 분석에서는 데이터와 계산을 같은 리전에 두는 것이 특히 중요하다. S3 버킷이 `ap-northeast-2`에 있는데 EC2는 다른 리전에 띄우면, 지연 시간과 데이터 이동 비용이 늘어나고 운영도 복잡해진다. 따라서 실습 단계부터 는 원칙을 습관으로 만드는 편이 좋다. 첫 환경은 대개 다음과 같은 조합으로 충분하다. 운영체제는 범용 Linux AMI, 인스턴스는 `m` 또는 `c` 계열의 중간 크기, 루트 디스크와 작업용 EBS, 그리고 S3에 접근할 수 있는 IAM Role이다. 처음부터 화려한 구성을 만들기보다, 작게 시작해서 병목을 확인한 뒤 자원을 키우는 편이 훨씬 안전하다.

또 하나 중요한 점은 EC2를 "서버 한 대"로만 이해하지 않는 것이다. EC2 인스턴스는 계산 그 자체이고, 그 주변에는 키 페어, 보안 그룹, IAM Role, EBS 볼륨, CloudWatch 로그, S3 경로가 함께 움직인다. 즉 실전 분석 환경은 인스턴스 단품이 아니라, 계산·저장·권한·접속이 묶인 실행 단위다. 이 관점을 초기에 이해하면 나중에 Batch, ParallelCluster, HealthOmics 같은 상위 계층을 배우더라도 훨씬 자연스럽다.

인스턴스 패밀리 읽기 – `m`, `c`, `r`, `i`, `f`, `hpc`

Amazon EC2 인스턴스 이름은 복잡해 보이지만, 실은 를 드러내는 약속어에 가깝다. AWS 공식 문서는 인스턴스를 범용형, 컴퓨트 최적화형, 메모리 최적화형, 스토리지 최적화형, 가속형, HPC형으로 나눈다. 오믹스 분석자는 모든 세부 옵션을 외울 필요는 없지만, 적어도 `m`, `c`, `r`, `i`, `f`, `hpc`가 각각 어떤 종류의 병목을 겨냥하는지는 읽을 수 있어야 한다 (AWS 2026a).

가장 무난한 시작점은 `m` 계열이다. 노트북 서버, 소규모 샘플 테스트, 경량 QC, 교육용 실습처럼 CPU와 메모리 요구가 한쪽으로 심하게 기울지 않은 작업에 적합하다. 정렬, 단순 variant calling, batch QC, permutation test처럼 코어 수가 늘수록 비교적 곧바로 처리 시간이 줄어드는 작업은 `c` 계열이 더 잘 맞는다. 반대로 cohort aggregation, joint genotyping, 대형 annotation join, Hail 집계처럼 메모리가 먼저 바닥나는 단계는 `r` 계열을 우선 고려하는 편이 좋다. 즉 인스턴스 선택은 "더 비싼 서버를 사는 일"이 아니라, 병목이 CPU인지 메모리인지 I/O인지 판단하는 일이다.

`i` 계열과 `d` 계열처럼 로컬 스토리지 성능을 강조하는 인스턴스는 scratch-heavy workflow에서 강점을 보인다. 대용량 정렬 중간 파일, 반복적인 sort, shard 기반 분할 작업, 대형 temporary matrix 변환 같은 단계는 루트 디스크만으로 처리할 때보다 훨씬 안정적일 수 있다. 다만 여기서 중요한 점은 로컬 scratch가 곧 영구 저장소가 아니라는 사실이다. 빠르다는 이유로 모든 산물을 여기에 두면, 인스턴스 종료와 함께 결과를 잃거나 정리되지 않은 중간 파일만 쌓일 수 있다.

가속형 인스턴스는 더 조심해서 이해해야 한다. `f` 계열은 FPGA 같은 가속기를 활용하는 인스턴스인데, "가속기가 있으니 무조건 빠르다"는 뜻이 아니다. DRAGEN처럼 FPGA를 활용하도록 설계된 파이프라인은 최근 AWS HPC 블로그의 F2 평가 사례에서 매우 좋은 성능과 비용 효율을 보였지만, 일반적인 스크립트나 대부분의 오픈소스 도구는 자동으로 이 이득을 얻지 못한다 (AWS 2026f). `hpc` 계열 역시 모든 NGS 파이프라인의 기본 선택지가 아니라, EFA와 긴밀한 노드

간 통신이 중요한 계산이나 특수한 분산 알고리즘에 더 가깝다. 대부분의 연구실은 `m` 또는 `c`에서 시작해 실제 병목을 본 뒤 `r`, `i`, `f`, `hpc`로 이동하는 편이 더 현실적이다.

같은 패밀리 안에서도 세대와 접미사에 따라 실제 성능과 가격 구조가 달라진다. 예를 들어 `c5`는 Xeon 기반 5세대, `c6i`는 Ice Lake 기반 6세대, `c7g`는 Graviton3 기반 7세대, `c8gd`는 Graviton4 기반에 로컬 NVMe SSD가 포함된 최신 세대를 뜻한다. 표준 bwa/GATK 실행에는 `c6i.8xlarge`나 `c7g.16xlarge`가 무난하고, 정렬 sort와 shard 중심 작업에는 `c7gd.8xlarge`나 `c8gd.16xlarge`처럼 로컬 NVMe가 붙은 변형이 실질적인 차이를 만든다. 인스턴스 이름을 읽는 규칙과 구체적인 세대별 예시는 2장에서 자세히 다루므로, 실제 인스턴스 이름을 고를 때 그 표를 참고하면 된다.

Table 1. 오믹스 분석에서의 EC2 인스턴스 패밀리 첫 선택

인스턴스 패밀리	주된 병목	대표적인 유전체 작업
`m`	균형형	Jupyter, 소규모 QC, 교육용 분석 서버, 경량 파이프라인
`c`	CPU	정렬, 일반 variant calling, batch QC, permutation test
`r`	메모리	joint genotyping, cohort aggregation, Hail join, 대형 annotation
`i` / `d`	로컬 I/O	sort, shard 처리, scratch 중심 intermediate workflow
`f`	가속기	DRAGEN 같은 FPGA 활용 파이프라인
`hpc`	노드 간 통신	MPI형 계산, 특수한 tightly coupled 병렬 작업

보안 그룹, 키 페어, IAM Role 연결

초보자가 EC2를 어렵게 느끼는 이유 중 하나는 계산 자원보다 오히려 접속과 권한 구조가 낯설기 때문이다. 하지만 이를 세 부분으로 나누면 생각보다 단순하다. 키 페어는 서버에 SSH로 접속하기 위한 신분증이고, 보안 그룹은 어떤 네트워크 트래픽을 허용할지 정하는 방화벽이며, IAM Role은 인스턴스가 AWS 자원에 접근할 때 쓰는 권한 묶음이다. 세 요소는 서로 다른 문제를 해결한다. 키 페어가 "누가 서버 안으로 들어오느냐"를 정한다면, 보안 그룹은 "어떤 문을 열어 둘 것이냐"를 정하고, IAM Role은 "서버가 밖에 나가 무엇을 할 수 있느냐"를 정한다.

보안 그룹은 넓게 열어 둘수록 편해 보이지만, 실전에서는 최소한으로 열어 두는 습관이 중요하다. 예를 들어 개인 실습 서버라면 SSH 포트만 허용하고, 가능하면 특정 IP 범위로 제한하는 편이 낫다. 웹 기반 노트북이나 서비스가 필요할 때도 공개 인터넷 전체에 열기보다, 프록시나 제한된 접근 경로를 먼저 고려해야 한다. 유전체 데이터는 민감 정보를 포함할 수 있기 때문에, "어차피 테스트니까 대충 열어 둔다"는 습관은 나중에 더 큰 문제로 이어진다.

IAM Role은 더욱 중요하다. 여전히 많은 초보자가 액세스 키를 설정 파일이나 노트북에 직접 넣는 방식을 먼저 떠올리지만, AWS의 기본 권장 방식은 실행 주체에 Role을 붙여 임시 권한을 부여하는 것이다. 예를 들어 어떤 인스턴스가 `s3://my-input-bucket/`을 읽고 `s3://my-output-bucket/`에 결과를 쓰기만 하면 된다면, 그 범위만 허용하는 Role을 붙이는 것이 맞다. 이렇게 하면 장기 자격 증명을 파일에 보관할 필요가 없고, 누가 어떤 권한으로 작업했는지도 더 명확해진다 (AWS 2026c). 오믹스 분석에서 Role을 초반부터 정확히 익히는 것은 보안 문제를 넘어서 재현성과 운영 안정성을 위한 기본기다.

Table 2. EC2 실전 운영의 세 가지 기본 보안 구성요소

구성요소	역할	초보자가 자주 하는 실수
키 페어	SSH 접속용 인증 수단	키 파일을 여러 사람과 무분별하게 공유함
보안 그룹	인바운드/아웃바운드 네트워크 제어	SSH나 애플리케이션 포트를 전체 인터넷에 과도하게 개방함

구성요소	역할	초보자가 자주 하는 실수
IAM Role	인스턴스의 AWS 자원 접근 권한	액세스 키를 파일이나 코드에 하드코딩함

EBS 볼륨과 임시 스크래치 공간 운용

실전 분석 환경에서 저장소를 설계할 때 가장 중요한 질문은 “무엇을 오래 보관할 것인가”와 “무엇을 빨리 처리하고 버릴 것인가”를 구분하는 일이다. EBS는 EC2에 붙는 블록 스토리지이므로 운영체제 디스크, 분석 도구 설치, 작업 중 생성되는 중간 파일에 적합하다. 반면 장기 보관과 공유가 필요한 FASTQ, BAM, CRAM, VCF, 보고서, reference 자산은 S3에 두는 편이 자연스럽다. 초보자는 모든 데이터를 인스턴스 안에 넣고 시작하기 쉽지만, 그렇게 하면 서버가 곧 저장소가 되어 버리고, 분석이 끝났을 때 무엇을 지우고 남길지 판단하기 어려워진다.

실전에서는 최소한 세 층으로 나누어 생각하면 훨씬 단순해진다. 첫째, 루트 볼륨은 운영체제와 기본 도구를 위한 공간이다. 둘째, 별도 EBS 또는 로컬 NVMe scratch는 정렬·sort·temporary shard처럼 I/O가 많은 작업을 위한 작업대다. 셋째, S3는 원본과 결과를 남기는 영구 계층이다. 이 구조를 이해하면 “왜 인스턴스를 종료하기 전에 결과를 S3로 밀어야 하는가”가 자연스럽게 연결된다. 빠른 scratch는 계산을 위한 공간이지, 연구 자산을 영구 보관하는 장소가 아니다.

EBS의 볼륨 타입도 무작정 크게만 잡으면 되는 문제는 아니다. 일반적인 부팅 디스크나 보통의 분석 작업은 범용 SSD 기반 볼륨으로 충분한 경우가 많지만, IOPS와 처리량이 뚜렷한 병목인 workflow에서는 더 높은 성능의 볼륨 구성이 필요할 수 있다 (AWS 2026b). 그러나 교육 단계에서 더 중요한 것은 볼륨 타입의 세부 옵션보다 데이터 배치 원칙을 정확히 익히는 일이다. 즉 “원본은 S3, 실행 중 작업은 EBS 또는 scratch, 종료 전 결과는 S3”라는 기본 구조가 먼저다.

Table 3. 오믹스 분석에서의 데이터 배치 원칙

데이터 종류	우선 위치	이유
원시 FASTQ, 최종 BAM/CRAM, VCF	S3	장기 보관, 공유, 재사용에 유리함
정렬 중간 파일, sort 임시 파일	EBS 또는 로컬 scratch	높은 I/O와 짧은 수명에 적합함
운영체제, 패키지, 컨테이너 캐시	루트 EBS	인스턴스 부팅 및 실행 환경에 필요함
최종 리포트 사본, 로그 아카이브	S3	인스턴스 생명주기와 분리해 보존 가능함

비용을 아끼는 습관

EC2 비용 절감은 특별한 비법보다 운영 습관에서 갈리는 경우가 많다. 첫 번째 습관은 `(stop)`와 `(terminate)`를 구분하는 것이다. 인스턴스를 중지하면 계산은 멈추지만 EBS 볼륨은 그대로 남아 비용이 계속 발생한다. 따라서 다음 날 바로 이어서 작업할 서버라면 중지가 유용하지만, 분석이 끝났고 같은 환경을 다시 띄울 수 있다면 종료가 더 적절하다. 반대로 종료를 습관처럼 눌렀다가 scratch에만 있던 결과를 잃는 경우도 많다. 결국 비용 관리와 데이터 관리가 같은 문제라는 뜻이다.

두 번째 습관은 Spot Instance를 “싸지만 불안정한 자원”으로 정확히 이해하는 것이다. Spot은 온디맨드보다 훨씬 저렴할 수 있지만, 언제든 회수될 수 있으며 일반적으로 2분 interruption notice가 제공된다 (AWS 2026d; AWS 2026e). 따라서 긴 대화형 분석 서버나 중간 상태를 자주 저장하지 않는 작업에는 부적합할 수 있다. 반면 정렬 batch, permutation, Monte Carlo, 샘플 단위 QC처럼 중단에 견딜 수 있는 작업에는 매우 강력하다. 초보자는 보통 “싼 것이면 무조건 좋다”거나 “불안정하니 절대 쓰면 안 된다”는 두 극단으로 가는데, 실전에서는 작업 성격에 따라 나누어 쓰는 것이 핵심이다.

세 번째 습관은 눈에 보이지 않는 잔여 자원을 정리하는 것이다. 인스턴스를 껐는데도 EBS 스냅샷, 사용하지 않는 볼륨, 오래된 AMI, 불필요한 Elastic IP, 복사만 해 둔 대형 intermediate가 남아 비용을 만들 수 있다. 특히 오믹스 분석은 파일 크기가 크기 때문에 한 번의 정리 소홀함이 곧 큰 저장 비용으로 이어진다. 그래서 좋은 실전 규칙은 간단하다. 인스턴스 시작 전에는 S3 를 정하고, 실행 중에는 를 정하며, 종료 전에는 를 점검하는 것이다.

Table 4. `stop`과 `terminate`의 실전 구분

선택	언제 적합한가	주의점
`stop`	내일 다시 접속해 같은 서버를 이어서 써야 할 때	EBS 비용은 계속 발생하고, 임시 scratch 전략이 흐려질 수 있음
`terminate`	분석이 끝났고 환경을 다시 만들 수 있을 때	인스턴스 내부에만 있던 파일은 사라지므로 먼저 S3로 내보내야 함

EC2를 잘 쓰는 사람은 가장 큰 인스턴스를 고르는 사람이 아니라, 병목에 맞는 인스턴스를 고르고, 저장 계층을 분리하고, 권한을 Role로 관리하고, 끝난 자원을 정리할 줄 아는 사람이다. 즉 실전 AWS 운용의 출발점은 화려한 서비스 조합이 아니라, , , , 를 분리해 생각하는 습관이다. 이 기본기가 있어야 다음 장에서 다룰 자동화와 대규모 오케스트레이션도 비용과 재현성을 모두 잡는 방향으로 이어질 수 있다.

핵심 개념 정리

- EC2 인스턴스 선택은 “가장 큰 서버”를 고르는 일이 아니라 CPU, 메모리, I/O 병목을 읽는 일이다.
- 실전 분석 환경은 인스턴스 단품이 아니라 키 페어, 보안 그룹, IAM Role, EBS, S3가 함께 움직이는 실행 단위다.
- 원본과 최종 산물은 S3에, 실행 중 작업 파일은 EBS나 scratch에 두는 분리가 중요하다.
- `stop`과 `terminate`, On-Demand와 Spot의 차이를 이해해야 비용과 안정성을 함께 관리할 수 있다.

복습 질문

1. `m`, `c`, `r`, `i` 계열은 각각 어떤 병목을 겨냥하며, 오믹스 분석에서는 어떤 작업에 잘 맞는가?
2. IAM Role을 붙인 EC2와 액세스 키를 파일에 저장한 EC2는 운영과 보안 측면에서 어떻게 다른가?
3. 인스턴스를 종료하기 전에 어떤 파일을 S3로 옮겨야 하는지, 그 이유를 설명해 보라.

Further Reading

- AWS. Amazon EC2 instance type naming conventions.
- AWS. Amazon EBS volume types.
- AWS. Best practices for Amazon EC2 Spot.

References

- AWS. 2026a. Amazon EC2 instance type naming conventions.
- AWS. 2026b. Amazon EBS volume types.
- AWS. 2026c. IAM best practices.
- AWS. 2026d. Best practices for Amazon EC2 Spot.
- AWS. 2026e. Spot Instance interruption notices.
- AWS. 2026f. Evaluating next-generation cloud compute for large-scale genomic processing.

6장. 대규모 작업 자동화 – Ansible, AWS Batch, Lambda

학습 목표

- 반복 작업 자동화가 왜 중요한지 설명할 수 있다.
- Ansible, AWS Batch, Lambda가 각각 어떤 종류의 문제에 적합한지 구분할 수 있다.
- 오믹스 분석에서 대규모 잡 관리 전략을 개괄할 수 있다.

핵심 질문

- 인스턴스를 여러 대 띄워 permutation test를 돌릴 때 무엇이 가장 힘든가
- 구성 관리와 배치 실행은 어떻게 다른가

- Lambda는 어디까지 유용하고 어디서 한계가 있는가

수작업 운영이 무너지는 순간

오믹스 분석을 처음 AWS로 옮길 때는 한두 번의 실습이 모든 것을 단순해 보이게 만든다. 인스턴스 하나를 띄우고, 필요한 패키지를 설치하고, 스크립트를 올리고, 데이터를 S3에서 내려받아 돌리면 된다. 이 단계에서는 "클라우드가 생각보다 쉽다"는 인상이 생긴다. 그러나 같은 작업을 100번, 1,000번 반복해야 하는 순간 문제가 달라진다. 이제 병목은 CPU 수가 아니라, 누락된 패키지 버전, 서로 다른 설정 파일, 실패한 잡의 추적, 결과 수집, 다시 실행 범위 계산 같은 운영 문제로 이동한다.

대표적인 예가 permutation test다. 5분짜리 계산 하나는 사람이 직접 돌려도 어렵지 않다. 하지만 10,000번을 반복해야 하면, 그때부터 필요한 것은 "더 빠른 컴퓨터"보다 "동일한 환경을 반복 가능하게 만들고, 실패를 복구하고, 결과를 안전하게 모으는 구조"다. 이 지점에서 자동화는 편의 기능이 아니라 연구 설계의 일부가 된다. 분석이 커질수록 중요한 것은 서버를 얼마나 멋지게 띄우는가가 아니라, 같은 작업을 얼마나 일관되게 여러 번 실행할 수 있는가이다.

따라서 이 장에서는 자동화를 세 층으로 나누어 생각하는 것이 좋다. 첫째는 여러 서버의 환경을 같은 상태로 맞추는 (configuration management)이고, 둘째는 많은 잡을 큐에 넣고 재시도와 우선순위를 관리하는 (batch execution)이며, 셋째는 S3 업로드, 메타데이터 생성, 상태 변화처럼 계산 주변의 작은 사건을 이어 붙이는 (event automation)다. Ansible, AWS Batch, Lambda는 각각 이 세 층에서 강점을 가진다. 세 도구를 경쟁 관계로 이해하기보다, 서로 다른 운영 문제를 푸는 도구로 이해하는 편이 훨씬 정확하다.

Spot Instance로 짧고 많은 계산 저렴하게 처리하기

짧고 독립적인 작업을 많이 돌릴 때 가장 먼저 떠올릴 수 있는 자원이 EC2 Spot Instance다. Spot은 남는 용량을 할인된 가격으로 사용하는 방식이므로, 정렬 batch, permutation test, Monte Carlo simulation, bootstrap, 파라미터 스윕처럼 개별 작업이 독립적이고 다시 실행하기 쉬운 workload와 잘 맞는다. AWS 공식 문서도 Spot을 stateless, fault-tolerant, flexible workload에 적합한 자원으로 설명한다 (AWS 2026a). 오믹스 분석에서는 특히 샘플 단위로 쪼개기 쉬운 잡이나, 실패해도 일부 shard만 재실행하면 되는 작업이 좋은 후보가 된다.

하지만 Spot을 단순히 "싼 EC2"로만 이해하면 곤란하다. Spot은 언제든 회수될 수 있고, 일반적으로 중단 2분 전에 interruption notice가 제공된다 (AWS 2026b). 따라서 계산 상태가 인스턴스 안에만 남아 있거나, 중간 결과를 주기적으로 외부 저장소에 기록하지 않는 구조는 Spot과 궁합이 나쁘다. 반대로 입력 manifest와 결과 저장 위치가 S3에 있고, 작업 단위가 짧고 독립적이며, 실패한 조각만 다시 태울 수 있다면 Spot은 매우 강력한 선택이 된다. 여기서 핵심은 인스턴스보다 워크로드를 Spot 친화적으로 설계하는 것이다.

실전 운영에서는 세 가지 습관이 중요하다. 첫째, 작업을 가능한 한 짧고 독립적인 단위로 나눈다. 둘째, 결과와 체크포인트를 인스턴스 내부가 아니라 S3 같은 durable storage에 남긴다. 셋째, 한 종류의 인스턴스에 집착하지 않고 여러 타입과 여러 가용 영역에 유연하게 열어 둔다. Spot의 본질은 "예측 가능한 서버"가 아니라 "가용한 용량을 탄력적으로 활용하는 시장"에 가깝기 때문이다. 비용 절감은 계산 자체보다 이런 운영 원칙을 잘 지킬 때 생긴다.

Ansible로 환경을 반복 가능하게 만들기

Ansible의 강점은 많은 서버를 같은 상태로 만드는 데 있다. 예를 들어 50개의 단일 코어 인스턴스에 같은 R 버전과 패키지, 같은 스크립트, 같은 입력 경로를 배포하고 싶다면, 사람이 SSH로 하나씩 들어가 설정하는 방식은 금방 무너진다. Ansible은 이런 반복 작업을 playbook으로 선언해 두고 여러 인스턴스에 동일하게 적용할 수 있게 해 준다. 따라서 Ansible은 "대규모 계산을 스케줄링하는 도구"라기보다, "같은 계산 환경을 빠르게 복제하는 도구"로 이해하는 편이 맞다.

오믹스 분석에서 Ansible은 특히 bootstrap 단계에서 유용하다. 인스턴스 launch 이후 공통 패키지 설치, 환경 변수 설정, reference와 스크립트 배치, 실행 범위 분배, 로그 수집 지점 통일 같은 작업을 코드로 남길 수 있기 때문이다. 이 접근의 가장 큰 장점은 재현성이다. 서버 하나를 손으로 설정하면 그때그때는 빠를 수 있지만, 몇 주 뒤 같은 실험을 재현하거나 다른 학생이 이어받을 때 거의 항상 문제가 생긴다. 반면 playbook은 "무엇을 설치했고 어떤 순서로 설정했는가"를 문서이자 실행 코드로 남긴다.

그렇다고 Ansible이 잡 큐까지 알아서 관리해 주는 것은 아니다. Ansible로도 여러 인스턴스에 범위 기반 작업을 나눠 줄 수 있지만, 실패 재시도, 동적 스케일링, 수천 개 잡의 상태 추적, 인터럽션 대응 같은 기능은 본질적으로 구성 관리보다 상위의 오케스트레이션 문제다. 그래서 오늘날의 실전 구조에서는 Ansible = provisioning bootstrap, AWS

`Batch =  , S3 =  durable state`처럼 역할을 분리하는 편이 더 자연스럽다. 이 구분을 이해하면 Ansible을 과소평가하지도, 과대평가하지도 않게 된다.

Table 1. Ansible, AWS Batch, Lambda의 역할 차이

도구	가장 잘 푸는 문제	유전체 예시
Ansible	여러 서버에 같은 환경을 반복 배포	permutation용 EC2에 R, 패키지, 스크립트 설치
AWS Batch	많은 잡을 큐에 넣고 재시도·스케일링 관리	샘플 단위 QC, Monte Carlo, containerized batch pipeline
AWS Lambda	이벤트 기반의 작은 자동화	S3 업로드 감지 후 검증, 배치 실행 시작, 알림 발송

AWS Batch로 대규모 잡 큐 운영하기

AWS Batch는 "서버를 내가 직접 배열하느냐"보다 "작업을 큐와 정책으로 관리하느냐"에 초점을 두는 서비스다. 사용자는 잡 정의(job definition), 잡 큐(job queue), 컴퓨트 환경(compute environment)을 설정하고, 실제 인스턴스 선택과 확장은 상당 부분 Batch에 맡긴다. 이 구조의 핵심 이점은 대규모 반복 작업에서 사람이 직접 인스턴스 상태를 관리하지 않아도 된다는 점이다. 즉 Batch는 EC2를 없애는 서비스가 아니라, EC2 위에 관리형 스케줄러 계층을 얹는 서비스다 (AWS 2026c).

오믹스 분석에서는 이 점이 매우 중요하다. permutation, Monte Carlo, 샘플 단위 QC, 컨테이너화된 alignment step, cohort shard 분석처럼 서로 독립적인 잡이 많은 workload는 Batch와 잘 맞는다. 특히 array job은 Monte Carlo simulation이나 parametric sweep처럼 공통 정의를 가진 병렬 작업에 효율적이며, 공식 문서상 배열 크기는 2부터 10,000까지 가능하다 (AWS 2026d). 즉 permutation 10,000회를 "서버 10,000대"로 생각하기보다, "같은 정의를 가진 자식 잡 10,000개"로 생각하는 방식으로 전환할 수 있다.

Batch의 또 다른 장점은 실패를 운영 수준에서 다룰 수 있다는 점이다. 재시도 정책, timeout, CloudWatch 로그, Spot 기반 컴퓨트 환경, 다양한 인스턴스 타입 허용 같은 기능을 조합하면, 사람이 잡 하나하나를 손으로 돌보지 않아도 된다. 이때 중요한 설계 원칙은 짧은 작업을 너무 잘게 쪼개지 않는 것이다. AWS Batch 문서는 몇 초짜리 초단기 작업은 스케줄링 오버헤드가 더 클 수 있으므로, 필요하면 여러 작업을 묶어 3–5분 이상 실행되도록 binpack하는 방식을 권장한다 (AWS 2026e). 반대로 10–20분 정도의 독립 작업은 Batch와 잘 맞는 길이다.

실전에서는 `manifest -> array index -> shard range -> S3 output` 구조가 특히 유용하다. 부모 잡은 입력 manifest와 공통 파라미터를 가리키고, 각 자식 잡은 환경 변수나 배열 인덱스를 사용해 서로 다른 샘플, seed, permutation range를 처리한다. 이렇게 하면 결과 수집과 재실행 범위를 명확히 할 수 있고, 실패한 child job만 다시 실행하기도 쉬워진다. 즉 Batch의 핵심 가치는 서버 생성 자동화가 아니라, 반복 계산을 로 바꾸는 데 있다.

Lambda로 만드는 작은 자동화

Lambda는 종종 "서버리스니까 뭐든 할 수 있는 계산 서비스"로 오해되지만, 오믹스 분석에서의 가장 좋은 역할은 대규모 계산 엔진이 아니라 작은 접착제(glue)다. 즉 Lambda는 alignment나 variant calling을 직접 오래 수행하기보다, S3 업로드 감지, 메타데이터 검사, manifest 생성, HealthOmics나 Batch 실행 시작, 상태 알림 같은 짧은 이벤트 기반 작업에 더 적합하다. 이 차이를 분명히 이해해야 한다. 장시간 계산은 Batch, EMR, HealthOmics 같은 계층에 맡기고, Lambda는 흐름을 연결하는 쪽에 두는 편이 설계가 훨씬 깨끗해진다.

예를 들어 시퀀싱 센터가 FASTQ를 S3에 올리면, Lambda가 업로드 이벤트를 받아 파일명과 샘플 메타데이터를 검증하고, 필요한 manifest를 만들고, 그다음 Step Functions나 HealthOmics, Batch를 호출하는 흐름을 생각할 수 있다. 이 구조에서는 Lambda가 계산의 몸통이 아니라, 계산을 시작시키고 상태를 이어 주는 손잡이가 된다. 오믹스 분석 파이프라인이 커질수록 이런 작은 자동화는 체감상 매우 큰 차이를 만든다. 사람에게 메일을 보내는 것보다 먼저, 기계가 다음 단계를 준비하게 만드는 것이다.

따라서 Lambda의 한계도 함께 이해해야 한다. 길고 무거운 계산, 큰 메모리와 긴 실행 시간이 필요한 작업, 대용량 scratch를 요구하는 작업은 Lambda에 맞지 않는다. Lambda는 어디까지나 라는 정체성을 가진다. 이 선을 넘기려 할수록 설계가 불안정해진다. 반대로 이 선을 정확히 지키면, 복잡한 유전체 운영에서 사람의 클릭을 크게 줄일 수 있다.

Batch, EKS, ParallelCluster의 선택

자동화가 깊어질수록 "그렇다면 Batch 대신 Kubernetes나 Slurm을 바로 배우는 것이 낫지 않은가"라는 질문이 나온다. 이 질문에 대한 가장 실용적인 답은, 먼저 팀이 무엇을 이미 운영하고 있는지 보라는 것이다. AWS Batch는 배치 잡을 빠르게 운영하고 싶지만 자체 스케줄러를 관리하고 싶지 않은 팀에게 잘 맞는다. Amazon EKS는 이미 Kubernetes를 플랫폼으로 쓰고 있고, 배치 작업뿐 아니라 API, UI, 노트북, 추론 서비스까지 같은 클러스터에서 운영하려는 조직에 더 자연스럽다. AWS ParallelCluster는 Slurm 중심의 전통적 HPC 운영 방식과 공유 파일시스템을 AWS로 가져오고 싶은 경우에 더 적합하다 (AWS 2026f; AWS 2026g).

초급 연구실의 첫 선택지로는 대체로 Batch가 가장 단순하다. 컨테이너화된 잡을 큐에 넣고, 필요할 때만 확장하고, Spot과 On-Demand를 섞어 쓰기 쉽기 때문이다. EKS는 강력하지만 그만큼 플랫폼 복잡도가 크다. ParallelCluster 역시 매우 유용하지만, 이는 HPC 을 AWS로 옮기는 쪽에 가깝다. 즉 세 서비스는 우열 관계라기보다 출발점이 다르다. 현재 팀이 Kubernetes나 Slurm을 이미 중심 도구로 삼고 있지 않다면, 많은 유전체 batch workflow의 기본값은 여전히 Batch 쪽이 더 자연스럽다.

Table 2. 대규모 배치 실행 계층의 선택 기준

계층	가장 잘 맞는 상황	주의점
AWS Batch	containerized batch workload, 반복 실행, Spot 활용	초단기 작업은 binpack이 필요할 수 있음
Amazon EKS	배치 외에도 다양한 서비스와 Kubernetes 운영이 필요한 팀	클러스터와 RBAC 운영 복잡도가 큼
AWS ParallelCluster	Slurm 기반 HPC와 공유 파일시스템이 중요한 환경	전통적 클러스터 운영 개념을 여전히 이해해야 함

어떤 도구를 언제 선택할 것인가

지금까지의 내용을 하나로 정리하면, 도구 선택의 핵심은 "얼마나 많은 계산을 하느냐"보다 "무엇을 자동화하려 하느냐"에 있다. 여러 서버에 같은 환경을 배포하는 것이 문제라면 Ansible이 맞다. 많은 잡을 큐에 넣고 재시도와 스케일링을 관리하는 것이 문제라면 Batch가 맞다. 파일 업로드나 메타데이터 이벤트를 감지해 다음 단계를 시작시키는 것이 문제라면 Lambda가 맞다. 즉 자동화는 하나의 도구로 끝나는 일이 아니라, 문제 층에 따라 서로 다른 도구를 조합하는 일이다.

오믹스 분석의 관점에서 가장 현실적인 기본 조합은 다음과 같다. Spot + Batch, Ansible, Lambda, durable state S3다. 이 구조는 기능을 멋지게 나눠 놓은 추상도가 아니라, 실패 복구와 재현성을 높이는 실전 구조다. 사람 손이 덜 타는 파이프라인일수록 더 큰 규모에서도 흔들리지 않는다.

다음 장의 심화편에서는 바로 이 문제를 permutation workload 관점에서 더 좁혀 본다. 즉 Ansible + Spot으로도 가능한 구조와, 오늘날 더 권장되는 Batch Fleet 중심 구조를 비교하면서, 기술적으로 가능한 것과 운영상 권장되는 것을 구분해 볼 것이다. 자동화의 진짜 성숙은 "할 수 있다"에서 끝나지 않고, "오랫동안 반복 가능하게 운영할 수 있다"로 이어질 때 완성된다.

핵심 개념 정리

- 자동화의 핵심 병목은 CPU보다도 환경 일관성, 실패 복구, 결과 수집, 재실행 범위 관리에 있다.
- Spot은 짧고 독립적이며 재실행 가능한 workload에 특히 강하다.
- Ansible은 구성 관리와 bootstrap에, AWS Batch는 잡 큐와 스케일링에, Lambda는 이벤트 기반 연결에 가장 잘 맞는다.
- Batch, EKS, ParallelCluster는 같은 문제를 푸는 도구가 아니라 서로 다른 운영 전제를 가진 실행 계층이다.

복습 질문

1. permutation test 같은 workload에서 수작업 운영이 빠르게 무너지는 이유는 무엇인가?
2. Spot Instance를 오믹스 분석에 사용할 때 반드시 함께 설계해야 하는 세 가지 운영 원칙은 무엇인가?
3. Ansible과 AWS Batch는 각각 어떤 층의 문제를 풀며, 왜 이를 구분하는 것이 중요한가?

Further Reading

- AWS. What is AWS Batch?.
- AWS. Best practices for Amazon EC2 Spot.
- AWS. What is AWS ParallelCluster.

References

- AWS. 2026a. Best practices for Amazon EC2 Spot.
- AWS. 2026b. Spot Instance interruption notices.
- AWS. 2026c. What is AWS Batch?.
- AWS. 2026d. Array jobs.
- AWS. 2026e. When to use AWS Batch.
- AWS. 2026f. Getting started with AWS Batch on Amazon EKS.
- AWS. 2026g. What is AWS ParallelCluster.
- AWS. 2026h. Job timeouts.
- AWS. 2026i. Welcome to AWS Lambda.
- Ansible Community Documentation. 2026. amazon.aws.ec2_instance module.

7장. Spot, Batch, Fleet로 하는 대규모 병렬 작업 운영

학습 목표

- 짧고 독립적인 계산을 수천 개 규모로 묶어 돌려야 하는 오믹스 작업 유형을 설명할 수 있다.
- Ansible, AWS Batch, EC2 Fleet/Spot Fleet가 각각 맡아야 할 역할을 구분할 수 있다.
- Spot interruption, quota, scheduling overhead를 고려한 실전 아키텍처를 제시할 수 있다.

핵심 질문

- 어떤 종류의 오믹스 작업이 Spot 기반 대규모 병렬에 잘 맞는가
- Ansible로 수천 개 Spot 인스턴스를 직접 띄워도 되는가
- 2026년 기준으로는 왜 Batch나 Fleet가 더 자연스러운가

대규모 병렬이 자주 필요한 오믹스 작업들

이 장에서 다루는 운영 패턴은 특정 분석법을 위한 것이 아니다. 오믹스 연구에는 짧고 서로 독립적인 계산을 수천 개에서 수만 개 규모로 쏟아내야 하는 작업이 반복적으로 등장하는데, 이런 작업들은 공통된 특징을 가진다. 각 단위 계산이 수 분에서 수십 분 사이로 끝나고, 결과가 서로 의존하지 않으며, 중간에 일부가 실패해도 그 부분만 다시 돌리면 된다는 점이다. 이런 특징을 가진 작업은 중간 회수(interruption)가 있을 수 있는 Spot 인스턴스와 자연스럽게 맞물리고, 상위에서 Batch나 Fleet 같은 오케스트레이션 계층이 있을 때 안정적으로 운영된다. 장의 나머지는 permutation testing을 자주 다루는 예로 쓰지만, 같은 원리가 다른 작업에도 그대로 적용된다.

대표적인 예시는 다섯 부류로 묶어 볼 수 있다. 첫째, permutation testing과 label shuffling이다. 연관 분석이나 유전자 집합 검정에서 표현형 라벨을 수천에서 수만 번 섞고 다시 통계를 계산해 경험적 p-value 분포를 얻는 방식이다. 둘째, bootstrap resampling이다. 신뢰구간을 추정하거나 sampling 안정성을 평가하기 위해 데이터를 반복적으로 재표집하며 같은 추정을 다시 수행한다. 셋째, simulation이다. coalescent simulation, forward-time 유전 시뮬레이션, 모델 기반 가짜 데이터 생성처럼 한 번의 실행이 독립적인 random seed에만 의존하는 계산이 여기에 해당한다.

넷째, parametric sweep이다. 모델 하이퍼파라미터, 품질 관리 임계값, 필터링 조건의 조합을 대규모로 탐색하면서 각 조합을 독립 작업으로 돌리는 경우다. 머신러닝 모델 튜닝뿐 아니라 QC 파라미터의 민감도 분석, variant calling filter 조합 비교에도 이 패턴이 그대로 쓰인다. 다섯째, cohort shard 분석이다. 수만 명 규모의 코호트를 샘플 묶음이나 유전체 구간으로 쪼개어 각 shard를 독립적으로 분석하고, 끝난 결과를 나중에 한자리로 모으는 방식이다. single-cell 데이터에서 세포 배치별 전처리, 대규모 VCF에서 구간별 annotation도 같은 구조 안에 들어온다.

다섯 부류는 서로 다른 연구 목적을 가지지만 운영 관점에서는 같은 문제를 푼다. 어떻게 하면 수천 개의 짧은 독립 작업을, 저렴한 자원을 써서, 중간 실패에도 흔들리지 않게 돌려낼 것인가. 이 질문에 답하려면 Spot 인스턴스의 성격, vCPU quota, 오케스트레이션 계층의 선택, shard 설계, 상태 보존이 모두 함께 움직여야 한다. 뒤의 절들은 이 요소를 하나씩 짚어 나간다.

Table 1. 짧은 독립 작업의 대표 예시와 공통 성질

작업 유형	단위 계산	공통 특징
permutation / label shuffling	라벨 섞고 통계 재계산	수천~수만 반복, 각 반복 독립
bootstrap resampling	재표집 후 추정 반복	신뢰구간/안정성 평가, 각 샘플 독립
simulation	seed별 합성 데이터 생성·분석	순수 random seed 의존, 복원 가능
parametric sweep	파라미터 조합별 반복 실행	조합 공간 탐색, 각 조합 독립
cohort shard 분석	샘플 묶음/유전체 구간별 분석	최종적으로 결과 통합

옛 방식의 미덕 – Ansible + Spot 인스턴스

실제 연구 현장에서 가장 먼저 시도되는 방식은 `Ansible Spot` 였다. 이 방식은 오늘도 기술적으로 가능하다. 짧고 독립적인 계산을 대량으로 분산시키는 데 Spot은 잘 맞고, Ansible은 같은 환경을 빠르게 복제하는 데 매우 유용하기 때문이다. 따라서 이 방식은 "옛날이라 틀린 방식"이 아니라, 클라우드 초창기 연구 현장에서 실용적이었던 접근이다. 이 절의 목적은 그 경험을 부정하는 것이 아니라, 오늘날 그 경험이 어떤 상위 계층으로 흡수되었는지 연결해 보여 주는 데 있다.

이 구조의 장점은 직관성이다. 인스턴스를 띄우고, playbook으로 필요한 패키지를 설치하고, 각 서버에 반복 범위를 나눠 주고, 결과를 모으면 된다. 팀이 이미 SSH와 Linux 운영에 익숙하고, 한 번에 끝내야 하는 대량 실행을 빨리 해치워야 한다면 여전히 나쁜 선택은 아니다. 특히 계산 단위가 매우 단순하고, 중앙 큐나 컨테이너 계층을 새로 배우는 비용이 더 클 때는 직접 제어가 오히려 편할 수 있다.

하지만 규모가 수천 개 수준으로 커지면, 이 구조는 곧바로 의 문제가 아니라 의 문제로 바뀐다. 어떤 인스턴스가 중간에 회수되었는지, 어떤 범위가 아직 끝나지 않았는지, 어디서부터 재실행해야 하는지, quota 때문에 실제로 몇 대가 떠 있는지, 서로 다른 인스턴스 타입이 섞여도 결과 수집 규칙이 유지되는지 같은 질문이 더 중요해진다. 즉 `Ansible + Spot` 구조는 가능하지만, 오케스트레이션 부담을 사람이 더 많이 떠안는 구조라는 점을 분명히 이해해야 한다.

기술적으로 가능한가 – Spot vCPU quota와 Fleet quota

수천 개 Spot 인스턴스 launch가 가능한지 묻는다면, 공식 문서는 분명히 "가능하다"고 답한다. EC2 Fleet과 Spot Fleet 문서는 tens, hundreds, or thousands of EC2 instances를 한 번의 요청으로 시작할 수 있다고 설명하며, fleet당 target capacity 기본 quota는 10,000, 리전 전체 target capacity quota는 100,000으로 제시한다 (AWS 2026a). 즉 "수천 개 인스턴스" 자체는 비정상적인 요구가 아니다. 문제는 이것이 곧바로 아무 계정에서나 기본 설정으로 가능한 것은 아니라는 점이다.

실제 병목은 대개 Spot vCPU quota와 capacity availability다. Spot quota는 인스턴스 개수로 관리되지 않고 vCPU 단위로 관리되며, 계정과 리전별로 적용된다 (AWS 2026b). 예를 들어 기본 계정에서 Standard Spot 요청 quota가 매우 낮게 시작할 수 있으므로, 1,000개의 c 계열 인스턴스를 바로 띄우려 하면 quota에 먼저 막힐 수 있다 (AWS 2026c). 따라서 "기술적으로 가능하다"와 "내 계정에서 지금 바로 가능하다"는 서로 다른 문장이다. 대규모 병렬 실행을 준비할 때는 launch template보다 먼저 현재 Spot quota와 필요한 target capacity를 계산해야 한다.

또 하나 놓치기 쉬운 점은 capacity가 항상 즉시 확보되지 않는다는 사실이다. Spot best practices 문서는 필요한 aggregate capacity가 언제나 즉시 주어지거나 계속 유지된다고 보장하지 않는다고 분명히 말한다 (AWS 2026d). 따라서 짧은 독립 작업이 Spot과 잘 맞는다고 해서, 원하는 시점에 원하는 수만큼 인스턴스를 한 번에 확보할 수 있다고 가정하면 안 된다. 다중 인스턴스 타입, 다중 가용 영역, 유연한 shard 설계가 중요한 이유가 여기에 있다.

Table 2. 대규모 Spot 병렬에서 먼저 확인할 항목

항목	왜 중요한가	대표 확인 포인트
Spot vCPU quota	계정이 허용하는 총 Spot 규모를 결정함	필요한 인스턴스 총 vCPU 수가 현재 quota 안에 들어오는가
Fleet target capacity quota	한 fleet과 리전 전체에서 요청 가능한 규모를 제한함	fleet당 10,000, 리전 전체 100,000 기본 quota 확인
가용 용량	실제로 어느 타입·AZ에서 Spot이 확보되는지 좌우함	여러 인스턴스 타입과 여러 AZ를 열어 두었는가
작업 단위 길이	interruption에 대한 내성을 결정함	실패 시 다시 돌릴 shard가 충분히 짧은가

왜 요즘은 Batch나 Fleet가 더 자연스러운가

오늘날 `Ansible + Spot request`보다 `AWS Batch EC2 Fleet`가 더 자연스러운 첫 선택지인 이유는, 운영의 무게중심이 이미 인스턴스 생성보다 상위 계층으로 옮겨 갔기 때문이다. AWS 자체도 `RequestSpotInstances` API를 legacy API로 보고 강하게 권장하지 않는다 (AWS 2026e). 대신 Spot best practices는 EC2 Fleet, Auto Scaling, 통합 서비스, price-capacity-optimized 전략 같은 더 현대적인 요청 계층을 권한다 (AWS 2026d). 메시지는 분명하다. Spot을 쓰지 말라는 뜻이 아니라, Spot을 직접 원시 API 수준에서 다루기보다 interruption-aware orchestration 계층과 함께 쓰라는 뜻이다.

EC2 Fleet는 직접 제어를 유지하면서도 다중 인스턴스 타입과 다중 서브넷, 다중 용량 풀을 하나의 요청으로 관리하게 해 준다. 자체 스케줄러를 가지고 있거나, 컨테이너보다 인스턴스 수준 제어를 더 중요하게 여기는 팀에게는 매우 유용하다. 반면 AWS Batch는 그 위에 잡 큐, array job, retry, timeout, 로그, Spot-backed compute environment를 얹어 주므로, 많은 연구실에게 더 높은 수준의 기본값이 된다. 쉽게 말해 Fleet는 `capacity orchestration`에 가깝고, Batch는 `job orchestration`에 가깝다. 실제 관심사가 "몇 대의 서버"보다 "몇 개의 반복 계산"에 있을 때는 후자가 더 자연스럽다.

또한 Batch는 인스턴스 다양화와 Spot 전략을 서비스 수준에서 흡수하기 쉬운 장점이 있다. AWS Batch 문서는 Spot 기반 컴퓨트 환경에서 `SPOT_CAPACITY_OPTIMIZED` 전략과 다양한 인스턴스 타입 허용을 권장한다 (AWS 2026f). 이 조합은 연구자가 직접 용량 풀을 손으로 고르는 부담을 줄여 준다. Batch나 Fleet가 더 권장되는 이유는 "더 멋진 서비스"라서가 아니라, 사람 손으로 처리해야 할 실패 복구와 capacity 분산을 더 구조적으로 다룰 수 있기 때문이다.

Array job, shard, retry, timeout으로 짧은 작업 설계하기

짧은 독립 작업을 현대적으로 설계할 때 가장 중요한 단위는 인스턴스가 아니라 shard다. 예를 들어 100,000회 permutation을 해야 한다면, 이를 1회 단위로 다루는 대신 100회나 500회씩 묶어 child job 하나가 일정 범위를 책임지도록 설계하는 편이 낫다. 같은 원리가 bootstrap 1,000회, 시뮬레이션 seed 10,000개, 파라미터 조합 5,000개, 샘플 shard 2,000개에도 그대로 적용된다. 이렇게 묶으면 스케줄링 오버헤드를 줄이고, Spot interruption이 나더라도 실패한 범위만 다시 실행할 수 있다. AWS Batch array job은 바로 이런 구조에 적합하며, array size는 2에서 10,000까지 가능하다 (AWS 2026g). 각 child job은 `AWS_BATCH_JOB_ARRAY_INDEX`를 사용해 서로 다른 random seed, iteration range, 파라미터 인덱스, shard ID를 받도록 구성할 수 있다.

10–20분짜리 작업은 Batch에 잘 맞는 길이다. AWS Batch 문서는 수 초짜리 짧은 작업은 스케줄러 오버헤드가 더 클 수 있으니 필요하면 binpack해서 3–5분 이상이 되도록 권장한다 (AWS 2026h). 즉 10–20분은 "너무 짧아서 Batch에 부적합한" 길이가 아니라, 오히려 scheduler overhead와 interruption recovery 사이의 균형이 좋은 편이다. 반대로 각 반복이 너무 짧다면, 여러 개를 한 child job 안에서 루프 돌게 묶는 설계가 더 현실적이다.

retry와 timeout은 별도로 생각해야 한다. Batch의 timeout은 잡이 너무 오래 걸릴 때 강제로 종료하기 위한 장치이고, `attemptDurationSeconds`는 각 시도에 적용된다. 공식 문서는 timeout으로 종료된 작업은 자동 retry되지 않으며, 자체 실패한 작업만 retry 정책에 따라 다시 시도될 수 있다고 설명한다 (AWS 2026i). 따라서 설계 단계에서는 "실패"와 "멈춤"을 같은 사건으로 다루면 안 된다. stuck process를 막기 위해 timeout을 걸고, Spot interruption이나 일시적 실패는 재시도 정책으로 흡수하며, 이미 완료한 shard는 S3 manifest로 체크하는 식의 분리가 필요하다.

Table 3. 짧은 독립 작업 shard 설계의 실전 원칙

설계 항목	권장 방향	이유
child job 길이	대개 수 분 이상, 필요하면 10–20분 수준	scheduler overhead를 줄이고 interruption recovery 단위를 적절히 유지
상태 저장	S3 manifest, 로그, 결과 파일에 남김	인스턴스 손실 시에도 진행 상태를 복원 가능
실패 처리	retry와 timeout을 분리	stuck process와 일시적 실패는 성격이 다름
작업 분배	array index 기반 seed/range/param 분배	재실행 범위를 명확하게 제한 가능

Ansible은 어디에 남는가 – provisioning과 bootstrap

그렇다면 오늘날 Ansible은 더 이상 쓸모가 없는가? 전혀 그렇지 않다. 다만 역할이 바뀌었다고 보는 편이 정확하다. Ansible은 수천 개 child job의 주 오케스트레이터라기보다, VPC, IAM, launch template, AMI bootstrap, 공통 패키지 설치, 진단 스크립트 배포 같은 provisioning 층에서 여전히 강하다. 즉 에서는 Ansible이 매우 유용하고, 에서는 Batch나 Fleet가 더 적합한 경우가 많다.

이 구분은 Ansible 공식 문서에도 어느 정도 드러난다. `amazon.aws.ec2_instance` 모듈은 EC2 인스턴스 생성과 관리를 지원하지만 Spot instance 생성 자체는 지원하지 않고, 이를 위해 별도의 `amazon.aws.ec2_spot_instance` 모듈을 사용하도록 안내한다 (Ansible Community Documentation 2026a; 2026b). 현대적인 Ansible workflow에서도 Spot은 AWS 계층을 직접 존중하면서 다뤄야 한다. 즉 "Ansible이 있으니 AWS의 스케줄링 계층은 필요 없다"가 아니라, "Ansible은 준비와 부트스트랩에, Fleet/Batch는 실제 확장과 운영에"라는 역할 분리가 더 자연스럽다.

소규모 연구실의 일회성 대량 실행이라면, 여전히 `Ansible + Spot` 조합이 가장 빠른 해법일 수 있다. 중요한 것은 이를 보편적 기본값으로 일반화하지 않는 것이다. 반복적으로 돌아오는 대형 실행, 장기간 유지할 재현 가능한 운영, 여러 사람이 함께 쓰는 큐 환경에서는 상위 오케스트레이션 계층의 가치가 훨씬 커진다.

추천 아키텍처 – 2026년 기준 실전 선택표

2026년 기준으로 가장 현실적인 구조는 세 층으로 정리할 수 있다. 첫째, `Ansible = provisioning bootstrap`. 둘째, `AWS Batch EC2 Fleet = interruption-aware orchestration`. 셋째, `S3 + manifest + logs = durable state observability`. 이 세 층을 분리하면 인프라 준비와 실행 제어와 상태 보존이 뒤섞이지 않는다. 대규모 병렬 작업 운영의 관건은 "서버를 많이 띄우는 기술"이 아니라, 를 만드는 일이다.

가장 일반적인 권장안은 `AWS Batch array jobs + Spot-backed managed compute environment`다. 반복적이고 재현 가능한 대규모 운영에 잘 맞고, child job 단위 재시도와 timeout, 로그 추적을 서비스 수준에서 다루기 쉽기 때문이다. 자체 스케줄러를 유지하면서 인스턴스 수준 제어를 더 세밀하게 하고 싶다면 `EC2 Fleet/Spot Fleet +` 이 맞을 수 있다. 반면 학습 비용을 최소화하고 빠르게 한 번 돌려야 하는 연구라면 `Ansible + Spot`도 여전히 실용적이다. 중요한 것은 가능성과 권장을 구분하는 것이다.

Table 4. 2026년 기준 대규모 병렬 작업 운영 선택표

상황	가장 자연스러운 선택	이유
일회성 대량 실행, 팀 규모 작음	Ansible + Spot	빠르게 환경을 맞추고 직접 제어하기 쉬움
반복적이고 재현 가능한 대규모 운영	AWS Batch array jobs + Spot-backed compute environment	queue, retry, timeout, scaling을 구조적으로 관리 가능
자체 scheduler 유지, 세밀한 capacity 제어 필요	EC2 Fleet/Spot Fleet	인스턴스 수준 제어와 capacity orchestration에 유리

영어로 한 문장 요약하면 이렇다. `Ansible can still provision and coordinate Spot-based infrastructure, but for launching and managing thousands of short-lived independent jobs - whether permutation,`

bootstrap, simulation, parametric sweep, or cohort shard analysis - AWS Batch or EC2 Fleet provides a more scalable and interruption-aware orchestration layer. 여기서 배울 것은 특정 도구의 승패가 아니라, 어떤 계층이 어떤 운영 부담을 흡수하는지를 구분하는 감각이다. 그래야 같은 원리를 permutation뿐 아니라 bootstrap, simulation, parametric sweep, cohort shard 분석까지 일관되게 확장할 수 있다.

핵심 개념 정리

- 짧고 독립적인 계산이 수천~수만 개 필요한 작업은 오믹스 연구 전반에서 반복 등장한다. permutation은 그중 한 대표 예시일 뿐이다.
- 수천 개 Spot 인스턴스 launch 자체는 기술적으로 가능하지만, 실제 병목은 quota, capacity, failure recovery에 있다.
- Spot 병렬 설계의 핵심 단위는 인스턴스가 아니라 shard와 durable state다.
- Batch는 job orchestration에, Fleet는 capacity orchestration에, Ansible은 provisioning과 bootstrap에 더 가깝다.
- 2026년 기준 권장 기본값은 대체로 Batch Fleet + Ansible 구조다.

복습 질문

1. 이 장에서 다룬 다섯 가지 작업 유형(permutation, bootstrap, simulation, parametric sweep, cohort shard)이 운영 관점에서 같은 문제로 묶이는 이유는 무엇인가?
2. “기술적으로 가능하다”와 “기본 계정에서 바로 가능하다”는 왜 다른 말인가?
3. 짧은 독립 작업에서 shard, retry, timeout을 분리해 설계해야 하는 이유는 무엇인가?
4. Batch와 Fleet는 각각 어떤 층의 오케스트레이션을 담당하며, Ansible은 어디에 남는가?

Further Reading

- AWS. Quotas for EC2 Fleet and Spot Fleet.
- AWS. Array jobs.
- AWS. Use Amazon EC2 Spot best practices for AWS Batch.

References

- AWS. 2026a. Quotas for EC2 Fleet and Spot Fleet.
- AWS. 2026b. Spot Instance quotas.
- AWS. 2026c. Amazon EC2 instance type quotas.
- AWS. 2026d. Best practices for Amazon EC2 Spot.
- AWS. 2026e. RequestSpotInstances.
- AWS. 2026f. Use Amazon EC2 Spot best practices for AWS Batch.
- AWS. 2026g. Array jobs.
- AWS. 2026h. When to use AWS Batch.
- AWS. 2026i. Job timeouts.
- Ansible Community Documentation. 2026a. amazon.aws.ec2_instance module.
- Ansible Community Documentation. 2026b. amazon.aws.ec2_spot_instance module.

8장. EMR과 Hail로 하는 대규모 WGS 분석

학습 목표

- AWS EMR이 무엇인지 설명할 수 있다.
- Hail이 Spark 기반 유전체 분석 도구로 왜 중요한지 이해할 수 있다.
- Hail VariantDataset (VDS)가 cohort-scale 분석에서 왜 중요한지 설명할 수 있다.
- WGS 수준의 대규모 변이 분석에서 EMR과 Hail의 장단점을 개괄할 수 있다.

핵심 질문

- EC2 여러 대를 직접 관리하는 것과 EMR을 쓰는 것은 어떻게 다른가
- Hail은 어떤 데이터 구조와 계산 모델을 사용하는가
- gnomAD와 같은 자원이 왜 Hail 생태계와 잘 맞는가
- 왜 VDS는 "큰 VCF"를 더 크게 만든 것이 아니라, 다른 데이터 표현인가

왜 EMR인가 – 직접 Spark 클러스터 대신 관리형 분산 분석

AWS EMR은 Spark, Hadoop, Hive, Trino 같은 분산 데이터 처리 도구를 더 쉽게 운영하도록 만든 관리형 플랫폼이다. 초보자에게는 "Spark 클러스터를 대신 띄워 주는 서비스"라고 설명해도 출발점으로는 충분하다. 하지만 유전체 분석 맥락에서 더 중요한 점은, EMR이 단순히 노드를 여러 대 띄우는 일을 넘어 클러스터 구성, 스케일링, 로그, 스토리지 연결을 하나의 운영 체계로 묶어 준다는 사실이다. WGS처럼 샘플 수와 변이 수가 동시에 커지는 분석에서는 계산 자체보다 가 더 큰 부담이 되기 쉽다. EMR은 바로 이 운영 부담을 줄여 주는 계층이다.

EMR을 이해할 때 자주 놓치는 포인트는 데이터 위치다. 전통적인 Hadoop 사고에서는 데이터를 HDFS로 먼저 옮겨 놓고 그 위에서 계산하는 흐름이 강했다. 하지만 AWS에서는 S3가 사실상의 기본 저장 계층이고, EMR은 EMRFS를 통해 S3 데이터를 직접 읽고 쓰는 구조를 자연스럽게 제공한다. 이 점은 오믹스 연구자에게 매우 중요하다. 이미 S3에 있는 VCF, Parquet, annotation table, 공개 데이터 자산을 다시 별도 스토리지로 복사하지 않고도 분산 계산에 연결할 수 있기 때문이다. 즉 EMR의 가치는 노드 수보다 S3 는 데 있다.

Hail이 무엇인가 – MatrixTable, Table, Spark

Hail은 Spark 위에서 유전체 데이터를 다루기 좋게 설계된 분산 분석 라이브러리다. 학생에게는 이를 "유전체학을 위한 분산 데이터프레임 환경" 정도로 소개하면 이해가 빠르다. Hail의 핵심 자료구조 가운데 하나인 `MatrixTable`은 행과 열이 각각 변이와 샘플에 대응하는 거대한 유전형 행렬을 표현하는 데 적합하다. 여기에 annotation, QC 지표, phenotype, covariate를 연결하면 cohort-scale 분석을 상대적으로 일관된 방식으로 표현할 수 있다. 즉 Hail은 Spark 자체를 직접 다루게 하기보다, Spark 위에 유전체 분석에 맞는 언어를 얹은 도구라고 볼 수 있다.

`joonan-lab/hail_tutorial`이 보여 주는 흐름은 Hail 교육의 좋은 출발점이다. VCF import로 시작해 MatrixTable 구조를 이해하고, 샘플 정보와 annotation을 붙이고, QC를 수행하고, qualifying variant를 추출하고, 마지막에 association testing까지 나아간다. AWS EMR 장에서는 이 개념 흐름을 그대로 유지하되, 입력 데이터를 `s3://...` 경로에서 읽고 계산이 Spark executor에 의해 병렬로 확장된다는 점만 추가하면 된다. 즉 로컬 튜토리얼과 클라우드 튜토리얼은 서로 다른 과목이 아니라, 같은 분석 사고를 다른 규모로 확장한 버전이다. 이 관점이 있어야 학생이 Hail을 "클라우드 전용 도구"가 아니라 "규모가 커질수록 더 빛나는 도구"로 이해할 수 있다.

VariantDataset – sparse representation, reference blocks, local alleles

Hail을 제대로 이해하려면 `VariantDataset (VDS)`를 따로 배워야 한다. Hail 문서에 따르면 VDS는 `variant_data`와 `reference_data`를 분리한 sparse, split representation이다. 즉 모든 샘플과 모든 위치를 dense하게 펼쳐 쓰는 대신, 실제 변이가 있는 부분과 참조 블록(reference block)을 나누어 저장한다. 이 구조의 핵심은 단순히 파일을 더 작게 만드는 데 있지 않다. 오히려 대규모 코호트를 실제로 다룰 수 있는 데이터 표현을 만드는 데 있다. Hail 문서와 All of Us 문서는 모두 VDS를 `stores less data, but more information`에 가까운 구조로 설명한다.

이 차이는 production-scale 사례에서 훨씬 선명해진다. gnomAD v4.0은 exome callset 생성 과정에서 VDS를 사용했고, 955,213 samples 규모의 전체 데이터를 18 TB로 유지할 수 있었지만 전통적 project VCF였다면 897 TB가 필요했을 것이라고 설명한다. 이 수치는 VDS가 압축 포맷이 아니라 cohort-scale 분석 표현이라는 사실을 매우 직관적으로 보여 준다. 또한 gnomAD v4.1은 VDS 덕분에 all callable sites에서 allele number를 유지할 수 있었고, 변이가 없는 위치까지 포함한 AN 자원을 제공할 수 있었다. 학생은 여기서 는 중요한 교훈을 배워야 한다.

Figure 8-1은 VCF의 dense 표현과 VDS의 sparse, split 표현을 교육용으로 단순화해 비교한 것이다. VCF는 모든 샘플과 모든 위치를 펼쳐 저장하지만, VDS는 참조 블록과 변이 블록을 분리하고 실제 변이가 있는 곳만 저장한다. 같은 데이터를 담아도 정보량은 더 크고 크기는 더 작아지는 이 대비를 시각적으로 기억하면, 이후 cohort-scale 분석에서 왜 저장 구조가 질문의 범위를 바꾸는지 쉽게 떠올릴 수 있다.

Figure 8-1. VCF(dense) vs VDS(sparse, split) 표현

VDS에서 또 하나 중요한 개념은 `local alleles`다. highly multiallelic site에서는 전통적인 VCF식 전역 allele indexing이 매우 비효율적일 수 있다. Hail은 이를 피하기 위해 `LGT`, `LAD`, `LPL`, `LA` 같은 local representation을 사용한다. 이 점은 대규모 분석에서 왜 VDS가 단순한 VCF의 압축판이 아닌지를 보여 주는 좋은 교육 소재다. Hail 문서가 `local_to_global` 변환을 조심하라고 경고하는 이유도 바로 여기에 있다. local 표현을 무심코 전역 표현으로 다시 펴면, 얻는 정보보다 늘어나는 데이터 크기가 훨씬 클 수 있기 때문이다.

Hail 분석 흐름 – import, annotation, QC, qualifying variant, association

실전에서는 Hail을 거창한 분산 이론보다 분석 단계별 흐름으로 배우는 편이 낫다. 첫 단계는 보통 `hl.import_vcf("s3://bucket/path/*` S3에서 데이터를 읽어 MatrixTable을 만드는 것이다. 다음으로 샘플 정보와 phenotype, annotation table을 결합하고, variant QC와 sample QC를 수행한다. 그다음 기능적 annotation과 필터링을 통해 qualifying variant를 정의하고, 마지막에 burden test나 association testing으로 이어진다. 이 흐름은 로컬이든 EMR이든 동일하다.

EMR이 필요한 이유는 이 분석 흐름의 각 단계가 더 큰 데이터와 더 많은 샘플을 만나면 갑자기 무거워지기 때문이다. import와 repartition, join과 aggregate, annotation lookup, cohort-level summary는 CPU뿐 아니라 메모리와 네트워크 셔플을 함께 요구한다. 따라서 Hail을 AWS에서 크게 돌릴 때는 "무조건 메모리형 인스턴스"보다 stage별 병목을 보는 감각이 중요하다. annotation join과 aggregation은 `r` 계열이, VCF import와 executor 확장이 중요한 단계는 `c` 계열이 더 잘 맞을 수 있다. Hail은 GPU 도구라기보다, CPU 노드 수와 적절한 분산 구조가 더 중요한 도구라는 점도 함께 강조하는 편이 좋다.

EMR on EC2, EMR on EKS, EMR Serverless 비교

2026년 현재 EMR은 하나의 실행 형태만 가진 서비스가 아니다. `EMR on EC2`, `EMR on EKS`, `EMR Serverless` 세 가지 큰 경로가 있다. `EMR on EC2`는 전통적인 형태로, 클러스터 구성을 가장 명시적으로 제어할 수 있고 Spark와 HDFS 문화에 익숙한 팀에게 잘 맞는다. `EMR on EKS`는 이미 Kubernetes 중심의 플랫폼을 운영하는 조직에서 유리하다. 반면 `EMR Serverless`는 클러스터를 직접 관리하고 싶지 않은 팀에게 매력적이지만, 장시간 실행과 큰 executor 조정이 필요한 일부 유전체 워크로드에서는 사전 검토가 더 필요하다. 따라서 학생에게는 "어느 것이 최신이라서 정답"이라고 가르치기보다, 조직이 이미 어떤 플랫폼을 갖고 있는지에 따라 선택이 달라진다고 설명하는 편이 낫다.

Table 1은 교육용으로 세 실행 형태를 비교한 것이다. 여기서 중요한 것은 기능 나열이 아니라, 운영 철학이 다르다는 점이다. 즉 `EMR on EC2`는 클러스터 제어권, `EMR on EKS`는 플랫폼 통합, `EMR Serverless`는 운영 단순화에 더 가깝다. Hail을 처음 배울 때는 보통 `EMR on EC2`로 이해를 시작하는 것이 가장 직관적이다. 하지만 연구실이나 기관이 이미 Kubernetes를 운영 중이라면 `EMR on EKS`가 더 자연스러울 수 있다. 기술 선택은 분석 규모만으로 결정되지 않으며, 조직이 이미 쓰고 있는 운영 방식과도 깊게 얽혀 있다.

Table 1. EMR의 세 가지 실행 형태

형태	장점	주의점	잘 맞는 상황
EMR on EC2	제어권이 크고 전통적 Spark 설명과 잘 맞음	클러스터 개념을 더 많이 이해해야 함	처음 Hail/EMR을 배우는 팀, Spark 중심 분석
EMR on EKS	Kubernetes 기반 플랫폼과 잘 통합됨	EKS 운영 경험이 없으면 오히려 복잡함	이미 EKS를 운영 중인 조직
EMR Serverless	클러스터 관리 부담이 적음	일부 장시간·대형 워크로드는 사전 검토 필요	간헐적 사용, 운영 단순화 우선 팀

로컬 Hail 튜토리얼을 AWS EMR로 확장하기

학생에게 가장 좋은 학습 경로는 로컬 튜토리얼의 개념을 먼저 익힌 뒤, 같은 개념을 AWS EMR로 확장하는 것이다. 예를 들어 `joonan-lab/hail_tutorial`에서 VCF import, MatrixTable 구조, QC, annotation, association testing을 익혔다면, EMR에서는 입력만 `s3://...`로 바꾸고 실행 환경을 Spark cluster로 넓히면 된다. 이때 개념은 그대로인데 규모와 운영 방식만 달라진다는 점이 중요하다. 클라우드는 새로운 분석법을 강요하는 곳이 아니라, 같은 분석을 더 큰 규모로 반복 가능하게 만드는 곳이다.

정리하면 Hail은 Spark 위에서 유전체 데이터를 다루기 쉽게 만든 분석 계층이고, EMR은 그 계층을 AWS에서 안정적으로 운영하도록 돕는 실행 계층이다. VDS는 대규모 코호트를 위한 데이터 표현이고, S3는 그 데이터를 두는 공유 저장 계층이다. 학생이 이 네 요소의 관계를 이해하면, gnomAD, All of Us, cohort-scale WGS 분석 같은 주제가 더 이상 막연한 "거대한 데이터"가 아니라 설계 가능한 분석 구조로 보이기 시작한다.

핵심 개념 정리

- EMR은 Spark/Hadoop 클러스터를 AWS에서 더 쉽게 운영하기 위한 관리형 계층이다.
- Hail의 `MatrixTable`과 `VDS`는 서로 다른 분석 목적을 가진 자료구조이며, 같은 것으로 보면 안 된다.
- VDS는 reference block과 variant data를 분리한 sparse representation이므로 cohort-scale 분석에 특히 유리하다.
- Hail을 AWS에서 크게 돌릴 때는 stage별 병목에 따라 인스턴스와 실행 형태를 고르는 감각이 중요하다.

복습 질문

1. EMR on EC2, EMR on EKS, EMR Serverless는 각각 어떤 조직 상황에 더 잘 맞는가?
2. VDS가 전통적 project VCF보다 유리한 이유를 저장과 질의 두 관점에서 설명해 보라.
3. 로컬 Hail 튜토리얼과 EMR 기반 Hail 분석은 무엇이 같고 무엇이 다른가?

Further Reading

- Hail. Amazon Web Services.
- Hail. Variant Dataset (VDS).
- joonan-lab. hail_tutorial.

References

- All of Us Research Program. 2024. The new VariantDataset (VDS) format for All of Us short-read WGS data.
- AWS. 2026a. What is Amazon EMR?.
- AWS. 2026b. Amazon EMR on EKS.
- AWS. 2026c. Amazon EMR Serverless.
- gnomAD Production Team. 2023. gnomAD v4.0.
- Hail. 2026d. Amazon Web Services.
- Hail. 2026e. Variant Dataset (VDS).
- He Q et al. 2025. The Scalable Variant Call Representation: Enabling Genetic Analysis Beyond One Million Genomes.

9장. AWS HealthOmics와 Nextflow 파이프라인

학습 목표

- AWS HealthOmics의 목적과 핵심 기능을 설명할 수 있다.
- Nextflow 기반 파이프라인을 관리형 서비스에서 돌리는 장점을 이해할 수 있다.
- 워크플로우 실행, 입력 데이터, 출력 저장의 흐름을 설명할 수 있다.

핵심 질문

- HealthOmics는 일반 EC2 운영과 무엇이 다른가
- Nextflow를 HealthOmics에 연결하면 어떤 점이 편해지는가
- 어떤 연구실은 HealthOmics가 맞고 어떤 연구실은 직접 운영이 나은가

관리형 오믹스 플랫폼의 등장 배경

유전체 분석을 AWS에서 운영할 때 가장 먼저 부딪히는 문제는 계산 자원 자체보다 운영 복잡성이다. 컨테이너를 어디에 둘지, 참조 유전체를 어떻게 배포할지, 샘플별 입력 manifest를 어떻게 만들지, 재시도와 로그를 어떻게 관리할지, 결과를 어떤 경로로 남길지 같은 문제가 빠르게 쌓인다. EC2와 Batch만으로도 충분히 강력한 시스템을 만들 수 있지만, 그만큼 사용자가 직접 이해하고 관리해야 하는 요소도 많다. AWS HealthOmics는 바로 이런 복잡성을 일부 흡수하기 위해 등장한 purpose-built omics 서비스다. 학생에게는 이를 `workflow` · 으로 이해시키는 편이 가장 명확하다.

HealthOmics를 설명할 때는 보통 `Storage`, `Workflows`, `Analytics` 세 축으로 구분하면 이해가 쉽다. 다만 현재형으로는 `Analytics`의 일부 기능이 신규 고객에게 제한되어 있다는 점을 함께 적어 두는 편이 안전하다. 중요한 것은 서비스 이름보다 역할이다. 즉 Storage는 read set과 reference를 다루는 데이터 운영 계층, Workflows는 WDL, Nextflow, CWL 기반의 실행 계층, Analytics는 구조화된 해석과 질의에 가까운 계층으로 이해하면 된다. 학생은 여기서 "관리형이므로 아무것도 몰라도 된다"는 인상을 받으면 안 된다. HealthOmics는 복잡성을 줄여 주지만, 컨테이너, 입력 구조, 권한, 참조 데이터 관리 같은 핵심 문제를 완전히 사라지게 하지는 않는다.

Figure 9-1은 HealthOmics의 세 계층을 가장 단순한 형태로 정리한 것이다. Storage는 시퀀싱 원시와 중간 산물을 표준화된 read set으로 보관하고, Workflows는 Nextflow/WDL/CWL 정의를 실제로 실행하고, Analytics는 annotation과 질의 가능한 테이블을 만드는 계층이다. 세 계층이 분리되어 있다는 점이 이 서비스의 핵심이다. 한 층에서 받은 데이터를 다음 층에서 자연스럽게 넘기는 흐름이 전체 운영을 단순하게 만든다.

Figure 9-1. AWS HealthOmics 3계층 구조

Sequence store, read set, reference store, run

HealthOmics의 Storage 계층은 단순 S3 버킷과는 조금 다른 운영 철학을 가진다. Sequence store는 `FASTQ`, `uBAM`, `BAM`, `CRAM` 같은 데이터를 read set 단위로 관리하며, ingest 과정에서 provenance와 ETag 같은 운영 메타데이터를 남긴다. 또한 read set 태그와 S3 tag propagation을 활용해 샘플 단위 운영 규칙을 설계할 수 있다. 학생에게는 이를 이 아니라 로 설명하는 편이 낫다. 즉 HealthOmics는 단순 저장을 넘어서 데이터의 상태와 출처를 함께 관리하려는 계층이다.

이 점은 시퀀싱 센터 데이터 handoff를 설명할 때 특히 유용하다. 시퀀싱 회사에서 파일이 도착하면, 연구실은 이를 그냥 폴더에 복사해 두는 대신 sequence store나 S3를 통해 명시적 운영 계층으로 등록할 수 있다. 이후 각 샘플은 read set으로 식별되고, 메타데이터 JSON이나 manifest와 연결되며, 워크플로 입력으로 넘겨진다. 또한 활성 read set은 S3 URI로 접근할 수 있으므로, IGV, HTSlib/samtools, Nextflow, CWL/WDL과 직접 연결되는 흐름도 만들 수 있다. 즉 sequence store는 단순 저장소가 아니라 에 가깝다.

Nextflow, Ready2Run, private workflow 연결 방식

HealthOmics Workflows 계층의 핵심은 표준 workflow 언어를 관리형 환경에서 실행할 수 있게 해 준다는 점이다. 공식 문서는 WDL, Nextflow, CWL을 지원하며, Nextflow는 특정 버전군과 DSL1/DSL2에 대한 지원 규칙을 제공한다. 초보자에게 중요한 것은 `Nextflow` `HealthOmics` 가 아니라, `Nextflow` 는 사실이다. 즉 관리형 서비스는 workflow 언어를 대체하지 않고, 그 실행 계층을 단순화한다.

그러나 관리형이라고 해서 workflow 정의를 이해하지 않아도 되는 것은 아니다. HealthOmics의 Nextflow 문서는 `publishDir`, `errorStrategy`, `maxRetries`, `omicsRetryOn5xx`, `time` directive 같은 구체적 동작을 명시적으로 다룬다. 이 점은 교육적으로 중요하다. 학생은 관리형 서비스가 숨겨 주는 복잡성과 숨겨 주지 못하는 복잡성을 구분해 배워야 한다. 컨테이너가 ECR에 어떻게 올라가 있는지, 입력 manifest가 어떤 구조인지, 어떤 step이 재시도 가능한지, 결과가 어디에 남는지 같은 질문은 여전히 사용자가 이해해야 한다.

Ready2Run workflows는 또 다른 진입 경로를 제공한다. 이는 third-party나 사전 구성된 workflow를 바로 사용할 수 있게 해 주는 형태로, 인프라 자체를 설계하기보다 결과 해석과 실험 운영에 집중하고 싶은 팀에 적합하다. 반면 private workflow는 연구실이 직접 관리하는 Nextflow pipeline을 HealthOmics에 올리는 방식이다. 따라서 교육용으로는 `Ready2Run =` , `private workflow =` 이라는 대비가 유용하다. 둘 중 어느 것이 더 낫다고 말하기보다, 연구실의 기술 역량과 반복 사용 패턴에 맞추어 고르는 방식으로 설명하는 편이 현실적이다.

다중 워크플로우 orchestration과 샘플 핸드오프

실제 연구실 운영에서는 샘플 하나짜리 파이프라인보다 여러 샘플과 여러 workflow가 연결된 구조가 더 흔하다. AWS의 `Orchestrating Multiple AWS HealthOmics Workflows at Scale` 사례는 이 점을 잘 보여 준다. 이 구조에서 S3의 metadata JSON 업로드는 단순 파일 저장이 아니라 이벤트의 시작점이다. Lambda와 Step Functions가 이를 감지하고, 샘플별 HealthOmics workflow와 QC workflow를 병렬로 실행한다. 즉 클라우드 네이티브 오믹스 운영은 "스크립트 하나를 돌린다"보다 "메타데이터 이벤트가 파이프라인을 움직인다"에 더 가깝다.

이런 구조는 시퀀싱 회사 데이터 handoff에도 매우 잘 맞는다. 예를 들어 vendor가 S3나 sequence store에 데이터를 전달하면, 연구실은 이를 read set으로 등록하고, 샘플 메타데이터를 JSON이나 manifest로 정리하고, 이를 기준으로 Nextflow 또는 Ready2Run workflow를 자동 실행할 수 있다. 그 뒤 QC와 결과가 S3에 쌓이고, 필요한 경우 Athena나 후속 분석 계층으로 넘어간다. 학생에게는 이 흐름을 `vendor upload -> registration -> metadata -> workflow -> QC -> downstream analysis`라는 일직선으로 보여 주는 것이 이해에 도움이 된다. 관리형 서비스의 진짜 가치는 버튼 클릭 수보다 이런 운영 흐름을 표준화하기 쉽게 만든다는 데 있다.

시퀀싱 회사 데이터 납품에서 자동 전처리까지

이 장에서 가장 실용적인 질문은 "AWS를 쓰면 시퀀싱 회사에서 나온 데이터를 바로 전처리할 수 있는가?"이다. 답은 가능하지만, 는 식으로 이해하면 곤란하다. 먼저 데이터가 어디로 들어오는지, sample metadata가 어떤 형식인지, 어떤 pipeline이 어떤 참조 유전체와 컨테이너를 쓰는지, 실패 시 누가 어떻게 재실행할지까지 정해야 한다. HealthOmics는 이 구조를 구현하는 데 필요한 많은 부분을 제공하지만, 운영 규칙 그 자체를 대신 설계해 주지는 않는다. 따라서 학생은 서비스 사용법과 함께 도 같이 배워야 한다.

여기서 data lake, lakehouse, warehouse 사고방식도 연결된다. sequence store나 S3에 원본 read set이 들어오는 단계는 data lake에 가깝다. workflow 결과가 Parquet나 annotation table처럼 반복 질의 가능한 구조로 변환되는 순간부터 lakehouse 사고가 시작된다. cohort dashboard나 임상 요약 리포트가 만들어지는 단계는 warehouse에 가깝다. 즉 HealthOmics 장은 단지 workflow 실행 장이 아니라, 데이터가 원본에서 구조화된 분석 자산으로 이동하는 과정을 보여 주는 장이기도 하다. 이 연결을 이해하면 뒤에서 나올 Athena, Bedrock, 통합 예제 장이 훨씬 자연스럽게 이어진다.

핵심 개념 정리

- HealthOmics는 `Storage + Workflows (+ Analytics)`라는 세 층으로 이해하는 것이 가장 쉽다.
- 관리형 서비스는 운영 부담을 줄여 주지만, 컨테이너, 입력 구조, 권한, 참조 데이터 관리를 대신 설계해 주지는 않는다.
- Sequence store와 read set은 단순 파일 저장을 넘어 provenance와 운영 메타데이터를 포함한 연구 데이터 계층이다.
- 시퀀싱 데이터 handoff에서 중요한 것은 파일 업로드 자체보다 metadata와 workflow orchestration의 연결이다.

복습 질문

1. HealthOmics가 EC2나 Batch 중심 직접 운영과 다른 점은 무엇인가?
2. Nextflow를 HealthOmics에 올릴 때 관리형이 숨겨 주는 복잡성과 여전히 사용자가 이해해야 하는 복잡성은 각각 무엇인가?
3. 시퀀싱 회사에서 도착한 데이터를 자동 전처리로 연결하려면 어떤 메타데이터와 운영 규칙이 필요한가?

Further Reading

- AWS. What is AWS HealthOmics?.
- AWS. Workflow definition specifics for Nextflow.
- AWS. Orchestrating multiple AWS HealthOmics workflows at scale.

References

- AWS. 2026a. What is AWS HealthOmics?.

- AWS. 2026b. AWS HealthOmics pricing.
- AWS. 2026c. Sequence stores.
- AWS. 2026d. Create a sequence store.
- AWS. 2026e. Accessing HealthOmics read sets with Amazon S3 URIs.
- AWS. 2026f. Workflow definition specifics for Nextflow.
- AWS. 2026g. Supported workflow language versions.
- AWS. 2026h. Ready2Run workflows.
- AWS Industries Blog. 2024. Orchestrating multiple AWS HealthOmics workflows at scale.

10장. htslib의 S3 접근과 파일 스트리밍

학습 목표

- htslib가 생물정보학 파일 접근의 핵심 라이브러리라는 점을 설명할 수 있다.
- htslib의 S3 plugin이 왜 중요한지 이해할 수 있다.
- BAM, CRAM, VCF 같은 파일을 전체 다운로드 없이 부분 접근하는 개념을 설명할 수 있다.
- 인덱스 기반 랜덤 접근이 클라우드 비용과 성능에 어떤 차이를 만드는지 이해할 수 있다.

핵심 질문

- 왜 많은 생물정보학 도구가 htslib 위에 서 있는가
- S3 plugin은 기존 파일 함수 호출과 어떻게 연결되는가
- 인덱스 기반 랜덤 접근은 클라우드에서 왜 특히 중요해지는가
- 전체 복사보다 스트리밍이 더 적합한 상황과 그렇지 않은 상황은 어떻게 구분하는가

htslib는 왜 “보이지 않는 표준”인가

생물정보학 입문자는 보통 `samtools`, `bcftools`, `tabix`, `pysam` 같은 도구 이름부터 배우지만, 실제 분석 현장에서 더 중요한 것은 그 밑바닥을 공통으로 받치고 있는 라이브러리 계층이다. `htslib`는 바로 그 공통 계층이다. HTSlib 논문은 이 라이브러리를 SAM, BAM, CRAM, VCF, BCF, FASTA 같은 고처리량 시퀀싱 데이터 형식에 대한 읽기·쓰기와 인덱싱을 제공하는 통합 C 라이브러리로 설명한다. 즉 `samtools`는 정렬 파일 조작 도구이고, `bcftools`는 변이 파일 조작 도구이지만, 실제 파일을 열고 압축 블록을 읽고 인덱스를 따라가며 원격 자원을 처리하는 핵심 기능의 상당 부분은 htslib가 담당한다 (Bonfield et al. 2021).

이 구조가 중요한 이유는 도구 하나하나를 따로 고치는 대신, 공통 라이브러리 하나를 개선함으로써 생태계 전체의 능력이 함께 올라가기 때문이다. 예를 들어 로컬 디스크 파일만 다루던 도구 집합이 있다고 하자. 이 도구들이 모두 htslib의 파일 접근 계층을 사용한다면, htslib가 HTTP, HTTPS, S3 같은 원격 백엔드를 이해하는 순간 같은 생태계에 속한 도구들도 비교적 적은 수정으로 원격 데이터 접근 능력을 얻게 된다. 학생이 이 점을 이해하면 “왜 어떤 프로그램은 클라우드 파일을 바로 열고, 어떤 프로그램은 꼭 먼저 복사해야 하는가”라는 차이를 도구 이름이 아니라 라이브러리 계층의 차이로 읽을 수 있게 된다.

또한 htslib는 파일 형식을 읽는 파서에 그치지 않는다. 압축, 인덱스, 좌표 기반 질의, 참조 서열 접근, 원격 전송 프로토콜까지 함께 다룬다. 이 때문에 htslib를 배운다는 것은 단순히 C 라이브러리 하나를 소개받는 것이 아니라, 현대 유전체 분석이 파일을 어떻게 표현하고 어디에서 어떻게 읽는지에 대한 운영 원리를 배우는 일에 가깝다. 1장에서 보았던 `bring compute to data`라는 원칙이 실제 생물정보학 도구 내부에서 어떻게 구현되는지를 보여 주는 대표 사례가 바로 htslib다.

Table 1은 학생이 자주 접하는 도구와 htslib의 관계를 단순화한 것이다. 중요한 점은 “모든 도구가 정확히 같은 방식으로 htslib를 쓴다”는 뜻이 아니라, 공통 파일 접근 계층이 넓게 공유된다는 사실이다. 이 공통 계층 덕분에 S3 plugin의 의미도 단순한 부가 기능이 아니라 생태계 수준의 변화로 읽을 수 있다.

Table 1. htslib 생태계의 기본 구성

계층	대표 도구 또는 형식	역할
응용 도구	`samtools`, `bcftools`, `tabix`, `pysam`, `rsamtools`	사용자가 직접 실행하거나 호출하는 분석 도구
공통 라이브러리	`htslib`	파일 열기, 압축 블록 처리, 인덱스 탐색, 원격 접근
데이터 형식	SAM, BAM, CRAM, VCF, BCF, BGZF, FASTA	정렬, 변이, 참조 서열, 압축 표현
저장 위치	로컬 디스크, HTTP/HTTPS, S3, HealthOmics S3 URI	데이터가 실제로 놓이는 위치

BGZF, 인덱스, 랜덤 접근의 기본 원리

클라우드 스트리밍을 이해하려면 먼저 왜 BAM이나 `vcf.gz`가 "부분 읽기"에 적합한지 알아야 한다. 여기서 핵심은 BGZF와 인덱스다. HTSlib 논문은 BGZF를 GZIP과 호환되면서도 블록 단위 접근을 허용하는 압축 방식으로 설명한다. 일반적인 gzip 파일은 앞에서부터 순서대로 풀어야 하므로 파일 뒤쪽의 특정 좌표만 읽고 싶어도 사실상 앞부분을 모두 지나가야 한다. 반면 BGZF는 압축 데이터가 비교적 작은 블록들로 나뉘어 있어, 인덱스가 가리키는 블록부터 바로 찾아갈 수 있다. 이것이 유전체 좌표 기반 질의가 가능한 기술적 출발점이다 (Bonfield et al. 2021).

인덱스는 이 출발점을 실제 분석 동작으로 바꾸는 지도다. BAM에는 보통 `BAI`나 `CSI`, BGZF 압축 VCF에는 `TBI`나 `CSI`, CRAM에는 `CRAI`나 `CSI`가 함께 사용된다. 이 인덱스는 "염색체의 어느 구간이 파일의 어느 압축 블록 근처에 있는가"를 기록한다. 따라서 사용자가 `chr1:1000000-1010000` 같은 범위를 요청하면, 도구는 파일 전체를 읽지 않고도 관련 블록만 찾아가서 필요한 부분만 해제할 수 있다. 학생은 이 원리를 "파일 전체를 여는 것"이 아니라 "좌표를 바이트 범위로 번역하는 것"으로 이해하는 편이 좋다.

이 지점에서 로컬 접근과 클라우드 접근이 연결된다. 로컬 디스크에서는 인덱스가 디스크 seek를 줄여 준다. 클라우드 객체 스토리지에서는 같은 원리가 네트워크 전송량을 줄여 준다. 즉 인덱스 기반 랜덤 접근은 원래도 중요했지만, S3 같은 원격 저장소에서는 그 가치가 훨씬 커진다. 분석자가 한 유전자 주변만 보고 싶다면, 수백 기가바이트 BAM 전체를 복사하는 대신 필요한 바이트 범위만 요청하는 편이 훨씬 합리적이기 때문이다. 이때 파일 형식의 설계와 저장 방식의 설계가 맞물리면서 "클라우드 친화적" 접근이 가능해진다.

Table 2는 자주 쓰는 형식과 인덱스의 관계를 학습용으로 정리한 것이다. 실제 현장에서는 더 많은 예외와 세부 옵션이 있지만, 입문 단계에서는 어떤 형식이 어떤 인덱스를 통해 부분 읽기를 지원하는지만 먼저 이해해도 충분하다. 특히 인덱스가 없으면 원격 스트리밍의 장점이 크게 줄어든다는 점을 먼저 익혀 두자.

Table 2. 대표 형식과 인덱스 기반 부분 접근

데이터 형식	흔한 압축 또는 저장 방식	대표 인덱스	부분 접근의 의미
BAM	BGZF 기반 바이너리 정렬 파일	`BAI`, `CSI`	특정 유전체 구간의 read만 빠르게 조회
CRAM	참조 기반 압축 정렬 파일	`CRAI`, `CSI`	더 작은 저장 크기와 구간 단위 접근
VCF.gz	BGZF 압축 텍스트 변이 파일	`TBI`, `CSI`	특정 염색체 구간의 variant만 조회
BCF	바이너리 변이 파일	`CSI`	대규모 cohort 변이 질의에 유리
tab-delimited genomics file	BGZF + 좌표 정렬	`TBI`, `CSI`	BED, GFF 같은 좌표 파일의 구간 조회

S3 plugin은 무엇을 바꾸었는가

htslib의 S3 plugin은 겉보기에는 단순하다. 문서에 따르면 이 plugin은 `s3://bucket/path/to/file` 형태의 URL을 이해하고, 필요한 인증 정보는 URL, 환경 변수, AWS 자격 증명 파일, 또는 `s3cfg` 파일에서 가져올 수 있다. 그러나 교육적으로는 이 간단함이야말로 핵심이다. 사용자는 더 이상 "먼저 S3에서 로컬로 복사한 뒤 도구를 실행한다"는

사고방식에만 머물 필요가 없다. 응용 프로그램이 htslib의 파일 함수를 쓰고 있다면, 같은 함수 호출이 로컬 파일 경로뿐 아니라 S3 URI에도 적용될 수 있기 때문이다 (HTSlib 2025).

이 변화는 기술적 번역 계층 덕분에 가능하다. `samtools view local.bam`과 `samtools view s3://bucket/sample.bam`은 사용자가 보기에는 입력 문자열만 다르지만, 내부에서는 htslib가 적절한 backend를 선택해 파일 열기, 인증, 요청, 블록 읽기, 인덱스 탐색을 처리한다. 즉 S3 plugin은 "S3 전용 새 프로그램"을 만드는 것이 아니라, 기존 프로그램이 파일 API를 그대로 유지한 채 S3를 이해하게 만드는 방식이다. 이런 설계는 생물정보학에서 특히 강력하다. 많은 연구용 도구가 파일 경로를 인자로 받는 오래된 인터페이스를 유지하고 있기 때문이다.

인증과 권한 측면에서도 plugin은 클라우드 운영 원칙과 잘 맞는다. htslib S3 plugin manual은 환경 변수와 `~/.aws/credentials`, `AWS_PROFILE`, 임시 자격 증명, 만료 시간 갱신 같은 메커니즘을 설명한다. 이는 액세스 키를 코드에 하드코딩하지 않고, 실행 환경에 맞는 자격 증명 체계를 사용하라는 2장의 원칙과 정확히 연결된다. 다시 말해 S3 plugin의 가치는 단순히 원격 파일을 여는 편의성에 있지 않고, AWS 권한 모델과 연구 도구를 자연스럽게 연결해 준다는 데도 있다.

최근 AWS HealthOmics 문서는 이 점을 더욱 실용적으로 보여 준다. HealthOmics는 활성 read set을 Amazon S3 URI로 접근할 수 있게 하며, 이 문서에서 HTSlib 1.20 이상은 Amazon S3 Access Point를 원활하게 지원한다고 설명한다. 구버전에서는 `HTS_S3_HOST` 설정이나 presigned URL, 혹은 Mountpoint 같은 우회 경로가 필요할 수 있지만, 최신 라이브러리에서는 S3 URI 접근이 더 직접적인 표준 경로가 되었다. 이는 S3 plugin이 더 이상 실험적 보조 기능이 아니라, 클라우드 오믹스 도구 체계의 정식 일부가 되었음을 보여 준다 (AWS HealthOmics 2026).

BAM, CRAM, VCF를 클라우드에서 스트리밍한다는 것

이제 개념을 실제 분석 장면으로 바꿔 보자. 연구자가 BAM 파일을 S3에 두고 특정 유전자 근처 정렬 상태만 확인하고 싶을 때, 이상적인 경로는 간단하다. 인덱스가 함께 존재하고, 도구가 htslib를 통해 S3에 접근할 수 있다면, 필요한 좌표 범위만 읽어 와서 즉시 화면이나 파이프라인 다음 단계로 넘기면 된다. 이때 중요한 것은 "원격 파일을 열었다"가 아니라 "원격 파일의 필요한 일부만 읽었다"는 점이다. 클라우드 스트리밍의 경제성은 바로 여기에 있다.

CRAM에서는 한 가지 생각을 더해야 한다. CRAM은 참조 기반 압축 형식이므로, 구간을 해석하려면 참조 서열 접근 전략도 함께 설계해야 한다. 참조 FASTA를 로컬 캐시에 둘 수도 있고, 같은 S3나 공유 파일시스템에서 읽을 수도 있다. 따라서 CRAM 스트리밍은 BAM보다 저장 효율이 좋을 수 있지만, 참조 자원 위치까지 포함한 운영 설계가 필요하다. 학생은 이를 형식 비교 문제로만 볼 것이 아니라, + + 라는 세 요소가 함께 움직이는 시스템 문제로 이해해야 한다.

VCF와 BCF의 경우도 논리는 같다. 브라우저나 `bcftools`가 특정 locus의 변이만 확인할 때, 실제로는 `vcf.gz` 전체를 모두 옮기는 것이 아니라 인덱스를 따라 필요한 BGZF 블록만 읽어 온다. 따라서 cohort-scale VCF를 다룰수록 "클라우드에 두고 필요한 범위만 읽는 전략"이 더욱 자연스러워진다. 특히 공용 참조 변이 자원이나 반복해서 재사용되는 annotation 자산은 매번 프로젝트별로 복제하기보다, 공용 위치에 두고 질의하는 편이 훨씬 운영 친화적이다.

Table 3은 대표 파일 형식에서 스트리밍 접근이 특히 유리한 장면을 정리한 것이다. 이 표를 외우기보다, 어떤 분석 질문이 인지, 어떤 질문이 인지를 구분하는 훈련이 더 중요하다.

Table 3. 클라우드 스트리밍이 잘 맞는 대표 사례

파일 형식	필요한 부속 자산	스트리밍이 유리한 작업	주의할 점
BAM	`.bai` 또는 `.csi`	특정 locus read 점검, QC spot check, IGV 시각화	인덱스 없으면 장점이 크게 줄어듦
CRAM	`.crai` 또는 `.csi`, 참조 FASTA	대형 정렬 자산의 저장비 절감과 구간 조회	참조 접근 경로를 함께 설계해야 함
VCF.gz	`.tbi` 또는 `.csi`	유전자 주변 variant 확인, 대상 구간 annotation	압축과 인덱스가 함께 있어야 효율적
BCF	`.csi`	cohort-scale binary variant 질의	하위 도구의 BCF 지원 범위를 확인해야 함

언제는 스트리밍하고, 언제는 복사해야 하는가

클라우드 스트리밍이 항상 정답은 아니다. 가장 잘 맞는 상황은 분석 질문이 좁고, 파일이 크며, 여러 사용자가 같은 공용 자산을 반복해서 참조할 때다. 예를 들어 특정 변이 주변 read를 확인하거나, 패널 유전자 목록만 빠르게 훑거나, 브라우저에서 일부 구간만 시각화하는 작업은 스트리밍이 매우 자연스럽다. 같은 맥락에서 대형 reference BAM, CRAM, VCF를 여러 작업자가 공유한다면, 매번 로컬 복사본을 만들기보다 공용 S3 경로에서 필요한 부분만 읽는 편이 훨씬 간결하다.

반대로 전체 파일을 여러 번 순차 스캔해야 하거나, 도구가 htslib의 원격 접근을 제대로 지원하지 않거나, 같은 파일을 매우 자주 재사용하면서 네트워크 왕복이 병목이 되는 경우에는 한시적 로컬 staging이 더 나을 수 있다. 예를 들어 대규모 전수 계산에서 입력 파일 전체를 반복해 읽는다면, 한 번 내려받아 EBS나 로컬 scratch에 두는 편이 더 예측 가능한 성능을 보일 수 있다. 즉 문제는 "스트리밍 대 다운로드"의 이념 대결이 아니라, 접근 패턴과 반복 횟수와 도구 호환성의 문제다.

교육적으로 가장 중요한 결론은 다음과 같다. htslib의 S3 접근은 파일을 어디에 둘 것인가의 문제를 다시 쓰게 만들었다. 예전에는 클라우드 객체를 분석 도구가 직접 읽지 못하면, 사실상 모든 데이터 전략이 로컬 복사로 수렴했다. 이제는 공용 자산은 S3에 두고, 인덱스를 붙이고, 필요한 부분만 읽고, 정말 필요할 때만 로컬로 stage하는 혼합 전략이 훨씬 자연스럽다. 이 전환을 이해하면 다음 장에서 다룰 S3 마운트 도구가 왜 편리하면서도 근본 해결책은 아닐 수 있는지도 더 잘 보이게 된다.

핵심 개념 정리

- htslib는 `samtools`, `bcftools`, `tabix`, `pysam` 등 많은 도구가 공유하는 파일 접근 핵심 계층이다.
- BGZF와 인덱스는 유전체 좌표를 파일의 특정 블록으로 연결하여 부분 읽기와 랜덤 접근을 가능하게 한다.
- htslib의 S3 plugin은 기존 파일 함수 호출을 S3 URI까지 확장함으로써 많은 도구가 큰 수정 없이 클라우드 파일을 읽게 해 준다.
- 클라우드 스트리밍의 핵심 이점은 전체 다운로드 회피가 아니라, 필요한 구간만 읽어 비용과 시간을 줄이는 데 있다.
- 스트리밍은 구간 질의와 공용 자산 재사용에 강하지만, 반복적 전체 스캔이나 비호환 도구에는 로컬 staging이 더 적합할 수 있다.

복습 질문

1. htslib가 "보이지 않는 표준"이라고 불릴 수 있는 이유를 도구 생태계 관점에서 설명해 보라.
2. BGZF와 인덱스는 왜 S3 같은 객체 스토리지 환경에서 특히 더 중요해지는가?
3. BAM, CRAM, VCF를 클라우드에서 직접 읽을 때 각각 어떤 부속 자산이 필요하며, 그 이유는 무엇인가?
4. htslib 기반 직접 스트리밍과 로컬 복사 전략은 어떤 기준으로 선택해야 하는가?

Further Reading

- HTSlib. htslib-s3-plugin manual.
- AWS HealthOmics. Accessing HealthOmics read sets with Amazon S3 URIs.
- Samtools. HTSlib GitHub repository.

References

- AWS HealthOmics. 2026. Accessing HealthOmics read sets with Amazon S3 URIs.
- Bonfield JK, Marshall J, Danecek P, Li H, Ohan V, Whitwham A, Keane T, Davies RM. 2021. HTSlib: C library for reading/writing high-throughput sequencing data.
- HTSlib. 2025. htslib-s3-plugin manual page.
- Samtools/GA4GH. 2026. SAM/BAM and related specifications.

11장. goofys, s3fs, s3fuse와 S3 마운트 도구

학습 목표

- S3를 파일시스템처럼 보이게 하는 도구의 목적을 설명할 수 있다.
- goofys, s3fs, s3fuse 계열 도구의 차이와 한계를 이해할 수 있다.
- “마운트해서 쓰는 방식”과 “애플리케이션이 직접 S3를 읽는 방식”을 비교할 수 있다.
- POSIX 파일시스템과 객체 스토리지의 차이가 왜 워크플로 동작에 영향을 주는지 설명할 수 있다.

핵심 질문

- 왜 연구자들은 S3를 로컬 디렉터리처럼 보이게 하려 하는가
- 어떤 상황에서는 편리하지만 어떤 상황에서는 위험한가
- 생물정보학 도구와 잘 맞는 경우와 잘 맞지 않는 경우는 무엇인가
- 직접 S3를 읽는 도구가 있는데도 마운트 도구가 계속 쓰이는 이유는 무엇인가

S3를 “마운트”하고 싶어지는 이유

S3는 본질적으로 객체 스토리지다. 즉 전통적인 리눅스 파일시스템처럼 디렉터리, inode, 파일 잠금, 부분 수정, 원자적 rename을 기본 전제로 설계된 저장소가 아니다. 그럼에도 많은 연구자가 S3를 로컬 디렉터리처럼 마운트하고 싶어 하는 이유는 분명하다. 분석 도구의 상당수가 여전히 POSIX 파일 경로를 중심으로 설계되어 있고, 스크립트와 워크플로도 `open("/path/to/file")` 패턴을 전제로 하기 때문이다. 학생이 처음 클라우드에 올라왔을 때 가장 먼저 느끼는 불편함도 바로 이것이다. “S3에 파일은 있는데, 내 도구는 왜 버킷 URI를 그대로 못 읽는가?”라는 질문은 매우 자연스럽다.

이때 등장하는 것이 FUSE 기반 마운트 도구다. FUSE는 사용자 공간에서 파일시스템 인터페이스를 구현하게 해 주는 계층이다. 다시 말해 커널과 애플리케이션 사이에 번역기를 하나 두고, 애플리케이션은 평범한 파일시스템처럼 보이는 경로를 사용하지만 실제 데이터는 S3 API를 통해 읽고 쓰게 만드는 방식이다. 따라서 마운트 도구의 핵심 가치는 성능보다 호환성에 있다. 애플리케이션을 크게 고치지 않고도 S3 객체를 파일처럼 취급할 수 있게 해 주기 때문이다.

그러나 이 편의성은 오해를 부르기 쉽다. S3를 마운트했다고 해서 S3가 갑자기 POSIX 파일시스템으로 변신하는 것은 아니다. 마운트 도구는 어디까지나 번역 계층이다. 이 번역이 잘 맞는 워크로드에서는 매우 유용하지만, 객체 스토리지와 파일시스템의 의미 차이가 크게 드러나는 워크로드에서는 쉽게 성능 저하나 예상 밖 동작이 발생한다. 따라서 이 장의 출발점은 “마운트는 마법이 아니라 적응 계층”이라는 사실을 분명히 이해하는 데 있다.

FUSE 기반 S3 마운트는 실제로 무엇을 번역하는가

전통적인 파일시스템에서 프로그램은 파일을 열고, 임의 위치로 이동하고, 일부 바이트를 덮어쓰고, 임시 파일을 만든 뒤 rename으로 교체하고, 파일 잠금으로 동시 접근을 조정한다. 객체 스토리지에서는 이 모델이 그대로 성립하지 않는다. S3의 기본 단위는 디렉터리가 아니라 객체이고, rename은 보통 “복사 후 삭제”로 흉내 내야 하며, 객체 일부를 수정하는 대신 새 객체를 다시 써야 하는 경우가 많다. 따라서 FUSE 기반 마운트 도구는 단순히 경로 표시만 바꾸는 것이 아니라, 파일시스템 호출을 객체 스토리지 호출로 계속 번역하는 부담을 떠안는다.

이 차이는 메타데이터와 일관성에서도 드러난다. 파일시스템에서는 디렉터리 listing, 파일 크기 확인, 권한 비트, 수정 시간 같은 정보가 운영체제 수준에서 매우 자연스럽게 다뤄진다. 반면 S3에서는 listing이 곧 API 호출이고, 디렉터리는 실제 객체가 아니라 prefix 개념이며, 파일 권한과 잠금 의미론도 완전히 같지 않다. AWS의 Mountpoint for Amazon S3 문서가 “이 도구는 클라우드 네이티브 워크로드를 위해 설계되었으며, 파일 시스템 전체 POSIX 의미론을 제공하지 않는다”고 분명히 밝히는 이유가 여기에 있다. 즉 마운트 도구를 쓰는 순간 사용자는 편의성을 얻는 대신, 일부 의미론을 포기하거나 우회하는 구조 위에 올라서게 된다 (AWS 2026a).

학생에게는 이 점을 “S3를 붙이는 일”보다 “호출을 번역하는 일”로 설명하는 편이 훨씬 효과적이다. 그래야 왜 작은 파일이 많은 워크로드에서 요청 수가 폭증할 수 있는지, 왜 rename-heavy 파이프라인이 비효율적일 수 있는지, 왜 SQLite나 lock 파일을 많이 쓰는 소프트웨어가 문제를 일으킬 수 있는지 자연스럽게 이해할 수 있다. 마운트 도구는 접근성을 높여 주지만, 객체 스토리지의 본질을 지워 주지는 않는다.

goofys, s3fs, s3fuse 계열, Mountpoint 비교

현장에서 자주 언급되는 도구는 크게 세 부류로 나눌 수 있다. 첫째, `goofys`는 높은 처리량과 단순성을 목표로 한 S3 중심 마운트 도구다. 공식 README는 goofys가 `close-to-open consistency`, 부분 읽기, multipart upload, 디렉터리 이름 rename을 지원하지만, 하드링크나 심볼릭 링크, 임의 위치 쓰기 같은 완전한 POSIX 기능은 지원하지 않는다고 설명한다. 즉 goofys는 S3를 최대한 빠르게 읽고 쓰는 데 초점을 맞춘 대신, 파일시스템 호환성을 일부 포기한 도구라고 이해하면 된다 (goofys 2025).

둘째, `s3fs-fuse`는 더 오래되고 널리 알려진 범용 계열이다. 커뮤니티에서는 `s3fuse`라는 표현을 넓게 써서 이런 S3 FUSE 마운터 전반을 가리키는 경우가 많다. 공식 저장소는 s3fs가 FUSE를 통해 S3 버킷을 로컬 파일시스템처럼 마운트하며, 대형 객체용 multipart upload, 서버 측 암호화, IAM Role, requester pays, 커스텀 endpoint 같은 폭넓은 옵션을 제공한다고 설명한다. 즉 호환성과 설정 유연성 면에서는 강점이 있지만, 그만큼 운영자가 이해해야 할 옵션과 성능 trade-off도 늘어난다 (s3fs-fuse 2026).

셋째, `Mountpoint for Amazon S3`는 AWS가 최근 제시하는 클라우드 네이티브 방향이다. 공식 문서는 Mountpoint가 S3 객체를 파일처럼 접근하게 해 주며, 높은 읽기 처리량과 예측 가능한 성능을 목표로 한다고 설명한다. 동시에 기존 객체는 읽기 전용이고, 새 파일 생성은 가능하지만 임의 수정, 디렉터리 rename, 심볼릭 링크, 파일 잠금 같은 완전한 POSIX 기능은 지원하지 않는다고 명시한다. 즉 Mountpoint는 "S3를 진짜 파일시스템으로 흉내 내는 범용 번역기"라기보다, 읽기 중심 대용량 워크로드에 최적화된 S3 전용 접근 계층에 가깝다 (AWS 2026a; AWS 2026b).

Table 1은 세 부류를 교육용으로 단순 비교한 것이다. 실제 선택에서는 운영체제, 커널 버전, 캐시 정책, 인증 방식, 작업 패턴까지 더 살펴야 하지만, 초급자 단계에서는 무엇을 최적화한 도구인지부터 먼저 읽는 것이 중요하다.

Table 1. 대표적인 S3 마운트 도구 비교

도구	주된 목표	장점	약점 또는 제약	잘 맞는 작업
`goofys`	속도와 단순성 중심 S3 마운트	부분 읽기와 비교적 가벼운 동작	완전한 POSIX 의미론을 제공하지 않음	대형 읽기 중심 데이터 접근, 간단한 파이프라인
`s3fs-fuse` / `s3fuse` 계열	범용 호환성과 옵션 유연성	다양한 인증·암호화·endpoint 지원	설정 복잡도와 성능 변동성이 큼	레거시 도구 연결, 범용 마운트 실험
`Mountpoint for Amazon S3`	고처리량 클라우드 네이티브 읽기	AWS 최적화, 대형 읽기 워크로드에 강함	기존 객체 수정, rename, 잠금 등 제약이 큼	읽기 중심 배치 분석, 공유 reference 자산 접근

편의성 뒤에 있는 성능과 의미론의 함정

마운트 도구가 실전에서 문제를 일으키는 이유는 대개 "느리다"라는 한마디로 설명되지만, 실제로는 더 구조적인 이유가 있다. 첫째, 작은 파일이 매우 많을 때 listing과 stat 호출이 폭증한다. 객체 스토리지에서는 디렉터리 내용을 확인하는 일 자체가 원격 API 요청이므로, 수만 개의 조각 파일을 순식간에 훑는 워크플로는 로컬 파일시스템보다 훨씬 큰 부담을 만들 수 있다. 둘째, rename과 임시 파일 교체가 많은 워크플로는 객체 복사와 삭제를 반복하게 되어 예상보다 느려질 수 있다. 셋째, random write, mmap, lock 파일 같은 동작은 마운트 계층이 가장 버거워하는 패턴이다.

이 문제는 생물정보학에서 특히 자주 드러난다. 많은 도구가 중간 파일을 만들고, 정렬 후 rename하고, 인덱스를 새로 쓰고, lock 파일이나 임시 디렉터리를 적극 사용하기 때문이다. 예를 들어 정렬 중간 파일과 sort shard가 대량으로 발생하는 워크플로는 S3 마운트보다 EBS나 로컬 NVMe scratch가 훨씬 낫다. 반대로 이미 완성된 대형 reference FASTA, annotation bundle, 결과 BAM을 읽기 전용으로 여러 노드가 공유하는 장면에서는 Mountpoint나 goofys가 꽤 실용적일 수 있다. 즉 마운트 도구의 적합성은 "S3를 쓸 수 있느냐"가 아니라 "워크로드가 파일시스템 의미론을 얼마나 강하게 요구하느냐"에 달려 있다.

AWS는 2020년 12월부터 S3의 strong read-after-write consistency를 제공하지만, 이것이 POSIX 파일시스템과 동일하다는 뜻은 아니다. 객체 수준 일관성이 좋아졌다고 해서 rename 비용이 사라지는 것도 아니고, 파일 잠금이 생기는 것도 아니다. 따라서 초보자는 "S3도 이제 강한 일관성이니까 파일시스템처럼 써도 된다"고 오해해서는 안 된다. 강한 일관성은 중요한 개선이지만, 객체 스토리지와 파일시스템의 의미 차이를 지워 주는 마법은 아니다.

htslib의 직접 접근과 비교하면 무엇이 다른가

10장에서 본 htslib의 S3 접근은 애플리케이션이 S3를 직접 이해하는 방식이었다. 이 방식의 장점은 분명하다. 도구가 BAM, CRAM, VCF의 구조와 인덱스를 알고 있으므로, 필요한 바이트 범위만 요청하고 파일시스템 의미론을 억지로 흉내 낼 필요가 없다. 반대로 마운트 방식은 애플리케이션이 S3를 모를 때도 기존 경로 인터페이스를 유지할 수 있다는 장점이 있지만, 그 대가로 번역 비용과 의미론 차이를 떠안는다. 따라서 두 방식은 대체 관계가 아니라 서로 다른 문제를 푸는 선택지다.

Table 2는 이 차이를 단순화한 것이다. 학생이 가장 먼저 익혀야 할 원칙은 "S3를 직접 읽을 수 있는 도구라면 직접 읽는 편이 대체로 더 자연스럽다"는 점이다. 마운트는 주로 레거시 호환성과 전환기 편의성을 위해 쓰는 보조 수단으로 보는 편이 정확하다.

Table 2. 직접 S3 접근과 마운트 방식의 비교

질문	애플리케이션 직접 접근	S3 마운트 도구
누가 S3를 이해하는가	애플리케이션 또는 라이브러리	FUSE 번역 계층
부분 읽기 최적화	파일 형식과 인덱스를 알고 최적화하기 쉬움	파일시스템 호출을 S3 요청으로 바꾸므로 비효율 가능
레거시 도구 호환성	도구가 URI를 알아야 함	경로만 있으면 연결 가능
POSIX 기대와의 충돌	적음	높음
권장 장면	BAM/CRAM/VCF 직접 질의, object-native workflow	읽기 중심 reference 공유, 빠른 이식, 제한적 레거시 지원

연구 워크플로우에서의 현실적인 사용 지점

그렇다면 연구 현장에서는 언제 마운트 도구를 써야 할까. 가장 현실적인 답은 "완성된 읽기 중심 자산을 빠르게 연결할 때"다. 예를 들어 대형 reference genome, 알려진 variant resource, annotation bundle, 이미 끝난 결과 BAM과 CRAM을 여러 노드에서 읽기 전용으로 공유하는 장면에서는 마운트 도구가 꽤 유용할 수 있다. AWS HPC 블로그도 2026년 1월 Mountpoint가 Nextflow on AWS Batch 워크플로에서 입력 데이터를 stage하지 않고 바로 읽는 패턴을 소개한다. 즉 모든 입력을 로컬 scratch로 복사하는 오랜 관행을 줄이고 싶을 때, 특히 읽기 위주의 대형 객체에는 마운트 도구가 실용적일 수 있다 (AWS 2026c).

반면 쓰기 중심 작업, 빈번한 rename, 데이터베이스 파일, 임시 파일이 많은 정렬과 정리 단계, 혹은 수많은 작은 파일을 지속적으로 생성하는 파이프라인에는 마운트 도구를 기본 경로로 삼지 않는 편이 안전하다. 이 경우는 S3를 최종 저장소로 두되, 계산 중간 단계는 EBS, instance store, FSx for Lustre 같은 파일시스템 성격이 강한 계층으로 분리하는 편이 더 예측 가능하다. 결국 좋은 설계는 "편해서 마운트한다"가 아니라, S3 , 는 구분에서 나온다.

goofys, s3fs, s3fuse 계열, Mountpoint는 모두 S3와 기존 파일 중심 소프트웨어 사이의 간극을 메우는 도구다. 하지만 이 간극을 메운다고 해서 두 세계가 완전히 같아지는 것은 아니다. 클라우드 네이티브 분석으로 갈수록 애플리케이션 직접 접근 방식이 더 자연스러워지고, 마운트 도구는 전환기 호환성이나 읽기 중심 자산 공유에 더 적합한 위치로 정리된다. 이 경계를 잡아 두면 편의성과 안정성 사이에서 훨씬 더 나은 선택을 할 수 있다.

핵심 개념 정리

- S3 마운트 도구는 객체 스토리지를 POSIX 파일시스템처럼 완전히 바꾸는 것이 아니라, 호출을 번역하는 계층이다.
- `goofys`는 단순성과 처리량에, `s3fs-fuse` 계열은 범용 호환성에, `Mountpoint`는 읽기 중심 클라우드 네이티브 성능에 상대적으로 초점을 둔다.
- 작은 파일이 많고 rename이나 random write가 많은 워크로드는 S3 마운트와 잘 맞지 않는다.
- BAM, CRAM, VCF처럼 애플리케이션이 직접 S3를 이해할 수 있는 경우에는 마운트보다 직접 접근이 대체로 더 자연스럽다.
- 마운트 도구는 읽기 전용 공유 자산이나 레거시 소프트웨어 연결에 특히 유용하다.

복습 질문

1. FUSE 기반 S3 마운트 도구를 “번역 계층”이라고 부르는 이유는 무엇인가?
2. `goofys`, `s3fs-fuse`, `Mountpoint for Amazon S3`는 각각 무엇을 더 중요하게 최적화하는가?
3. 왜 strong consistency가 도입되었다고 해서 S3를 POSIX 파일시스템처럼 다뤄도 된다는 뜻은 아닌가?
4. 유전체 워크플로에서 어떤 단계는 S3 마운트에 적합하고, 어떤 단계는 EBS나 scratch 디스크가 더 적합한가?

Further Reading

- AWS. Mountpoint for Amazon S3 user guide.
- goofys. Official GitHub repository.
- s3fs-fuse. Official GitHub repository.

References

- AWS. 2026a. Mountpoint for Amazon S3.
- AWS. 2026b. Mountpoint file system behavior and limitations.
- AWS. 2026c. Optimize Nextflow workflows on AWS Batch with Mountpoint for Amazon S3.
- goofys. 2025. goofys README.
- s3fs-fuse. 2026. s3fs-fuse README.

12장. SageMaker로 배우는 오믹스 머신러닝

학습 목표

- SageMaker의 기본 역할을 설명할 수 있다.
- 오믹스 데이터 분석에서 SageMaker를 언제 고려할 수 있는지 이해할 수 있다.
- 노트북, 학습 작업, 모델 배포를 서로 구분할 수 있다.
- 전통적인 생물정보학 파이프라인과 머신러닝 워크플로우가 어떻게 연결되는지 설명할 수 있다.

핵심 질문

- SageMaker는 단순한 Jupyter 서버와 무엇이 다른가
- 유전체와 전사체 데이터 분석에서 어떤 유형의 모델 작업에 적합한가
- 전통적인 생물정보학 파이프라인과 머신러닝 워크플로우는 어떻게 연결되는가
- Notebook, Training Job, Endpoint를 각각 언제 써야 하는가

SageMaker를 오믹스 분석에서 어디에 놓아야 하는가

오믹스 분석에서 SageMaker를 이해하는 가장 좋은 방법은 그것을 “생물정보학 파이프라인을 대체하는 도구”로 보지 않는 것이다. 정렬, 변이 호출, 발현 정량, QC처럼 원시 데이터를 분석 가능한 형태로 바꾸는 과정은 여전히 Nextflow, WDL, CWL, AWS Batch, HealthOmics 같은 워크플로 계층이 더 자연스럽다. 반면 SageMaker는 이렇게 정리된 표, feature matrix, embedding, label, 이미지 타일, sequence representation 위에 탐색, 모델 학습, 추론, 실험 관리, 배포를 얹는 계층에 더 가깝다. 즉 SageMaker는 생물정보학의 앞단을 대체하기보다, 그 위에 머신러닝과 상호작용형 분석을 올리는 역할을 맡는다.

서비스 이름의 변화도 여기서 짚고 넘어갈 필요가 있다. AWS 공식 문서에 따르면 2024년 12월 3일 기존 `Amazon SageMaker`는 `Amazon SageMaker AI`로 이름이 바뀌었고, 같은 날 “next generation of Amazon SageMaker”가 데이터, 분석, AI를 아우르는 상위 플랫폼으로 발표되었다. 따라서 오늘 문서를 읽을 때 `SageMaker`라는 이름은 두 층의 의미를 가진다. 하나는 모델을 만들고 학습하고 배포하는 `SageMaker AI`이고, 다른 하나는 Unified Studio, Lakehouse, Governance 등을 포함한 더 큰 통합 플랫폼이다. 학생이 이 날짜와 구조를 알고 있으면 문서 이름이 섞여 보여도 훨씬 덜 혼란스럽다 (AWS 2024a; AWS 2026a).

오믹스 관점에서 보면 이 구조는 오히려 자연스럽다. 데이터 저장과 대규모 워크플로 실행은 S3, HealthOmics, Batch, Athena 같은 계층에 맡기고, 그 결과를 가지고 분류, 회귀, 군집화, 차원 축소, 임베딩 생성, foundation model 학습과 추론을 수행하는 계층으로 SageMaker를 두면 된다. 따라서 이 장에서 SageMaker는 "Notebook 도구"로 좁게 보아서는 안 된다. 노트북은 진입점일 뿐이고, 실제 강점은 데이터와 계산과 모델 운영을 하나의 관리형 흐름으로 묶어 준다는 데 있다.

Notebook, Studio, Unified Studio, 학습 작업은 어떻게 다른가

초보자가 가장 많이 헷갈리는 부분은 SageMaker의 화면과 실행 단위를 한 덩어리로 보는 것이다. 하지만 실제로는 역할이 분리되어 있다. 노트북은 탐색과 실험을 위한 대화형 환경이고, 학습 작업(training job)은 코드와 데이터를 받아 독립적으로 실행되는 관리형 배치 학습이며, 실시간 endpoint는 학습이 끝난 모델을 서비스형 추론으로 노출하는 계층이다. 따라서 "SageMaker를 쓴다"는 말은 단순히 JupyterLab을 켠다는 뜻이 아니라, 필요에 따라 다른 실행 모드를 선택한다는 뜻이다.

여기에 최근의 UI 변화가 겹친다. AWS 문서에 따르면 2023년 11월 30일 이전의 SageMaker Studio 경험은 `SageMaker Studio Classic`으로 이름이 바뀌었고, 기존 workload 유지용으로만 남아 있으며 신규 온보딩은 중단되었다. 이후의 `SageMaker Studio`는 최신 ML workflow용 웹 경험이고, 2024년 12월 3일 preview로 공개된 `SageMaker Unified Studio`는 데이터, SQL, AI, ML 도구를 한 프로젝트 안에서 함께 다루는 상위 통합 환경이다. 즉 오늘날 학생이 새로 배우는 기준점은 `Studio Classic`이 아니라 `Studio`와 `Unified Studio`라고 이해하는 편이 맞다 (AWS 2026b; AWS 2026c; AWS 2026d).

Table 1은 오믹스 분석에 필요한 주요 실행 모드를 정리한 것이다. 중요한 것은 어떤 화면이 더 최신인가보다, 어떤 작업이 대화형 탐색에 맞고 어떤 작업이 배치 실행에 맞는가를 구분하는 일이다.

Table 1. SageMaker의 주요 실행 모드

구성 요소	핵심 역할	오믹스 예시
Notebook 또는 JupyterLab	데이터 탐색, 시각화, 가벼운 실험	PCA, UMAP, QC plot, feature engineering
SageMaker Studio	최신 ML 중심 개발 환경	모델 실험, 코드 작성, job 관리
SageMaker Unified Studio	데이터와 AI를 함께 다루는 통합 프로젝트 공간	Athena 결과 탐색, 데이터 자산 공유, ML 작업 연결
Training Job	독립적이고 재현 가능한 관리형 학습	genotype matrix 분류기 학습, 딥러닝 모델 pretraining
Processing Job 또는 스크립트 실행	전처리와 feature preparation	VCF 정리, cohort matrix 변환, 임베딩 계산
Real-time Endpoint	실시간 추론 서비스	sequence embedding API, variant effect scoring API
Batch Transform 또는 오프라인 추론	대량 예측 배치 실행	샘플 집합 전체에 대한 risk score 계산

Amazon Genomics CLI와 Nextflow 결과를 SageMaker로 연결하기

오믹스 머신러닝의 첫 번째 실제 패턴은 "무거운 생물정보학 계산은 워크플로 엔진이 처리하고, 결과 해석과 모델링은 SageMaker가 맡는다"는 구조다. AWS HPC 블로그의 2022년 예제는 이를 매우 교육적으로 보여 준다. 이 글에서 저자들은 Amazon Genomics CLI(AGC)가 AWS Batch 기반 실행 환경을 자동 구성해 `fetchngs`로 SRA 데이터를 FASTQ로 바꾸고, 이어서 Sarek 파이프라인으로 variant calling을 수행한 다음, 최종 VCF 결과를 SageMaker notebook으로 넘겨 상호작용형 분석과 시각화를 진행한다. 이 흐름은 학생에게 매우 중요한 메시지를 준다. 머신러닝은 생물정보학 파이프라인 이후에 얹히는 "3차 분석 계층"이라는 것이다 (Nocaj et al. 2022).

이 패턴의 장점은 두 가지다. 첫째, 파이프라인 단계와 모델링 단계를 분리함으로써 각 단계에 가장 적합한 실행 환경을 선택할 수 있다. 정렬과 변이 호출은 대개 workflow engine, container, batch queue가 더 잘 맞는다. 반면 결과를 정리해 분류 모델을 실험하고 시각화하는 일은 노트북 환경이 더 자연스럽다. 둘째, 입력과 출력을 S3에 두면 두 계층이 느슨하게 연결된다. 즉 파이프라인이 끝난 뒤 생성된 VCF, TSV, parquet, summary table을 SageMaker가 그대로 읽어 이어서 탐색하고 학습할 수 있다.

학생이 이 구조를 이해하면 SageMaker를 불필요하게 과장하지 않게 된다. SageMaker는 BWA나 GATK를 대신 돌려 주는 서비스가 아니다. 하지만 BWA와 GATK가 만든 결과를 feature table로 정리하고, 유전자별 패턴을 시각화하고, 예측 모델이나 군집 모델을 훈련하는 데에는 매우 적합하다. 좋은 오믹스 시스템은 이 두 세계를 경쟁 관계로 놓지 않고, `workflow for transformation`과 `SageMaker for learning and interpretation`으로 분업시킨다.

1000 Genomes 예제로 보는 tertiary analysis

이 장의 두 번째 핵심 패턴은 `HealthOmics + Athena + SageMaker` 조합이다. AWS Industries 블로그는 2025년 1월 29일, 1000 Genomes 데이터를 사용해 population variation을 분석하는 흐름을 제시했다. 이 예제의 교육적 가치는 PCA 자체보다, 대규모 변이 데이터를 어떻게 tertiary analysis용 구조로 연결하는지 보여 준다는 데 있다. 글의 흐름은 대략 이렇다. 원본 변이 자산을 AWS에서 관리 가능한 저장 계층으로 가져오고, 필요한 cohort를 Athena로 질의해 좁힌 다음, SageMaker Studio에서 PCA와 시각화를 수행한다 (Tzouvanas 2025).

이 예제가 중요한 이유는 대규모 유전체 분석에서 머신러닝 전처리가 곧 데이터 질의 문제라는 사실을 잘 보여 주기 때문이다. PCA를 하려면 결국 샘플 x 변이 행렬 또는 요약 feature matrix가 필요하다. 그런데 그 행렬을 만들기 전 단계에서 이미 cohort filtering, annotation selection, population label 정리, missingness 처리 같은 일이 발생한다. 즉 오믹스 머신러닝은 "바로 모델부터 돌리는 일"이 아니라, 질의 가능한 저장 계층과 상호작용형 분석 계층을 연결하는 일에 훨씬 가깝다.

Table 2는 이 패턴을 학습용으로 요약한 것이다. 이 구조는 1000 Genomes에만 해당하지 않는다. 암 코호트, rare disease cohort, single-cell feature matrix, proteomics abundance table에도 거의 그대로 확장된다.

Table 2. 오믹스 ML에서 자주 쓰는 연결 패턴

단계	주된 서비스	의미
원시 또는 준정리 데이터 저장	S3, HealthOmics	재사용 가능한 데이터 자산 보관
메타데이터 필터링과 cohort selection	Athena 또는 SQL 계층	필요한 샘플과 feature 범위 좁히기
feature matrix 생성	Processing job, notebook, workflow 후처리	모델 입력 형태로 변환
탐색과 시각화	Notebook, Studio, Unified Studio	PCA, clustering, QC, 가설 형성
모델 학습	SageMaker Training Job	재현 가능한 학습과 대규모 실험
배포 또는 배치 추론	Endpoint, Batch Transform	실제 사용 또는 대규모 scoring

실용 모델에서 genomic foundation model까지

오믹스 머신러닝이라고 하면 초보자는 곧바로 거대한 딥러닝 모델부터 떠올리기 쉽다. 하지만 실제 연구 현장에서는 분류, 회귀, 차원 축소, 임베딩 생성, 이상치 탐지 같은 비교적 실용적인 작업이 여전히 매우 중요하다. 예를 들어 환자 샘플의 subtype 분류, 발현 signature 기반 score 예측, variant burden을 이용한 population clustering, single-cell embedding 계산은 모두 SageMaker가 잘 맞는 작업이다. 특히 입력이 이미 S3에 있고, 전처리 결과를 notebook으로 확인한 뒤 학습 작업으로 넘기고 싶을 때 SageMaker의 관리형 실행 모델이 강점을 보인다.

그 위에 더 큰 확장으로 foundation model 계열이 있다. AWS Machine Learning 블로그의 2024년 글은 AWS HealthOmics sequence store를 오믹스 데이터 저장 계층으로 사용하고, SageMaker Training을 이용해 HyenaDNA 계열 genomic language model을 pre-train하는 예제를 제시한다. 여기서 중요한 메시지는 "오믹스 ML의 최신 확장도 결국 S3 또는 HealthOmics에 정리된 데이터 위에서 관리형 학습 작업을 돌리는 구조"라는 점이다. 즉 foundation model이 등장해도 데이터 계층과 학습 계층의 분리는 여전히 유지된다 (Ariyawansa and Handley 2024).

다만 모든 팀이 foundation model stack을 직접 구축할 필요는 없다. 대부분의 연구실과 바이오텍 팀은 먼저 작은 supervised task, embedding 생성, notebook 기반 탐색에서 더 큰 가치를 얻는다. 학생에게는 이 순서를 분명히 가르치는 편이 중요하다. 먼저 `SageMaker` 을 익히고, 그다음에야 분산 학습과 대형 모델로 확장하는 것이 바람직하다. 특히 single-cell과 spatial omics의 데이터 접근 계층은 따로 복잡하므로, 그 부분은 13장에서 다루는 chunked storage와 remote access 개념과 함께 보는 편이 정확하다.

비용과 운영 원칙을 어떻게 잡아야 하는가

SageMaker의 가장 흔한 오해는 "관리형 서비스니까 알아서 싸고 알아서 효율적일 것"이라는 기대다. 실제로는 Notebook, Training Job, Endpoint의 비용 구조가 다르므로 작업 종류에 맞춰 선택해야 한다. 잠깐 탐색하고 그림을 그릴 일이라면 대화형 노트북이 적합하지만, 수 시간 또는 수일이 걸리는 학습은 Training Job으로 분리하는 편이 좋다. 모델을 한 번만 대량 추론할 것이라면 실시간 endpoint보다 batch 방식이 낫다. 결국 비용 최적화의 핵심은 같은 도구를 오래 켜 두는 것이 아니라, 실행 모드를 목적에 맞게 분리하는 데 있다.

오믹스 분석에서는 데이터 위치도 중요하다. Training Job과 입력 S3 버킷, HealthOmics sequence store, Athena query result가 같은 리전 안에 있어야 운영이 단순하고 전송 비용 위험도 줄어든다. 또한 notebook에서 우연히 잘 돌아간 코드를 곧바로 논문용 생산 파이프라인으로 착각해서는 안 된다. 재현성 있는 학습을 원하면 코드 버전, 데이터 snapshot, 파라미터, 컨테이너 이미지를 분리해 관리형 job으로 승격시켜야 한다. 이 원칙은 작은 random forest에서 대형 genomic language model에 이르기까지 모두 같다.

SageMaker는 오믹스 분석의 전 과정을 혼자 담당하는 서비스가 아니다. 대신 S3와 HealthOmics에 정리된 데이터, Batch와 Nextflow가 만든 결과, Athena가 좁혀 준 cohort를 바탕으로, 탐색에서 학습과 배포까지 이어지는 관리형 ML 계층을 제공한다. 따라서 오믹스에서 SageMaker를 잘 쓴다는 것은 "클라우드에 Jupyter를 띄운다"가 아니라, 데이터 파이프라인과 모델 파이프라인의 경계를 명확히 세우고 그 사이를 안정적으로 연결하는 일에 가깝다.

핵심 개념 정리

- SageMaker는 생물정보학 파이프라인을 대체하기보다, 그 결과 위에 머신러닝과 상호작용형 분석을 얹는 계층이다.
- 2024년 12월 3일 이후 `SageMaker AI`는 기존 ML 서비스 이름이고, `next generation of SageMaker`는 더 큰 통합 플랫폼 이름이다.
- Notebook, Training Job, Endpoint는 각각 탐색, 재현 가능한 학습, 서비스형 추론이라는 서로 다른 역할을 가진다.
- AGC, Nextflow, HealthOmics, Athena와 연결될 때 SageMaker는 오믹스 tertiary analysis의 핵심 계층이 된다.
- foundation model도 중요하지만, 대부분의 실전 가치는 작은 supervised task와 feature engineering, 시각화, 배치 학습에서 먼저 나온다.

복습 질문

1. SageMaker를 단순한 Jupyter 서버로 이해하면 놓치게 되는 핵심 기능은 무엇인가?
2. 오믹스 분석에서 workflow engine과 SageMaker가 각각 맡기 좋은 역할은 무엇인가?
3. `HealthOmics + Athena + SageMaker` 패턴이 tertiary analysis에 적합한 이유는 무엇인가?
4. Notebook, Training Job, Endpoint는 어떤 기준으로 선택해야 하는가?

Further Reading

- AWS. What is Amazon SageMaker AI?.
- AWS. What is Amazon SageMaker Unified Studio?.
- AWS HPC Blog. Analyzing Genomic Data using Amazon Genomics CLI and Amazon SageMaker.

References

- Ariyawansa S, Handley S. 2024. Pre-training genomic language models using AWS HealthOmics and Amazon SageMaker.
- AWS. 2024a. Introducing the next generation of Amazon SageMaker.
- AWS. 2026a. What is Amazon SageMaker AI?.
- AWS. 2026b. Amazon SageMaker Studio.
- AWS. 2026c. Amazon SageMaker Studio Classic.
- AWS. 2026d. What is Amazon SageMaker Unified Studio?.
- Nocaj A, Guruacharya A, Choudhury O, Pang L. 2022. Analyzing Genomic Data using Amazon Genomics CLI and Amazon SageMaker.

- Tzouvanas K. 2025. Examine genomic variation across populations with AWS.

13장. 클라우드 네이티브 single-cell과 spatial omics - Zarr, AnnData, OME-Zarr, SOMA

학습 목표

- 대규모 single-cell과 spatial omics에서 왜 방식이 점점 비효율적인지 설명할 수 있다.
- `AnnData`, `h5ad`, `Zarr`, `OME-Zarr`, `TileDB-SOMA`가 각각 어느 추상화 계층에 있는지 구분할 수 있다.
- scRNA-seq과 spatial transcriptomics가 왜 서로를 대체하기보다 보완하는지 설명할 수 있다.
- AWS에서 S3, 분산 계산, 웹 시각화를 이용해 클라우드 네이티브 오믹스 분석을 설계하는 기본 사고방식을 이해할 수 있다.

핵심 질문

- 왜 큰 `h5ad` 파일 하나를 열어 두는 방식만으로는 최신 single-cell과 spatial omics를 다루기 어려운가
- 왜 single-cell에서는 정렬보다 annotation과 benchmarking이 더 큰 병목이 되기 시작했는가
- Zarr는 단순한 파일 포맷이 아니라 어떤 접근 패턴의 변화를 뜻하는가
- OME-Zarr와 TileDB-SOMA는 Zarr와 어떤 관계에 있으며, 언제 서로 다른 선택지가 되는가

이 장의 구성

이 장은 내용이 빽빽하므로 네 부분으로 나누어 읽는 것이 좋다. 각 부분은 독립적인 질문에 답한다.

- **Part A. 왜 파일 중심 사고가 무너졌는가** (13.1 ~ 13.2): 대규모 single-cell이 인프라 문제가 된 배경.
- **Part B. 저장 모델의 전환** (13.3 ~ 13.4): Zarr의 chunk 기반 접근과 AnnData/Zarr 조합.
- **Part C. 공간·이미징으로의 확장** (13.5 ~ 13.6): OME-Zarr, SpatialData, single-cell + spatial 통합.
- **Part D. 플랫폼과 운영** (13.7 ~ 13.9): TileDB-SOMA, AWS 실전 배치, 한계와 주의점.

개념이 처음이라면 Part A → B 순서로 두 부분만 먼저 읽고 13.9 요약으로 넘어가도 좋다. Part C, D는 필요할 때 다시 돌아와 참조하면 된다.

Part A. 왜 파일 중심 사고가 무너졌는가

왜 `download -> open` 패턴이 무너지기 시작했는가

대규모 생물학 데이터 분석은 오랫동안 는 전제를 바탕으로 발전해 왔다. 이 방식은 작은 bulk RNA-seq 데이터나 제한된 수의 샘플을 다룰 때는 충분히 실용적이었다. 그러나 single-cell, spatial transcriptomics, 다중 모달 오믹스, 고해상도 현미경 이미지가 합쳐지기 시작하면서 병목은 계산보다 데이터 접근 방식에서 먼저 드러났다. 이제 문제는 파일을 어디에 저장하느냐가 아니라, 필요한 세포 집단과 유전자 집합, 공간 영역, 이미지 해상도만 골라 빠르게 읽고 바로 계산할 수 있느냐로 바뀌었다. 즉 분석의 중심 단위가 에서 으로 이동하기 시작한 것이다.

이 변화는 단순한 저장 형식의 차이로 설명하기 어렵다. 예전에는 데이터를 한 번 로컬에 받으면 그다음부터는 디스크 속도와 메모리만 신경 쓰면 되었다. 반면 객체 스토리지 시대에는 네트워크를 건너 어떤 조각을 몇 번 읽는지, 작은 객체가 얼마나 많은지, 메타데이터를 어떻게 관리하는지가 성능과 비용을 동시에 결정한다. 따라서 오늘날의 omics 교육에서는 파일 포맷을 저장 형식으로만 가르쳐서는 부족하다. 어떤 접근 패턴이 어떤 저장 구조와 맞는지를 함께 가르쳐야 한다. 이 장은 바로 그 전환을 single-cell과 spatial omics라는 가장 극적인 사례를 통해 설명한다.

single-cell은 왜 인프라 문제가 되었는가

single-cell RNA-seq이 기존 유전체 분석과 다른 이유는 데이터 규모가 샘플 수보다 세포 수에 의해 결정된다는 점이다. 하나의 실험에서 수만 개에서 수백만 개의 세포가 측정되고, 데이터는 본질적으로 거대한 `cells x genes` 희소 행렬이 된다. 이 구조에서는 단순 정렬과 카운팅만이 문제가 아니다. batch integration, embedding, cell-type annotation,

방법론 benchmarking 같은 후반부 단계가 점점 더 큰 계산 병목이 된다. 즉 single-cell 시대의 인프라 문제는
보다 에 더 가깝다.

AWS의 Open Problems 사례는 이 점을 잘 보여 준다. 로컬에서 개발된 Docker container와 Nextflow workflow는 실제 benchmarking 단계에서 `ECR -> Nextflow -> AWS Batch -> EC2 Spot -> S3` 구조 위로 확장된다. 이 플랫폼의 목적은 단일 분석을 한 번 빨리 끝내는 것이 아니라, 같은 데이터셋에 여러 알고리즘을 반복 적용하고 성능을 비교하며, spatial deconvolution 같은 과제까지 동일한 실행 계층에서 평가하는 것이다. 즉 single-cell 클라우드는 "더 센 서버"보다 "재현 가능한 대규모 비교 실험실"에 더 가깝다. 이 점을 이해해야 single-cell 인프라가 왜 파일 관리 문제가 아니라 플랫폼 설계 문제가 되었는지 자연스럽게 보이기 시작한다.

Figure 13-1은 기존의 파일 중심 접근과 chunk 기반 원격 접근의 차이를 개념적으로 보여 준다. 같은 데이터를 다룬다고 하더라도, 어떤 단위로 접근하느냐에 따라 네트워크 왕복, 메모리 사용, 비용 구조가 모두 달라진다. 이 차이가 곧 클라우드 네이티브 오믹스의 출발점이다.

Figure 13-1. "download → open" vs "read chunks in place"

Part B. 저장 모델의 전환

Zarr와 chunk 기반 접근 – 객체 스토리지 시대의 배열 저장

이 전환을 이해하는 데 Zarr는 매우 좋은 교육 도구다. Zarr는 N차원 배열을 작은 chunk 단위로 나누어 저장하고, 각 chunk를 독립적으로 접근 가능한 객체로 다룬다. AWS Public Sector의 Zarr on S3 글은 바로 이 점 때문에 객체 스토리지 위에서 필요한 조각을 빠르게 읽고 병렬 처리할 수 있다고 설명한다. 따라서 Zarr의 핵심은 "확장자 하나가 더 생겼다"가 아니라, 대신 는 사고방식을 가능하게 해 준다는 데 있다. 클라우드 네이티브 분석이라는 말은 대개 이 chunk 기반 접근을 전제로 한다.

하지만 여기서 바로 중요한 caveat가 나온다. Zarr는 만능이 아니고, S3 고 가르치면 오히려 부정확해진다. 실제 성능과 비용은 chunk 크기, sharding, metadata consolidation, object 수, request pattern에 크게 좌우된다. chunk를 너무 잘게 쪼개면 small object가 폭발하고, request cost와 listing overhead가 커질 수 있다. 따라서 Zarr를 단순 포맷으로 소개하기보다, 로 소개하는 편이 교육적으로 훨씬 낫다.

AnnData + Zarr + Vitessce – single-cell 데이터 모델의 클라우드 확장

single-cell 분야에서는 이 흐름을 `AnnData + Zarr`라는 조합으로 가장 쉽게 설명할 수 있다. `AnnData`는 세포 x 유전자 행렬과 `obs`, `var`, `obsm`, `layers`를 함께 다루는 데이터 모델이고, `h5ad`와 `Zarr`는 이를 저장하는 서로 다른 backend라고 보는 편이 가장 자연스럽다. 작은 분석이나 로컬 notebook 실습에서는 `h5ad`가 여전히 편리하다. 그러나 반복 공유, 웹 시각화, remote access, larger-than-memory 분석이 중요해질수록 Zarr backend의 장점이 커진다. AnnData 문서도 `write_zarr()`를 공식 API로 제공하고, zarr v3와 remote access를 중요한 발전 방향으로 다루고 있다.

Vitessce 예제는 이 변화를 학생들에게 직관적으로 보여 준다. 흐름은 `h5ad`를 읽고, 시각화에 필요한 embedding과 feature matrix를 정리한 뒤, `write_zarr()`로 Zarr store를 만들고, 이를 S3에 올려 웹 기반 인터랙티브 뷰어에서 공유하는 방식이다. 즉 single-cell 데이터는 더 이상 개인용 분석 파일에만 머무르지 않고, 객체 스토리지 위에 놓인 배포형 자산이 된다. 이때 Zarr는 단순한 분석 포맷이 아니라 협업과 공유를 위한 배치 전략이 된다. 학생은 여기서 `AnnData =` , `h5ad/Zarr =` `backend`, `Vitessce =` `/` 이라는 추상화 수준을 함께 배워야 혼란이 줄어든다.

Part C. 공간·이미징으로의 확장

OME-Zarr와 SpatialData – 공간오믹스와 이미징 데이터의 만남

공간오믹스(spatial omics)와 이미징으로 가면 이 논리는 더 강해진다. OME-Zarr는 생물학 이미지와 다중 해상도 피라미드, segmentation, annotation을 클라우드 친화적으로 다루기 위한 포맷이다. spatial omics에서는 세포 x 유전자 행렬만으로는 충분하지 않다. 위치 좌표, 세포 경계, 조직 이미지, 여러 해상도의 시각화 정보를 함께 다뤄야 한다. 이때 는 발상은 단순한 편의가 아니라 필수적인 운영 원리가 된다. OME-Zarr가 중요한 이유는 바로 이 멀티스케일 접근을 표준화하려 하기 때문이다.

SpatialData는 이 생태계를 더 상위 수준에서 정리하는 프레임워크다. 공식 문서는 이를 `Images`, `Labels`, `Points`, `Shapes`, `Tables`를 한 좌표계 안에서 함께 다루는 공통 데이터 모델로 설명한다. 교육적으로는 이 구분 자체가 매우

유용하다. 학생은 공간전사체를 단순히 `spot x gene matrix`로만 보는 대신, 여기에 이미지와 세포 경계, 위치 정보, 파생 표가 함께 묶인다는 점을 이해하게 된다. 즉 OME-Zarr는 공간오믹스의 저장과 전달 계층을, SpatialData는 그 위의 통합 표현 계층을 보여 주는 셈이다.

single-cell + spatial integration – matrix에서 matrix + coordinates로

single-cell만으로는 충분하지 않다는 사실도 함께 배워야 한다. scRNA-seq은 세포 정체성과 이질성을 세포 단위로 보여 주지만, 조직 안에서 그 세포가 어디 있었는지는 잃어버린다. spatial transcriptomics는 바로 이 빈틈을 메우지만, 해상도와 유전자 수, 시야 범위 사이의 trade-off를 가진다. 그래서 최신 계산의 핵심은 둘 중 하나를 고르는 것이 아니라 `single-cell + spatial integration`이다. 공간전사체 데이터는 단순한 행렬이 아니라 `matrix + coordinates (+ image)`이며, 실제로는 neighborhood, cell-cell communication, spatial domain, deconvolution까지 함께 해석해야 의미가 커진다.

`gimVI`는 이런 통합 계산의 대표적 예시다. 이 모델은 짝지어지지 않은 scRNA-seq와 spatial 데이터를 함께 사용해 spatial에서 직접 측정되지 않은 유전자 정보를 추정하려는 접근을 보여 준다. 다만 이것을 유일한 표준처럼 가르칠 필요는 없다. 더 중요한 것은 왜 통합 계산이 필요해졌는지를 이해하는 것이다. single-cell은 세포 정체성을 세밀하게 제공하고, spatial은 조직 문맥을 제공하므로, 둘을 함께 봐야 실제 생물학적 해석이 훨씬 풍부해진다.

Part D. 플랫폼과 운영

CELLxGENE Census와 TileDB-SOMA – 왜 더 큰 플랫폼은 다른 길을 택했는가

여기서 Zarr만이 유일한 길이라고 가르치면 오히려 부정확해진다. CZ CELLxGENE Discover와 Census는 대규모 single-cell 데이터 접근을 위해 TileDB-SOMA를 사용한다. 이 선택은 Zarr가 틀렸다는 뜻이 아니라, tens of millions of cells 규모의 larger-than-memory 질의, cross-language API, 버전된 데이터 스냅샷, 더 엄격한 스키마 관리가 필요한 플랫폼에서는 상위 계층이 달라질 수 있음을 보여 준다. 교육적으로는 `AnnData + Zarr`를 연구자 친화적이고 scverse 중심의 경로로, `TileDB-SOMA`를 더 큰 규모의 플랫폼형 질의 계층으로 설명하는 편이 좋다. 즉 둘은 경쟁 관계라기보다 서로 다른 문제 규모에 대응하는 선택지에 가깝다.

또한 CELLxGENE 플랫폼과 Census object를 구분하는 것도 중요하다. 개별 source dataset은 여전히 `h5ad`로 유통될 수 있지만, corpus-wide query 계층은 `TileDB-SOMA`로 제공된다. 학생이 이 점을 이해하면 "클라우드 네이티브 single-cell"이 반드시 하나의 형식만을 뜻하지 않는다는 사실을 자연스럽게 받아들일 수 있다. 결국 핵심은 특정 확장자에 대한 충성도가 아니라, 어떤 규모와 어떤 질의 패턴에 어떤 데이터 모델이 맞는지를 판단하는 능력이다.

AWS에서의 실전 운영 – S3, Batch, ECS/Fargate, SageMaker, Bedrock

AWS 관점에서 핵심은 결국 같다. 데이터를 S3에 두고, 필요한 부분만 읽고, 계산은 데이터 가까이에서 병렬로 수행하고, 결과는 notebook이나 웹 인터페이스로 다시 연결하는 것이다. AWS의 Zarr 사례는 geospatial 데이터를 예로 들지만, 원리는 single-cell과 spatial omics에도 거의 그대로 적용된다. Dask, ECS/Fargate, SageMaker notebook, S3 VPC endpoint 같은 조합은 대규모 배열 데이터를 클라우드에서 직접 처리하는 전형적인 패턴을 보여 준다. 중요한 것은 이 아니라, 파일이 아니라 chunk를 계산 단위로 바꾸는 것이다.

실제 산업 사례도 비슷한 방향을 보여 준다. Sonrai는 scRNA-seq cluster annotation을 `HealthOmics + S3 + SageMaker + Bedrock` 구조로 자동화했고, One BioSciences는 tens of thousands of RNA profiles를 다루기 위해 `EC2 + S3 + MWAA + Athena + Lambda`와 GPU 자원을 조합했다. 여기서 중요한 점은 AI가 raw omics 전체를 대체한다는 환상이 아니라, annotation과 interpretation 같은 후반부 병목을 줄이는 해석 계층으로 작동한다는 사실이다. 이 장에서 Bedrock은 "모든 분석을 해 주는 엔진"이 아니라, 잘 구조화된 데이터와 워크플로 위에서 마지막 해석 단계를 보조하는 계층으로 이해하는 편이 정확하다. 결국 클라우드 네이티브 omics는 저장, 계산, 시각화, 해석이 느슨하게 연결된 하나의 플랫폼 구조로 보아야 한다.

한계와 주의점 – small objects, sharding, evolving APIs

물론 Zarr와 클라우드 네이티브 접근에는 한계도 분명하다. chunk를 너무 잘게 쪼개면 small object가 급격히 늘고, request cost와 listing 비용이 커질 수 있다. zarr v3의 sharding은 이런 문제를 줄이기 위한 방향이지만, 실제 구현 세부는 계속 진화 중이다. AnnData 문서 기준 일부 기능은 아직 실험적 성격이 있으므로, 교과서에서는 세부 API보다

원리와 설계 감각을 남기는 편이 더 오래간다. 또한 같은 S3에 데이터를 올렸다고 해서 자동으로 빠르고 경제적인 구조가 되는 것은 아니다.

운영 감각도 함께 가르쳐야 한다. compute와 S3를 같은 region에 두는지, private VPC에서 S3 endpoint를 사용하는지, 고동시성 접근에서 `503 Slow Down` 같은 신호를 어떻게 모니터링하는지 같은 문제는 클라우드에서도 여전히 중요하다. Sonrai나 One BioSciences 같은 고객 사례의 시간 단축 수치는 학술 벤치마크가 아니라 사례 보고라는 점도 분명히 적어야 한다. `h5ad`를 버리자는 것이 아니라, 클라우드 시대의 single-cell과 spatial omics에서는 에서 , 나아가 에서 으로 넘어가야 한다는 점이다.

핵심 개념 정리

- single-cell과 spatial omics의 핵심 병목은 큰 파일 자체보다 희소 행렬, 좌표, 이미지, 반복 질의와 통합 계산에 있다.
- `AnnData` = , `h5ad/Zarr` = backend, `OME-Zarr` = / , `TileDB-SOMA` = 으로 구분하면 이해가 쉬워진다.
- Zarr의 핵심은 파일 확장자가 아니라 chunk 기반 원격 접근과 병렬 계산이라는 접근 패턴이다.
- 클라우드 네이티브 분석은 더 큰 서버를 사는 일이 아니라, 필요한 부분만 읽고 데이터 가까이에서 계산하는 구조를 만드는 일이다.

복습 질문

1. 왜 최신 single-cell 분석에서는 정렬보다 annotation과 benchmarking이 더 큰 병목이 되기 시작했는가?
2. `h5ad`와 `Zarr`의 차이를 저장 형식이 아니라 접근 패턴의 차이로 설명해 보라.
3. OME-Zarr와 TileDB-SOMA는 각각 어떤 문제 규모와 어떤 데이터 계층에 더 잘 맞는가?

Further Reading

- AWS Public Sector Blog. Decrease geospatial query latency from minutes to seconds using Zarr on Amazon S3.
- Vitessce. Export data to AWS S3.
- SpatialData. Documentation.

References

- AnnData. 2026. AnnData.write_zarr.
- AnnData. 2026. zarr-v3 Guide/Roadmap.
- AWS Public Sector Blog. 2024. Decrease geospatial query latency from minutes to seconds using Zarr on Amazon S3.
- AWS. 2024. Driving innovation in single-cell analysis on AWS.
- AWS. 2025. One BioSciences case study.
- AWS. 2025. Sonrai case study.
- Czerny et al. 2025. CZ CELLxGENE Discover: a single-cell data platform for scalable exploration, analysis, and modeling of aggregated data.
- Lopez et al. 2019. A joint model of unpaired data from scRNA-seq and spatial transcriptomics for imputing missing gene expression measurements.
- Moore et al. 2023. OME-Zarr: a cloud-optimized bioimaging file format with international community support.
- SpatialData. 2026. Documentation.
- Vitessce. 2026. Export data to AWS S3.

14장. AWS에서 Claude와 Kiro로 하는 바이브 코딩

학습 목표

- Kiro와 Claude on Bedrock이 각각 어떤 역할을 하는지 설명할 수 있다.
- 유전체 분석 코드 작성에서 agentic coding 도구가 어디까지 유용한지 구분할 수 있다.
- 바이브 코딩을 실험적 생산성 도구로 쓰되, 과학적 검증은 왜 별도로 필요하게 되는지 이해할 수 있다.

핵심 질문

- Kiro는 일반 IDE나 단순 코드 자동완성과 무엇이 다른가
- Claude를 AWS Bedrock 위에서 쓰는 것은 어떤 운영상 의미를 가지는가
- AI가 파이프라인 초안을 잘 만든다고 해서 왜 결과 검증까지 맡길 수는 없는가

바이브 코딩을 어떻게 가르칠 것인가

이라는 표현은 편리하지만, 교과서에서는 조금 더 조심해서 써야 한다. 학생에게 중요한 것은 AI가 "대충 감으로 코드를 써 준다"는 인상이 아니라, 요구사항을 더 빨리 문서화하고 반복 작업을 자동화하고 디버깅을 보조하는 도구로 쓰인다는 사실이다. omics 분석에서는 작은 스크립트 하나가 곧바로 논문 결과나 임상 해석으로 이어질 수 있기 때문에, 코드 생성 속도와 결과의 신뢰도를 반드시 분리해서 생각해야 한다. 따라서 이 장의 목표는 AI 도구를 찬양하거나 두려워하게 만드는 것이 아니라, 어디까지 자동화하고 어디서부터 사람이 다시 검토해야 하는지 경계를 그리게 하는 것이다. 그 경계가 분명할수록 AI 보조 도구는 연구 생산성을 높이는 실용 도구가 된다.

2026년 기준으로 AWS 문맥에서 이 문제를 설명하기에 좋은 조합은 Kiro와 Claude on Bedrock이다. Kiro는 agentic coding IDE로서 spec-driven workflow를 강조하고, Claude on Bedrock은 AWS 계정과 IAM, 리전, 모델 접근 정책 안에서 생성형 모델을 호출하게 해 준다. 즉 하나는 개발 환경에 가깝고, 다른 하나는 모델 접근 계층에 가깝다. 이 둘을 같은 것으로 가르치면 학생이 개념을 혼동하기 쉽다. Kiro는 코딩 작업을 조직하는 도구이고, Bedrock은 모델을 안전하게 호출하는 운영 계층이라는 구분이 먼저다.

Kiro – 감이 아니라 specification을 남기는 IDE

Kiro의 가장 큰 교육적 장점은 "생각나는 대로 코드를 쓴다"는 흐름보다, `requirements.md`, `design.md`, `tasks.md`처럼 중간 산물을 남기며 코딩하게 만든다는 점이다. 이 방식은 omics 연구와 특히 잘 맞는다. 유전체 분석 파이프라인은 입력 형식, 참조 데이터 버전, 출력 구조, 품질 기준이 조금만 어긋나도 결과 해석이 크게 달라질 수 있기 때문이다. 따라서 학생이 AI 도구를 쓸 때도 먼저 문제를 구조화하고, 입출력 조건과 예외를 적고, 작업 단계를 나누는 습관을 들이는 것이 중요하다. Kiro는 바로 이런 점에서 단순 자동완성 도구보다 더 교육적이다.

예를 들어 어떤 학생이 "VCF를 읽어 rare damaging variant를 필터링하고 샘플별 요약표를 만든다"는 작은 과제를 한다고 하자. 전통적인 바이브 코딩은 곧바로 코드를 생성하게 만들 수 있다. 반면 Kiro 스타일은 먼저 입력이 단일 VCF인지 cohort VCF인지, annotation이 이미 붙어 있는지, 출력이 TSV인지 Parquet인지, 결측값을 어떻게 처리할지, 재현 가능한 실행 방법은 무엇인지를 specification으로 적게 만든다. 이 과정은 느려 보이지만, 실제로는 나중에 생기는 대부분의 오류를 줄여 준다. omics 분석에서는 코드 작성 속도보다 이 더 중요할 때가 많다.

Claude on Bedrock – 모델을 AWS 운영 체계 안으로 가져오기

Claude를 AWS Bedrock 위에서 쓰는 이유는 모델 품질만이 아니다. Bedrock은 IAM 권한, 리전, 모델 접근 승인, 로깅과 같은 운영 요소를 AWS 계정 안으로 가져온다. 연구실 관점에서는 이것이 매우 중요하다. 같은 모델을 쓰더라도 누가 어떤 리전에서 어떤 권한으로 호출하는지, 어떤 프로젝트가 어떤 비용을 발생시키는지, 어떤 데이터와 함께 연결되는지를 더 명시적으로 관리할 수 있기 때문이다. Claude on Bedrock이나 Claude Code on Bedrock 문서는 이런 운영 준비가 model access, IAM permissions, region selection, model pinning 같은 문제와 연결된다고 설명한다.

교과서에서는 이를 "Claude를 어디서 쓰느냐"의 문제가 아니라, "생성형 AI를 연구 인프라 안에 어떻게 배치하느냐"의 문제로 설명하는 편이 좋다. 예를 들어 로컬 개발 환경에서는 Kiro 같은 IDE를 쓰고, 실제 모델 호출은 Bedrock을 통해 수행하게 하면, 코드 작성과 모델 운영의 경계를 더 분명히 나눌 수 있다. 이런 구조는 연구실이나 기관 단위 협업에서 특히

유용하다. 즉 Bedrock은 코딩 경험 그 자체보다, 코딩 도구가 조직의 보안과 비용 체계 안에서 움직이게 하는 계층으로 이해해야 한다.

유전체 분석에서 AI 코딩 도구가 실제로 잘하는 일

omics 분석에서 AI 코딩 도구가 가장 잘하는 일은 의외로 "새로운 알고리즘 발명"이 아니다. 오히려 파일 파서 초안 만들기, manifest 생성, CLI 스크립트 뼈대 작성, 문서화, 반복적 리팩터링, 작은 시각화 코드 작성, 테스트 케이스 제안, 에러 메시지 해석 같은 주변부 작업에서 강하다. 이 영역은 사람이 하자니 시간이 아깝고, 그렇다고 완전히 방치하면 코드 품질이 빨리 떨어지는 부분이다. AI 도구는 이런 작업을 빠르게 도와 주면서, 연구자가 더 중요한 분석 설계와 해석에 시간을 쓰게 만든다. 따라서 학생에게는 "AI가 어떤 코드를 가장 잘 보조하는가"를 구체적으로 가르치는 편이 훨씬 실용적이다.

실제 예시를 들면, AI는 Nextflow 입력 manifest를 샘플 시트에서 생성하는 작은 Python 스크립트, VCF annotation 결과를 cohort summary table로 바꾸는 SQL 초안, Hail QC 결과를 시각화하는 notebook 셀, README와 실행 예시를 자동으로 정리하는 작업에 매우 유용하다. 반면 통계 모델의 타당성 판단, rare variant 기준 정의, phenotype encoding, confounder 선택, clinical interpretation 같은 부분은 여전히 사람이 책임져야 한다. 즉 AI는 코딩의 많은 부분을 빠르게 하지만, 과학적 질문 자체를 대신 정하지는 못한다. 이 구분이 흐려지는 순간 바이브 코딩은 생산성 도구가 아니라 오류 증폭기가 될 수 있다.

바이브 코딩의 위험 – 환각, 잘못된 라이브러리, 검증 누락

AI 코딩 도구의 가장 큰 위험은 코드가 그럴듯해 보인다는 점이다. 특히 omics 분야는 포맷과 라이브러리의 차이가 복잡해서, 존재하지 않는 함수나 오래된 API, 잘못된 파일 형식 가정이 섞여 들어가기 쉽다. 더 큰 문제는 이런 오류가 종종 즉시 드러나지 않는다는 것이다. 파이프라인이 끝까지 돌아가더라도, 잘못된 phenotype merge나 변이 필터링 기준 때문에 결과가 조용히 왜곡될 수 있다. 따라서 이 장에서는 "AI가 쓴 코드는 우선 의심하고, 사람이 검증한 뒤에만 연구 코드로 승격한다"는 원칙을 명시해야 한다.

여기서 좋은 습관은 네 가지다. 첫째, AI가 만든 코드는 반드시 작은 샘플 데이터로 먼저 테스트한다. 둘째, 실행 결과뿐 아니라 중간 산물의 개수, 헤더, 샘플 수, 변이 수 같은 sanity check를 확인한다. 셋째, reference와 annotation 버전을 명시적으로 고정한다. 넷째, README와 test를 함께 생성하게 만들어 나중에 사람이 검토하기 쉽게 만든다. AI 코딩 도구의 진짜 가치는 코드 한 번에 있지 않고, 검증 가능한 초안을 빨리 만들게 해 주는 데 있다. 학생이 이 원칙을 익히면 바이브 코딩은 훨씬 안전한 생산성 도구가 된다.

작은 오믹스 프로젝트에서의 추천 워크플로

교육용으로는 너무 큰 프로젝트보다 작은 end-to-end 예제를 반복하는 편이 좋다. 예를 들어 `-> FASTQ QC -> -> README`나 `VCF -> filtering script -> variant summary -> plots` 같은 흐름은 AI 코딩 도구를 가르치기에 적합하다. 먼저 요구사항을 글로 적고, Kiro 같은 도구로 태스크를 쪼개고, Claude로 코드 초안을 만든 뒤, 작은 입력으로 실행하고, 결과를 수동 검토하고, 마지막에 test와 문서를 붙이는 순서를 반복하면 된다. 이 구조는 AI 도구를 쓰면서도 과학적 통제력을 잃지 않게 해 준다.

AI 코딩 도구는 omics 분석을 대신하는 과학자가 아니라, 초안 작성과 반복 작업을 보조하는 연구 보조원에 가깝다. Kiro는 생각을 구조화하게 해 주고, Claude on Bedrock은 모델 호출을 AWS 운영 체계 안에 두게 해 준다. 둘을 잘 조합하면 연구 생산성은 분명히 높아진다. 그러나 최종 분석 기준, 데이터 품질 판단, 결과 해석, 임상적 의미 부여는 여전히 사람이 책임져야 한다. 바이브 코딩의 핵심은 속도가 아니라, 빠른 초안을 검증 가능한 연구 자산으로 바꾸는 규율이다.

핵심 개념 정리

- Kiro는 spec-driven coding workflow를 강조하는 IDE이고, Bedrock은 모델 운영 계층이다.
- AI 코딩 도구는 파일 파서, 문서화, 작은 자동화 코드, 테스트 초안 같은 반복 작업에 특히 강하다.
- omics 분석에서는 그럴듯한 코드보다 검증 가능한 코드가 더 중요하다.
- 바이브 코딩은 결과를 자동 신뢰하는 방식이 아니라, 초안을 빨리 만들고 사람이 검토하는 방식으로 써야 안전하다.

복습 질문

1. Kiro와 Claude on Bedrock은 각각 어떤 문제를 해결하며, 왜 같은 도구로 보면 안 되는가?

2. omics 분석에서 AI 코딩 도구가 잘하는 일과 사람이 끝까지 책임져야 하는 일은 각각 무엇인가?
3. AI가 생성한 코드를 연구 코드로 승격하기 전에 어떤 검증 절차가 필요한가?

Further Reading

- Kiro. Official site.
- Anthropic. Claude on Amazon Bedrock.
- Anthropic. Claude Code on Amazon Bedrock.

References

- Anthropic. 2026a. Claude on Amazon Bedrock.
- Anthropic. 2026b. Claude Code on Amazon Bedrock.
- AWS. 2026. Model parameters for Anthropic Claude models.
- Kiro. 2026a. Official site.
- Kiro. 2026b. Pricing.
- Lee et al. 2026. Vibe Coding Omics Data Analysis Applications.
- NCBI. 2026. GeneGPT repository.

15장. 시퀀싱 데이터에서 변이 해석까지 – 통합 예제

학습 목표

- 시퀀싱 센터 데이터 handoff부터 최종 질의와 해석까지의 전체 흐름을 설명할 수 있다.
- data lake, lakehouse, warehouse가 하나의 워크플로 안에서 어떻게 이어지는지 이해할 수 있다.
- S3, HealthOmics, Athena, Bedrock, Quick이 각각 어느 단계에서 역할을 하는지 설명할 수 있다.

핵심 질문

- 시퀀싱 회사에서 데이터를 넘겨받은 뒤, 무엇부터 구조화해야 하는가
- 원시 데이터, 분석용 테이블, 요약 리포트를 왜 같은 저장 방식으로 다루면 안 되는가
- 변이 해석까지 가는 통합 워크플로에서 AWS 서비스들은 어떤 순서로 연결되는가

통합 워크플로를 하나의 이야기로 보기

이 책의 앞 장들은 개별 서비스와 개별 개념을 설명했지만, 실제 연구실 운영은 늘 통합 워크플로의 형태로 나타난다. 시퀀싱 회사에서 FASTQ나 CRAM이 도착하고, 이를 등록하고, 전처리와 정렬과 변이 호출을 돌리고, annotation을 붙이고, cohort 수준으로 질의하고, 마지막에 해석과 보고를 수행하는 과정은 하나의 긴 파이프라인이다. 학생에게는 이 흐름을 서비스별 목록보다 로 보여 주는 편이 훨씬 이해하기 쉽다. 같은 AWS라도 어떤 단계는 data lake에 가깝고, 어떤 단계는 lakehouse에, 어떤 단계는 warehouse에 가깝다. 따라서 통합 예제 장에서는 기술 명칭보다 데이터의 생애주기를 먼저 보는 것이 중요하다.

가장 단순한 실전 예제는 다음과 같이 그릴 수 있다. vendor 또는 시퀀싱 센터가 데이터를 S3나 HealthOmics sequence store로 전달한다. 연구실은 이를 sample metadata와 연결하고, Nextflow 또는 HealthOmics workflow를 돌려 정렬과 variant calling을 수행한다. 결과 VCF와 annotation table은 Athena나 Spark에서 질의 가능한 구조로 정리되고, 마지막에는 Bedrock이나 dashboard 계층을 통해 사람이 해석하고 공유한다. 이 흐름은 단순한 자동화가 아니라, 원시 파일이 점점 더 구조화된 연구 자산으로 변해 가는 과정이다.

Vendor handoff – read set 등록과 메타데이터 정리

통합 워크플로의 첫 단계는 파일 자체보다 메타데이터다. 시퀀싱 센터에서 파일이 도착했다고 해서 분석 준비가 끝난 것은 아니다. 샘플 ID, 라이브러리 정보, 시퀀싱 플랫폼, 참조 유전체 버전, 승인 상태, 프로젝트 코드, 환자 또는 코호트와의 연결 규칙이 함께 정리되어야 한다. HealthOmics sequence store와 read set 개념이 유용한 이유는 바로 이 단계에서

데이터의 등록과 provenance를 더 명시적으로 관리할 수 있기 때문이다. 단순히 폴더에 파일을 복사해 두는 방식은 빠를 수 있지만, 이후 누가 어떤 샘플을 어떤 파이프라인으로 분석했는지 추적하기가 훨씬 어려워진다.

실전에서는 메타데이터 JSON이나 manifest 파일이 워크플로의 진짜 출발점이 된다. 예를 들어 `sample_id`, `read_set_id`, `reference`, `pipeline_version`, `phenotype_group` 같은 정보가 구조화되어 있으면, 이후 Lambda나 Step Functions, HealthOmics workflow가 이를 기준으로 자동화될 수 있다. 반대로 파일만 있고 메타데이터가 흩어져 있으면, 아무리 좋은 클라우드 인프라를 가져와도 운영은 쉽게 불안정해진다. 학생에게는 보다 는 표현을 가르치는 편이 더 정확하다. 분석은 파일을 대상으로 하는 것 같지만, 실제 운영은 등록된 샘플과 메타데이터를 대상으로 이루어지기 때문이다.

Data lake, lakehouse, warehouse가 한 파이프라인 안에서 만나는 방식

omics 연구에서 data warehouse, data lake, lakehouse는 경쟁 개념이 아니라 서로 다른 데이터 상태를 가리키는 편이 더 맞다. 원시 FASTQ, BAM, CRAM, raw image는 보통 data lake 계층에 가깝다. 이 단계의 핵심은 원본 보존과 유연한 저장이며, 스키마를 엄격히 강제하기보다 나중 분석을 위해 자산을 남겨 두는 데 의미가 있다. 반면 annotation이 붙은 variant table, cohort filtering table, count matrix summary, quality summary 같은 반복 질의용 데이터는 lakehouse 계층에 더 가깝다. 여기서는 Parquet, S3 Tables, Glue catalog, Athena 같은 구조가 중요해진다. 마지막으로 임상 리포트용 요약 표, 운영 대시보드, PI용 cohort summary는 warehouse에 가까운 계층으로 볼 수 있다.

이 구분은 이론적 분류가 아니라 실제 설계 도구다. 원시 FASTQ를 바로 Athena로 질의하려 하거나, 임상 리포트를 만들 데이터를 raw data zone에 그대로 두면 운영이 금방 복잡해진다. 반대로 반복 질의가 필요한 annotation 결과를 계속 VCF와 TSV 상태로만 남겨 두면, cohort 분석과 dashboard 작성이 매우 비효율적이 된다. 따라서 통합 예제에서는 `lake`, `lakehouse`, `warehouse` 라는 큰 원칙을 분명히 해 두는 것이 좋다. 학생은 이를 통해 서비스 선택보다 먼저 데이터 계층을 나누는 사고를 배우게 된다.

Table 1. 통합 오믹스 워크플로에서의 데이터 계층

계층	대표 데이터	AWS 예시
Data lake	FASTQ, BAM, CRAM, raw image, read set	S3, HealthOmics sequence store
Lakehouse	annotation table, cohort filtering table, QC summary	S3 Tables, Parquet, Glue catalog, Athena
Warehouse	dashboard dataset, clinical report mart, 운영 요약표	Athena 결과셋, Amazon Quick, 별도 보고 계층

전처리와 분석 – HealthOmics, Nextflow, Batch의 역할

원시 데이터가 등록되면 다음 단계는 workflow 실행이다. 이때 연구실은 크게 두 가지 길을 선택할 수 있다. 하나는 HealthOmics처럼 관리형 워크플로 계층을 사용하는 것이고, 다른 하나는 Batch와 Nextflow 같은 범용 조합을 직접 운영하는 것이다. 전자는 실행 환경을 표준화하고 재시도와 provenance를 더 쉽게 관리하는 데 강하고, 후자는 더 큰 자유도와 커스터마이징을 제공한다. 통합 예제 장에서는 어느 것이 더 우월하다고 말하기보다, 는 점을 보여 주는 편이 좋다.

예를 들어 WGS나 panel 분석의 전처리 파이프라인은 `input manifest -> alignment -> variant calling -> QC -> output VCF/BAM`의 큰 흐름을 가진다. 이 과정은 workflow 실행 단계에서는 여전히 compute 중심처럼 보이지만, 사실은 동시에 데이터 계층을 바꾸는 단계이기도 하다. 원시 FASTQ는 정렬과 변이 호출을 거치며 분석 결과 자산으로 변하고, 이후 annotation과 cohort query를 위해 다른 구조로 다시 정리된다. 학생은 여기서 "파이프라인이 끝났다"보다 "데이터가 다음 계층으로 변환되었다"는 시각을 가져야 한다. 그래야 후속 Athena 질의와 Bedrock 인터페이스가 왜 필요한지 자연스럽게 이해된다.

테이블화와 cohort query – Athena가 들어오는 지점

변이 해석으로 가기 전에 반드시 필요한 단계가 구조화된 테이블화다. VCF 자체는 여전히 매우 중요하지만, cohort 질의나 운영 요약, phenotype과의 결합, frequency 비교 같은 작업에는 컬럼형 테이블과 SQL 계층이 훨씬 편리한

경우가 많다. Athena는 바로 이 지점에서 강점을 가진다. annotation이 끝난 variant table이나 sample-level QC table, phenotype metadata를 Glue catalog와 연결해 두면, 연구자는 복잡한 파서 없이도 반복 질의를 수행할 수 있다. 이 과정이 바로 lakehouse 사고의 실전적 구현이다.

학생에게는 이 단계가 왜 중요한지 구체적 질문으로 보여 주는 것이 좋다. 예를 들어 `BRCA1 BRCA2 pathogenic variant , phenotype group rare damaging variant burden , ancestry group allele frequency` 같은 질문은 Athena와 구조화된 테이블이 있을 때 훨씬 자연스럽다. 반대로 모든 것을 VCF 원본 위에서만 해결하려 하면, 분석은 가능하더라도 반복성과 협업성이 크게 떨어진다. 즉 변이 해석의 시작은 모델이 아니라, 질의 가능한 테이블을 만드는 일이다.

자연어 해석과 보고 – Bedrock과 Quick의 위치

마지막 단계에서 비로소 생성형 AI와 시각화 계층이 등장한다. Bedrock은 구조화된 데이터가 이미 준비된 뒤에, 자연어 질문을 더 쉽게 던지고 결과를 요약하고 보고 초안을 만드는 해석 계층으로 이해하는 편이 정확하다. 이것은 `AI VCF` 는 뜻이 아니다. 오히려 앞 단계에서 annotation, 테이블화, provenance 정리가 충분히 이루어졌기 때문에, 마지막에 자연어 인터페이스가 유용해지는 것이다. 따라서 학생에게는 `AI` 라는 메시지를 분명히 남겨야 한다.

Amazon Quick 같은 시각화 계층도 비슷하다. Quick은 직접 유전체 계산을 수행하는 서비스가 아니라, cohort dashboard, 운영 지표, 결과 공유를 위한 상위 요약 계층에 더 가깝다. 즉 Bedrock이 질의와 설명을 돕는 자연어 계층이라면, Quick은 요약과 관찰을 돕는 시각화 계층이다. 통합 워크플로의 끝은 반드시 거대한 알고리즘이 아니라, 사람이 이해하고 공유할 수 있는 형태로 결과를 정리하는 것이다. 이 점까지 포함해야 시퀀싱 데이터에서 해석까지의 흐름이 완성된다.

핵심 개념 정리

- 통합 오믹스 워크플로는 `vendor handoff -> workflow -> table -> ->` 의 흐름으로 보는 것이 가장 직관적이다.
- 원시 데이터, 분석용 테이블, 요약 보고용 데이터는 같은 계층에 두지 않는 것이 좋다.
- Athena와 같은 SQL 계층은 변이 해석 이전에 cohort 질의를 구조화하는 데 매우 중요하다.
- Bedrock과 Quick은 분석의 시작점이 아니라, 구조화된 데이터 위에 얹히는 해석과 시각화 계층이다.

복습 질문

1. 시퀀싱 센터에서 파일이 도착한 뒤, 왜 파일 복사보다 메타데이터 등록이 먼저인가?
2. data lake, lakehouse, warehouse는 통합 omics 워크플로 안에서 각각 어떤 역할을 하는가?
3. Bedrock과 Quick은 왜 raw data 단계가 아니라 구조화된 분석 결과 단계에서 더 유용한가?

Further Reading

- AWS. Sequence stores.
- AWS. Workflow definition specifics for Nextflow.
- AWS Machine Learning Blog. Accelerating genomics variant interpretation with AWS HealthOmics and Amazon Bedrock AgentCore.

References

- AWS. 2026a. Sequence stores.
- AWS. 2026b. Accessing HealthOmics read sets with Amazon S3 URIs.
- AWS. 2026c. Workflow definition specifics for Nextflow.
- AWS. 2026d. What is Amazon Athena?.
- AWS. 2026e. What is Amazon Quick?.
- AWS Machine Learning Blog. 2025. Accelerating genomics variant interpretation with AWS HealthOmics and Amazon Bedrock AgentCore.
- AWS Industries Blog. 2024. Orchestrating multiple AWS HealthOmics workflows at scale.

16장. 비용, 보안, 재현성을 함께 설계하기

학습 목표

- 클라우드 분석에서 비용, 보안, 재현성이 왜 동시에 중요해지는지 설명할 수 있다.
- 유전체 데이터의 민감성과 접근 제어 원칙을 이해할 수 있다.
- 태그, 로그, 버전 관리, 환경 기록 같은 운영 습관을 설명할 수 있다.

핵심 질문

- 왜 계산보다 저장과 전송이 더 큰 비용이 될 수 있는가
- 공개 데이터와 민감한 임상 데이터는 어떻게 다르게 다뤄야 하는가
- 실험을 다시 재현하려면 무엇을 남겨야 하는가

비용 구조 이해하기

클라우드 분석의 비용을 이야기할 때 초보자는 보통 인스턴스 시간부터 떠올린다. 실제 청구서에서 눈에 띄는 항목이 compute인 경우가 많기 때문이다. 그러나 오믹스 분석이 커질수록 비용은 훨씬 더 복합적인 구조를 가진다. 데이터를 얼마나 오래 저장하는지, 몇 번 다시 읽는지, 리전이나 가용 영역을 넘어 몇 번 이동하는지, 실패한 작업을 얼마나 자주 처음부터 다시 실행하는지, 로그를 얼마 동안 남기는지가 모두 비용을 만든다. 다시 말해 클라우드 비용은 "어떤 서버를 썼는가"보다 "어떤 생애주기로 데이터를 굴렸는가"에 더 가깝다.

이 점은 오믹스 분석에서 특히 중요하다. FASTQ와 BAM, CRAM, VCF는 파일 하나하나가 크고, 중간 산물은 순간적으로 더 커지기 쉽다. 정렬과 recalibration, joint calling, annotation, cohort aggregation을 거치는 동안 같은 샘플이 여러 형식으로 존재할 수 있고, 각 단계의 임시 파일이 예상보다 오래 남으면 저장비가 조용히 누적된다. 반면 compute는 대개 실행이 끝나면 비용도 멈춘다. 그래서 경험이 쌓일수록 연구자는 "비용은 서버가 아니라 생애주기에서 샌다"는 감각을 갖게 된다.

Table 1은 교육용으로 단순화한 비용 구조다. 이 표의 핵심은 절대액을 외우는 것이 아니라, 어떤 설계 선택이 어떤 종류의 비용으로 이어지는지를 연결해서 보는 데 있다. 이 장에서는 원칙 수준에서만 다루고, 세부 누수 지점은 다음 장에서 더 실전적으로 해부한다.

Table 1. 유전체 클라우드 분석의 대표 비용 축

비용 축	대표 원인	초보자가 놓치기 쉬운 점
컴퓨트	과도한 인스턴스 크기, 잘못된 instance family, 불필요한 재실행	가장 잘 보이지만 동시에 가장 통제하기 쉬운 항목이다
스토리지	원본 장기 보관, 중간 산물 방치, lifecycle 미설계	분석이 끝난 뒤에도 계속 남아 청구서를 만든다
요청 비용	작은 파일 폭증, 반복적인 LIST/GET, 무분별한 테이블 스캔	저장 단가보다 눈에 덜 띄지만 장기적으로 누적된다
데이터 이동	리전 간 이동, AZ 간 이동, 외부 다운로드 공유	"같은 AWS 안이니 무료"라고 오해하기 쉽다
반복 실행	resume 부재, checkpoint 부재, 환경 불일치로 인한 재시도	비용이자 재현성 저하의 신호다

스토리지 계층 선택도 이 틀 안에서 이해해야 한다. `S3 Standard`는 단순하고 빠르지만 계속 비싸고, `Intelligent-Tiering`은 접근 패턴이 흔들리는 데이터에 유리하며, `Standard-IA`와 Glacier 계열은 장기 보관에 도움이 되지만 retrieval 비용과 최소 저장 기간을 함께 고려해야 한다. 여기서 중요한 것은 가장 싼 storage class를 고르는 일이 아니라, 지금 이 데이터가 `hot`, `warm`, `cold` 중 어디에 해당하는지를 꾸준히 구분하는 습관이다. 비용 최적화는 구매가 아니라 분류의 문제라는 점을 학생에게 초반부터 심어 줄 필요가 있다.

최소 권한, 암호화, 접근 제어

유전체 데이터는 공개 참조 자원과 민감한 환자 데이터가 한 프로젝트 안에 함께 존재하기 쉽다. 따라서 보안은 "모든 것을 다 잠근다"가 아니라, 데이터 등급에 따라 다른 접근 규칙을 적용하는 일이다. gnomAD나 SRA처럼 공개 접근이 전제된 자원은 직접 읽는 편이 자연스럽지만, 환자 샘플 기반 FASTQ나 임상 annotation 테이블은 같은 방식으로 다루면 안 된다. 연구실 운영에서 중요한 것은 이런 차이를 문장으로 명시하는 것이다. 어떤 데이터가 공개용인지, 어떤 데이터가 내부 분석 전용인지, 어떤 결과가 외부 협업자에게 공유 가능한지를 미리 정해 두지 않으면 기술적 설정도 일관되기 어렵다.

최소 권한 원칙은 이 구분을 AWS 권한 모델로 옮긴 것이다. 사람은 필요한 계정과 리소스만 보아야 하고, 워크플로는 필요한 입력을 읽고 필요한 출력 위치에만 쓸 수 있어야 한다. 장기 액세스 키를 노트북이나 스크립트에 넣어 두는 방식은 작은 연구실에서는 편해 보일 수 있지만, 팀이 커질수록 누가 어떤 권한으로 무엇을 했는지 추적하기 어려워진다. 따라서 사람은 IAM Identity Center나 federation을 통해 임시 권한으로 접근하고, 실행 주체는 IAM Role을 통해 필요한 범위만 허용하는 구조가 기본이 된다.

암호화도 마찬가지다. 저장 중 암호화와 전송 중 암호화는 이제 특별한 기능이 아니라 기본 운영 가정으로 보는 편이 맞다. 다만 암호화만으로 충분하다고 생각하면 안 된다. 누가 그 데이터에 접근할 수 있는지, 어떤 네트워크 경로를 타는지, 감사 로그를 남기는지까지 함께 설계해야 실제 보호가 된다. 민감한 임상 데이터가 포함된 프로젝트에서는 S3 버킷 정책, KMS 키 접근 권한, private network 경로, 로그 감사, 외부 공유 절차가 한 묶음으로 이해되어야 한다. 반대로 공개 데이터는 지나치게 폐쇄적으로 다루기보다, 적절한 direct access를 활용해 복사와 중복 저장을 줄이는 편이 더 나을 수 있다.

Table 2는 데이터 등급에 따라 접근 원칙이 어떻게 달라지는지 교육용으로 정리한 것이다. 이는 법률 자문표가 아니라 운영 감각을 잡기 위한 첫 지도다. 실제 프로젝트에서는 기관의 IRB, 계약, 데이터 사용 동의 범위, 국가별 규정이 더해질 수 있으므로, 민감 데이터는 항상 기관 정책과 함께 검토해야 한다.

Table 2. 데이터 등급별 기본 접근 원칙

데이터 유형	대표 예시	기본 원칙
공개 참조 데이터	gnomAD, SRA public data, 공개 annotation	가능한 한 직접 참조하고 불필요한 복사를 줄인다
내부 연구 데이터	비식별 cohort table, 내부 QC 결과	프로젝트 단위 권한과 태그로 접근 범위를 제한한다
민감 임상 데이터	환자 유래 read set, 임상 annotation, 식별 가능 메타데이터	최소 권한, 암호화, 감사 로그, 공유 승인 절차를 함께 설계한다

하이브리드 클라우드가 여전히 중요한 이유도 여기서 드러난다. 모든 데이터를 같은 규칙으로 한꺼번에 클라우드로 밀어 넣는 것이 정답은 아니다. 법적 또는 네트워크 제약이 큰 데이터는 기관 내부에 두고, 공개 데이터 활용과 대규모 병렬 계산은 AWS에서 수행하는 혼합 구조가 더 현실적일 수 있다. 중요한 것은 어디에 두느냐 자체보다, 데이터 등급과 권한 모델이 그 위치 선택과 맞물려 있어야 한다는 점이다.

컨테이너와 환경 고정

재현성은 흔히 Docker를 쓰면 끝나는 문제처럼 설명되지만, 실제로는 훨씬 넓다. 같은 결과를 다시 얻으려면 최소한 워크플로 정의, 컨테이너 이미지, 입력 파라미터, 참조 유전체 버전, annotation 자원 버전, 실행 시점의 서비스 설정을 함께 고정해야 한다. 스크립트 파일만 공유하는 방식은 종종 "무엇을 실행했는가"는 남겨도 "어떤 환경에서 실행했는가"를 충분히 남기지 못한다. 그래서 WDL, CWL, Nextflow 같은 workflow language가 중요하다. 이들은 파이프라인의 논리뿐 아니라 실행 단위와 입력, 출력, 의존성을 더 명시적으로 표현하게 만든다.

AWS HealthOmics의 private workflow 기능은 이런 관점에서 유용하다. HealthOmics는 WDL, CWL, Nextflow 정의를 직접 받아들이고, workflow versioning과 call caching 같은 기능을 제공한다. workflow versioning은 "업데이트된 워크플로가 언제부터 어떤 결과를 만들었는가"를 분리해서 기록하게 해 주고, call caching은 이미 계산된 태스크 출력을 재사용해 비용을 줄이는 동시에 같은 작업을 불필요하게 반복하지 않게 해 준다. 이 기능들은 단순한 편의 기능이 아니라, 재현성을 운영 습관으로 끌어올리는 장치라고 보는 편이 맞다.

컨테이너도 태그만으로는 부족하다. `latest` 같은 느슨한 태그는 나중에 같은 이름이 다른 이미지를 가리킬 수 있기 때문이다. 따라서 재현성을 정말 중요하게 생각한다면 이미지 digest, workflow definition 버전, parameter JSON, reference build, annotation database 버전을 함께 남겨야 한다. 오믹스 분석에서는 GRCh37과 GRCh38의

차이만으로도 결과 해석이 크게 달라질 수 있고, ClinVar나 VEP annotation release 차이도 의미 있는 변화를 만든다. 결국 재현 가능한 분석은 "같은 코드를 다시 돌린다"보다 "같은 문맥을 다시 구성할 수 있다"에 더 가깝다.

동등성 검증과 provenance 남기기

재현성 논의에서 자주 빠지는 마지막 단계가 동등성 검증이다. 환경을 고정했다고 해서 결과가 자동으로 과학적으로 동일하다고 단정할 수는 없다. 실제 운영에서는 인스턴스 세대가 바뀌고, 워크플로가 새 버전으로 이전되고, FPGA나 GPU 같은 가속 계층이 바뀌기도 한다. 이때 중요한 것은 "문제 없이 돌아갔다"가 아니라 "동등한 결과를 만들었는가"를 검증하는 것이다. 이런 태도는 연구실 운영을 훨씬 성숙하게 만든다.

최근 AWS HPC 블로그의 F1 대 F2 평가 사례는 이를 매우 실무적으로 보여 준다. 이 검증은 단순한 성능 비교로 끝나지 않고, command provenance, DRAGEN metrics, VCF concordance를 함께 확인했다. 즉 더 빠르고 더 싸다는 주장만 한 것이 아니라, 동일한 파이프라인이 동일한 변이 결과를 유지하는지 `bcftools isec` 같은 도구로 검증했다. 이런 접근은 학생에게 재현성을 가르칠 때도 좋은 모델이 된다. 컨테이너를 썼다는 사실만으로 만족하지 말고, 바뀐 환경이 결과를 바꾸지 않았는지 실제 산출물 수준에서 확인해야 한다는 메시지를 주기 때문이다.

또한 바이트 수준 동일성과 과학적 동일성을 구분하는 감각도 필요하다. AWS HealthOmics storage 문서는 imported file과 exported file 사이에 bitwise equivalence가 항상 보존되지는 않을 수 있지만, HealthOmics ETag와 provenance metadata를 통해 read set의 semantic identity를 유지한다고 설명한다. 이는 교육적으로 아주 좋은 예시다. 과학적으로 같은 read set을 가리키는 정보와, 압축 방식이나 내부 표현 때문에 바이트가 정확히 일치하는지는 다른 문제라는 뜻이기 때문이다. 따라서 provenance는 단지 파일 이름을 적는 일이 아니라, 그 데이터가 어디서 왔고 어떤 의미적 동일성을 가지는지 남기는 일이다.

Table 3은 최소한 남겨 두어야 할 provenance 항목을 정리한 것이다. 이 목록이 완벽한 표준은 아니지만, 여기의 절반만 충실히 남겨도 재현성과 협업 품질은 크게 좋아진다.

Table 3. 재현성을 위한 최소 provenance 기록

항목	예시	남기는 이유
입력 데이터 위치	S3 URI, sequence store read set ID	어떤 원본을 썼는지 추적하기 위해
샘플 메타데이터	subject ID, sample ID, library ID	샘플 혼동을 방지하기 위해
워크플로 버전	workflow name, version, git commit	어떤 논리로 실행했는지 남기기 위해
컨테이너 정보	image URI, digest	실행 환경을 다시 구성하기 위해
파라미터 파일	JSON/YAML parameter set	같은 실행 조건을 복원하기 위해
참조 자원	reference build, annotation release	해석 차이의 원인을 추적하기 위해
동등성 검증 결과	metrics diff, concordance summary	새 환경 전환 시 결과 보존을 확인하기 위해

연구실 운영 규칙 만들기

좋은 운영 규칙은 연구를 느리게 만드는 절차가 아니라, 같은 실수를 반복하지 않게 해 주는 압축된 기억이다. 특히 학생과 포닥이 자주 바뀌는 연구실에서는 "다음 사람도 같은 방식으로 할 수 있게 만드는 규칙"이 곧 생산성이다. 이런 규칙은 길 필요가 없다. 오히려 짧고 반복 가능할수록 좋다.

예를 들어 다음과 같은 다섯 가지 원칙만 지켜도 연구실 운영 품질은 눈에 띄게 좋아진다. 첫째, 모든 실행에는 manifest와 parameter file을 남긴다. 둘째, 원본 경로와 결과 경로를 절대 섞지 않는다. 셋째, 태그 없는 주요 리소스는 만들지 않는다. 넷째, 30분 이상 걸리는 batch workload는 resume, retry, checkpoint 전략을 함께 설계한다. 다섯째, 로그 retention과 storage lifecycle을 기본값으로 명시한다. 이 다섯 가지는 비용을 줄이기 위한 규칙이면서 동시에 보안을 돕고, 재현성도 높이는 규칙이다.

Table 4. 연구실 운영 규칙의 첫 버전

규칙	이유	최소 구현
실행 전 manifest 고정	샘플 혼동과 재현성 저하를 막기 위해	입력 URI, sample ID, reference build를 표준 템플릿에 기록
원본과 결과 분리	덮어쓰기 사고를 막기 위해	`raw/`, `processed/`, `reports/` prefix 고정
태그 필수화	비용 배분과 권한 제어를 쉽게 하기 위해	`project_id`, `owner`, `data_class` 최소 태그 적용
retry와 checkpoint 사용	Spot, 실패 복구, 비용 절감을 위해	workflow 수준 resume와 task retry 기본 활성화
retention 명시	로그와 중간 산물의 무한 축적을 막기 위해	log group retention과 lifecycle rule을 프로젝트 시작 시 설정

비용, 보안, 재현성은 서로 다른 세 과목이 아니다. 같은 데이터를 오래 두고, 아무나 접근하게 하고, 실행 문맥을 남기지 않으면 결국 셋 모두가 동시에 나빠진다. 반대로 최소 권한, 표준 경로, 버전 고정, provenance 기록, lifecycle 설계를 초반부터 습관으로 만들면 셋 모두가 함께 좋아진다. 클라우드 운영이 어려운 이유는 요소가 많기 때문이 아니라, 이 요소들이 서로 얽혀 있기 때문이다. 이 얽힘을 이해하는 순간 연구자는 비로소 “서비스를 쓰는 사람”에서 “연구 인프라를 설계하는 사람”으로 한 단계 올라간다.

핵심 개념 정리

- 클라우드 비용은 compute만의 문제가 아니라 저장, 이동, 요청, 반복 실행이 함께 만드는 구조다.
- 보안은 모든 데이터를 똑같이 잠그는 일이 아니라 데이터 등급에 맞는 권한 모델을 설계하는 일이다.
- 재현성은 컨테이너 하나로 끝나지 않으며, workflow version, parameter, reference, provenance 기록까지 포함한다.
- 동등성 검증은 재현성의 마지막 단계이며, 환경 전환 시 결과가 실제로 유지되는지 확인하게 해 준다.

복습 질문

1. 왜 오믹스 분석에서는 compute 비용보다 저장과 전송과 재실행 비용이 더 크게 문제 될 수 있는가?
2. 민감한 임상 데이터와 공개 참조 데이터는 왜 같은 권한 모델로 다루면 안 되는가?
3. 컨테이너를 사용하고도 재현성이 부족해질 수 있는 이유를 provenance 관점에서 설명해 보라.

Further Reading

- AWS. IAM best practices.
- AWS HPC Blog. Evaluating next-generation cloud compute for large-scale genomic processing.
- Strozzi et al. Scalable Workflows and Reproducible Data Analysis for Genomics.

References

- AWS. 2026a. IAM best practices.
- AWS. 2026b. HealthOmics storage.
- AWS. 2026c. Private workflows in HealthOmics.
- AWS. 2026d. Amazon S3 storage classes overview.
- AWS Batch. 2026. Use Amazon EC2 Spot best practices for AWS Batch.
- AWS HPC Blog. 2026. Evaluating next-generation cloud compute for large-scale genomic processing.
- AWS Industries Blog. 2023. Secure your genomic workflows and data with AWS HealthOmics.
- Strozzi F, Janssen R, Wurmus R, et al. 2019. Scalable Workflows and Reproducible Data Analysis for Genomics.

17장. 한국 연구실을 위한 운영 체크리스트

학습 목표

- 한국 연구실 환경에서 AWS 도입 시 자주 부딪히는 실무 이슈를 설명할 수 있다.
- 학생, 포닥, PI가 각각 챙겨야 할 운영 포인트를 구분할 수 있다.
- 작은 연구실이 무리 없이 시작하는 단계적 도입 전략을 설계할 수 있다.

핵심 질문

- 처음부터 모든 것을 클라우드로 옮길 필요가 있을까
- 누가 계정을 만들고 누가 비용을 감시해야 하는가
- 로컬 서버와 클라우드를 어떻게 병행할 것인가

최소 시작 구성

한국 연구실이 AWS를 도입할 때 가장 먼저 버려야 할 생각은 “처음부터 완벽한 클라우드 연구소를 만들어야 한다”는 압박이다. 실제로 작은 연구실의 병목은 서비스가 부족해서 생기기보다, 데이터가 여기저기 흩어져 있고, 누가 어떤 버킷에 무엇을 올렸는지 모르고, 샘플 이름과 결과 경로가 사람마다 달라서 생기는 경우가 훨씬 많다. 따라서 출발점은 거대한 플랫폼이 아니라, 반복 가능한 최소 운영 단위를 만드는 일이어야 한다. 한 개의 표준 데이터 전달 경로, 한 개의 표준 실행 경로, 한 개의 표준 결과 정리 규칙만 갖추어도 연구실 운영은 크게 안정된다.

작은 연구실의 첫 구성은 놀랄 만큼 단순해도 된다. 공용 저장소는 Amazon S3 하나로 시작할 수 있고, 실행 계층은 AWS HealthOmics 또는 Nextflow 기반 AWS Batch 중 한 가지로만 먼저 정하면 된다. 여기에 사람 접근은 IAM Identity Center로 묶고, 비용 감시는 AWS Budgets 알림 한두 개만 걸어 두어도 상당수의 실수는 초반에 막을 수 있다. 중요한 것은 서비스 수를 늘리는 일이 아니라, 연구실 구성원 모두가 “원본은 어디로 들어오고, 분석은 어디서 돌고, 결과는 어디에 남는가”를 같은 문장으로 설명할 수 있게 만드는 일이다.

Table 1은 한국 연구실이 무리 없이 시작할 수 있는 최소 구성을 정리한 것이다. 이 표의 목적은 이상적인 엔터프라이즈 설계를 강요하는 것이 아니라, 는 감각을 주는 데 있다. 한 학기 안에 학생이 바뀌고, 과제 단위로 예산이 달라지고, 시퀀싱 센터와의 전달 방식도 프로젝트마다 달라질 수 있는 환경에서는 처음 구조가 단순할수록 유지가 쉽다.

Table 1. 작은 연구실을 위한 최소 시작 구성

구성 요소	최소 선택	왜 필요한가
공용 저장소	Amazon S3 버킷 1개와 고정된 prefix 규칙	원본, 중간 결과, 최종 결과의 위치를 연구실 전체가 공유하기 위해
실행 계층	AWS HealthOmics 또는 Nextflow + AWS Batch 중 1개	표준 파이프라인을 같은 방식으로 반복 실행하기 위해
사용자 접근	IAM Identity Center 기반 그룹 접근	학생 교체와 역할 변경 시 권한 회수를 단순화하기 위해
비용 감시	AWS Budgets 월간 예산 알림	청구서를 나중에 보는 대신 도중에 이상 징후를 잡기 위해
데이터 전달	시퀀싱 센터의 S3 납품 또는 DataSync	이메일 첨부나 수동 복사를 없애기 위해
운영 문서	짧은 README 또는 runbook 1개	새 구성원이 같은 절차로 시작할 수 있게 하기 위해

실제 연구실에서는 이 최소 구성이 곧 혼합 전략으로 이어진다. 모든 FASTQ와 BAM을 무조건 sequence store로 넣을 필요는 없다. 납품 직후 빠르게 분석해야 하는 파일은 S3에 두고 바로 Nextflow나 Batch로 넘길 수 있고, 반복 공유와 장기 관리가 필요한 read set은 AWS HealthOmics sequence store로 가져가는 방식이 더 자연스러울 수 있다. 즉 최소 시작 구성의 핵심은 특정 서비스에 올인하는 것이 아니라, , 를 연구실 수준에서 일관되게 정하는 데 있다.

계정, 예산, 권한 분리

운영이 흔들리는 연구실은 대개 기술이 부족해서가 아니라, 책임 경계가 흐려서 흔들린다. 누가 버킷을 만들 수 있는지, 누가 파이프라인을 승인하는지, 누가 예산 알림을 받는지, 누가 외부 협업자에게 결과를 공유하는지를 미리 정해 두지 않으면 작은 프로젝트도 곧 혼란스러워진다. 따라서 한국 연구실에서 AWS를 도입할 때는 기술 스택과 함께 책임 스택을 설계해야 한다. 가장 기본적인 원칙은 과 AWS 을 가능한 한 맞추는 것이다.

먼저 사람 접근은 장기 액세스 키를 개인 노트북에 뿌리는 방식보다 IAM Identity Center 기반의 그룹 접근이 훨씬 낫다. 학생과 포닥, 연구원은 로그인 후 역할을 받아 일시적인 권한으로 작업하고, 루트 계정은 일상 업무에 쓰지 않는 것이 기본이다. 이 구조는 단순히 보안을 강화하기 위한 것이 아니라, 연구실 구성원이 졸업하거나 프로젝트가 끝날 때 권한 정리를 쉽게 만들기 위한 운영 장치이기도 하다. 한 번 생성된 장기 키는 잊히기 쉽지만, 역할 기반 접근은 사람의 소속과 함께 비교적 자연스럽게 정리된다.

비용 관리도 권한 관리와 분리해서 보면 안 된다. 학생이 분석을 실행하는 역할을 가진다면, PI나 실무 관리자는 예산과 알림을 보는 역할을 따로 가져야 한다. AWS Budgets는 월 예산 한도를 정하는 도구이기 전에, 연구실이 "이번 달에 어떤 프로젝트가 얼마나 자원을 쓰고 있는가"를 조기에 감지하게 해 주는 운영 장치다. 실제 청구서는 한 달 뒤에 나오지만, 예산 알림은 그 전에 조정할 시간을 준다. 특히 과제비 단위로 운영되는 연구실에서는 와 를 같이 설계해야 나중에 비용 설명이 쉬워진다.

이때 태그와 경로 규칙은 사소한 취향이 아니라 회계와 재현성의 공통 언어가 된다. 예를 들어 모든 주요 리소스와 결과 산물에 `project_id`, `sample_id`, `subject_id`, `data_class`, `owner`, `grant_id` 같은 최소 태그 세트를 적용하면, 비용 배분과 접근 제어와 결과 정리가 동시에 쉬워진다. 특히 HealthOmics storage는 `omics:subjectId`와 `omics:sampleId` 태그를 기반으로 접근 정책을 설계할 수 있으므로, 샘플 수준 공유가 필요한 연구실에서는 사람 이름이나 임의 문자열보다 안정적인 subject/sample 식별자를 초기에 정해 두는 편이 좋다.

Table 2는 작은 연구실에서 현실적으로 많이 쓰게 되는 역할 구분을 정리한 것이다. 여기서 핵심은 모든 사람이 모든 권한을 가져야 한다는 생각을 버리는 것이다. 연구 속도는 만드는 것보다 만드는 쪽에서 더 잘 나온다.

Table 2. 연구실 역할별 기본 책임

역할	AWS에서 맡을 일	주의할 점
PI	예산 승인, 외부 공유 승인, 민감 데이터 정책 결정	직접 실행 권한보다 감독과 승인 권한을 우선 둔다
실무 관리자 또는 시니어 연구원	버킷 구조, 태그 규칙, 예산 알림, 권한 그룹 운영	루트 계정 대신 관리 역할을 사용하고 변경 이력을 남긴다
학생/포닥	표준 manifest 작성, 파이프라인 실행, 로그 확인, 결과 해석	개별 액세스 키 저장보다 역할 기반 로그인과 표준 경로 사용을 지킨다
외부 협업자	제한된 결과 열람 또는 특정 prefix 접근	원본 전체 접근을 주지 말고 목적별 공유 범위를 제한한다

실무적으로는 prefix 규칙도 초반에 명시해 두는 편이 좋다. 예를 들어 `incoming/`, `raw/`, `processed/`, `curated/`, `reports/`, `archive/` 같은 상위 구조만 정해 두어도, 누군가가 분석 결과를 원본 폴더에 덮어쓰는 사고를 크게 줄일 수 있다. 한국 연구실에서는 사람이 바뀌는 속도가 빨라 구조적 기억이 쉽게 사라지므로, 경로 규칙은 늘 문서에 남고 예시가 함께 있어야 한다.

교육과 문서화

작은 연구실에서 가장 과소평가되는 인프라는 서버가 아니라 문서다. 클라우드 도입 초기에는 대개 한두 명이 구조를 이해하고 나머지는 따라가는 방식으로 운영된다. 이 상태가 길어지면, 그 한두 명이 바쁘거나 자리를 옮겼을 때 연구실 전체가 멈춘다. 따라서 교육의 목표는 모든 구성원을 플랫폼 엔지니어로 만드는 것이 아니라, 각자가 자기 단계에서 막히지 않도록 필요한 최소 문맥을 문서와 실습으로 넘겨 주는 데 있어야 한다.

이 점에서 역할별 교육이 중요하다. 학생과 포닥에게 Hail, SageMaker, Bedrock, Athena, HealthOmics를 모두 깊게 가르칠 필요는 없다. 오히려 , , , `manifest parameter file` 를 먼저 가르치는 편이 훨씬 실용적이다. 반대로 시니어 연구원이나 운영 담당자는 예산, 태그,

접근 제어, 데이터 전달 구조, 표준 파이프라인 버전 관리에 더 익숙해야 한다. 모두에게 같은 내용을 같은 깊이로 가르치려 하면, 누구도 실제 필요한 수준까지 도달하지 못하는 경우가 많다.

문서화도 복잡할 필요는 없다. 실제로 가장 유용한 문서는 세 종류면 충분한 경우가 많다. 첫째, 새 사람이 왔을 때 따라 하는 1페이지짜리 온보딩 문서다. 둘째, 샘플 ID, subject ID, 프로젝트 코드, 폴더 규칙을 적어 둔 네이밍 가이드다. 셋째, 분석 실패 시 무엇을 확인할지 적어 둔 runbook이다. 이 세 문서만 있어도 연구실은 개인 기억에 의존하는 상태에서 절차에 의존하는 상태로 한 단계 올라간다.

좋은 교육은 추상적인 설명보다 작은 통합 예제를 반복하는 방식으로 이루어진다. 예를 들어 한 번의 실습에서 `manifest -> S3 -> -> -> -> ` 까지 이어지게 하면, 학생은 AWS를 개별 서비스 이름이 아니라 하나의 분석 흐름으로 배우게 된다. 이 방식은 이후 더 큰 프로젝트로 확장할 때도 효과가 크다. 원리는 같고 규모만 커지기 때문이다.

또 하나 중요한 것은 연구실 문서를 한국어 기준으로 정리하는 일이다. AWS 문서는 영어로 충분히 훌륭하지만, 실제 운영 중에 자주 쓰는 규칙은 연구실 내부 언어로 짧고 명확하게 남아 있어야 한다. 예를 들어 "시퀀싱 센터 납품본은 `incoming/`에서 7일 이내에 검수한다", "분석 시작 전 manifest에 reference build를 적는다", "최종 결과는 `reports/`에 PDF와 TSV를 함께 남긴다" 같은 문장은 길 필요가 없다. 다만 누구나 같은 의미로 읽을 수 있어야 한다.

단계적 확장 로드맵

한국 연구실이 AWS를 안정적으로 도입하려면, 기술 확장은 사람과 운영의 확장 속도보다 약간만 앞서가야 한다. 처음부터 수백 샘플 cohort query, AI 기반 variant interpretation, 다중 계정 거버넌스를 한꺼번에 도입하면 기술적으로는 멋져 보여도 지속되지 않는 경우가 많다. 반대로 너무 오래 단일 EC2와 임시 폴더 구조에 머물러 있으면, 사람이 늘어날수록 혼란만 커진다. 좋은 로드맵은 연구실이 실제로 감당할 수 있는 복잡도를 한 단계씩 늘리는 방식이다.

첫 단계는 를 만드는 일이다. 원본 전달 경로, manifest, 공용 저장 위치, 예산 알림만 있어도 연구실은 수동 복사 중심 운영에서 벗어나기 시작한다. 둘째 단계는 다. 이때부터 표준 파이프라인과 버전 관리가 중요해진다. 셋째 단계는 다. VCF와 리포트만 보관하던 연구실이 cohort summary table, annotation table, Athena query로 넘어가면 분석 질문의 속도가 크게 달라진다. 마지막 단계에서야 Bedrock, Amazon Quick, 고급 lakehouse 질의 같은 상위 계층이 자연스럽게 의미를 갖는다.

Table 3은 작은 연구실이 무리 없이 따라갈 수 있는 확장 순서를 제안한다. 여기서 중요한 것은 모든 연구실이 반드시 4단계까지 가야 한다는 뜻이 아니라, 어디까지 갈지 선택하더라도 앞단의 운영 습관이 먼저 갖추어져야 한다는 점이다.

Table 3. 단계적 확장 로드맵

단계	우선 목표	대표 산출물
1단계 파일 정리	원본과 결과를 잃지 않는 구조 만들기	S3 prefix 규칙, manifest, 예산 알림
2단계 파이프라인 표준화	같은 분석을 같은 방식으로 반복하기	표준 Nextflow 또는 HealthOmics workflow, runbook
3단계 cohort 질의	결과를 표와 질의 계층으로 전환하기	annotation table, Athena query, 공유용 요약 표
4단계 해석 자동화	반복 질문과 보고를 더 빠르게 만들기	Amazon Quick 대시보드, Bedrock 기반 질의 보조, 자동 리포트

로컬 서버와 클라우드를 병행하는 전략도 이 로드맵 안에 자연스럽게 들어간다. 예를 들어 로컬 서버는 민감 데이터의 1차 보관과 가벼운 전처리에 쓰고, 대규모 병렬 실행과 공개 데이터 활용, 공동 결과 공유는 AWS에서 담당할 수 있다. 중요한 것은 어느 쪽이 더 "진짜 인프라"인가를 따지는 일이 아니라, 어떤 데이터를 어디까지 옮기고, 어디서 계산하고, 어디에 결과를 남길지 명확히 나누는 것이다. 작은 연구실일수록 이 경계가 분명할수록 운영 피로가 줄어든다.

작은 연구실이 AWS를 잘 도입하는 방법은 거대한 플랫폼을 한 번에 세우는 것이 아니라, 사람과 데이터와 비용과 문서를 함께 움직이게 하는 최소 구조를 먼저 만드는 일이다. 는 원칙은 소극적인 태도가 아니라, 실제로 오래 가는 연구 운영의 핵심 전략이다.

핵심 개념 정리

- 작은 연구실의 첫 목표는 완벽한 플랫폼이 아니라 반복 가능한 최소 운영 구조를 만드는 것이다.
- 계정, 예산, 권한 분리는 보안 문제인 동시에 책임과 운영 속도의 문제다.
- 역할별 교육과 짧은 내부 문서는 서비스 수보다 더 큰 생산성 차이를 만든다.
- 가장 현실적인 확장 방식은 -> -> cohort -> 의 순서를 따르는 것이다.

복습 질문

1. 작은 연구실이 AWS를 처음 도입할 때 모든 서비스를 한꺼번에 배우지 않아도 되는 이유는 무엇인가?
2. PI, 실무 관리자, 학생/포닥의 AWS 역할을 분리하면 어떤 운영상 이점이 생기는가?
3. 로컬 서버와 AWS를 병행하는 하이브리드 전략이 특히 유용한 상황은 어떤 경우인가?

Further Reading

- AWS. What is IAM Identity Center?.
- AWS. Creating a budget.
- AWS. Scheduling when your AWS DataSync task runs.

References

- AWS. 2026a. What is IAM Identity Center?.
- AWS. 2026b. Creating a budget.
- AWS. 2026c. Scheduling when your AWS DataSync task runs.
- AWS. 2026d. Accessing HealthOmics read sets with Amazon S3 URIs.
- AWS. 2026e. Best practices for using tag policies.
- AWS. 2026f. Best Practices for Tagging AWS Resources.
- AWS. 2026g. Compare IAM identities and credentials.

18장. Bedrock으로 하는 유전체 질의와 변이 해석

학습 목표

- Amazon Bedrock이 유전체 분석에서 어떤 역할을 할 수 있는지 설명할 수 있다.
- HealthOmics, Athena, S3 Tables와 Bedrock을 연결한 자연어 질의 구조를 이해할 수 있다.
- 변이 해석과 cohort 질의에 생성형 AI를 적용할 때 필요한 검증 원칙을 설명할 수 있다.

핵심 질문

- Bedrock은 Claude Code나 Kiro와 무엇이 다른가
- 자연어로 유전체 데이터를 묻는 시스템은 실제로 어떤 데이터 구조 위에서 동작해야 하는가
- AI가 variant interpretation을 도울 수는 있어도 왜 human review를 없앨 수는 없는가

Bedrock, AgentCore, Claude의 역할 구분

오믹스 분야에서 생성형 AI를 이야기할 때 가장 먼저 해야 할 일은 도구의 역할을 구분하는 것이다. Bedrock은 AWS 안에서 foundation model을 호출하고 관리하는 운영 계층이다. Claude는 그 위에서 호출되는 개별 모델 계열이고, AgentCore 같은 구성은 이런 모델이 도구 사용, 질의 orchestration, 애플리케이션 로직과 연결되게 만드는 상위 구조에 가깝다. 학생이 이 세 층을 구분하지 못하면, 마치 하나의 거대한 “AI 서비스”가 모든 것을 이해하고 해결하는 것처럼 오해하기 쉽다. 실제로는 모델, 운영 계층, 응용 계층이 분리되어 있을수록 시스템이 더 이해 가능하고 검증 가능해진다.

이 구분이 중요한 이유는 유전체 분석에서 신뢰가 구조에서 오기 때문이다. Bedrock은 모델 접근 권한, IAM, 리전, 운영 정책을 AWS 계정 안으로 가져온다. Claude 모델은 자연어 이해와 요약, 질의 변환, 보고서 초안 생성에 강점을 보일 수

있다. 그러나 최종적인 답의 품질은 그 아래 어떤 annotation pipeline, 어떤 ClinVar나 VEP 버전, 어떤 Athena table, 어떤 provenance를 깔아 두었는지에 달려 있다. 따라서 이 장에서는 Bedrock = , Claude = , AgentCore = 이라는 구도를 명확히 두는 것이 중요하다.

AI가 먼저가 아니라 데이터 정돈이 먼저

Bedrock을 유전체 질의에 쓰고 싶다고 할 때 초보자가 가장 자주 하는 오해는 "VCF를 AI에게 주면 해석해 준다"는 기대다. 그러나 실제 사례는 정반대 방향을 보여 준다. AWS의 Accelerating genomics variant interpretation with AWS HealthOmics and Amazon Bedrock AgentCore 글은 먼저 raw VCF를 업로드하고, 그다음 HealthOmics VEP annotation을 수행하고, 결과를 구조화된 테이블로 바꾸고, 마지막에 Athena query와 자연어 인터페이스를 연결하는 구조를 제시한다. 즉 생성형 AI는 원시 유전체 파일을 직접 읽어 임상적 답을 꺼내는 엔진이 아니라, 이미 구조화된 질의 가능한 데이터 위에 얹히는 해석 계층에 가깝다.

이 점은 교육적으로 매우 중요하다. 학생이 "AI가 유전체를 이해한다"는 식으로 접근하면, annotation과 provenance의 중요성을 놓치기 쉽다. 반대로 AI 는 원칙을 익히면, 자연어 인터페이스의 한계와 장점이 동시에 분명해진다. 자연어 질의는 접근성을 높여 주지만, 구조화된 테이블과 명시적인 해석 규칙이 없으면 그럴듯한 환각을 더 빨리 만들어 낼 뿐이다. 따라서 Bedrock 장의 핵심 메시지는 AI보다도 에 있다.

HealthOmics, S3 Tables, Athena와 연결한 유전체 질의 구조

실전 구조를 단순화하면 다음과 같다. 첫째, 원시 VCF나 cohort 결과가 들어온다. 둘째, annotation pipeline이 이를 VEP, ClinVar, 기타 규칙 기반 자원으로 풍부하게 만든다. 셋째, 결과를 S3 Tables나 Parquet 같은 질의 가능한 구조로 정리하고 Glue catalog에 등록한다. 넷째, Athena가 이를 SQL 질의 가능한 테이블로 노출한다. 다섯째, Bedrock은 사용자의 자연어 질문을 이 구조화된 질의 계층과 연결하는 인터페이스로 동작한다. 이 흐름은 -> annotation -> table -> query -> answer의 연속이다.

이 구조의 장점은 세 가지다. 첫째, 같은 질문을 사람이 SQL로도 재현할 수 있다. 둘째, provenance를 남기기 쉽다. 셋째, 자연어 답변의 근거가 어디서 왔는지 추적 가능하다. 예를 들어 Which patients have pathogenic variants in BRCA1? 같은 질문이 들어오면, 좋은 시스템은 즉석에서 상상한 답을 내놓는 대신, 병리성 분류와 유전자 기준이 들어 있는 구조화된 테이블을 질의하고 그 결과를 설명해야 한다. 학생은 여기서 자연어 인터페이스의 진짜 가치를 배운다. 즉 LLM은 데이터베이스를 대체하는 것이 아니라, 사람이 데이터베이스에 더 자연스럽게 접근하게 하는 번역 계층이 된다.

Table 1. Bedrock 기반 유전체 질의의 계층 구조

계층	대표 구성요소	역할
원시 데이터	VCF, sample metadata	입력 자산
구조화 단계	HealthOmics annotation, VEP, ClinVar	해석 가능한 필드 생성
테이블 계층	S3 Tables, Glue catalog, Athena	재현 가능한 질의 수행
자연어 계층	Bedrock, Claude, AgentCore	질문을 구조화된 질의와 설명으로 연결

Figure 18-1은 이 전체 계층을 한 장으로 묶어 보여 준다. 아래에서 위로 올라갈수록 데이터는 점점 더 구조화되고, 사용자 인터페이스는 점점 더 자연스러워진다. 그러나 가장 중요한 점은, 자연어 답변이 아무리 매끄러워도 그 아래 calling·annotation·table 계층이 없으면 답의 근거가 사라진다는 사실이다.

Figure 18-1. Bedrock 기반 유전체 질의의 계층 스택

cohort query, pharmacogenomics, variant interpretation 사례

이 구조는 실제 연구 질문에서 더욱 분명해진다. cohort query에서는 BRCA1 pathogenic variant , phenotype group shared pathogenic variant frequency , ancestry group 같은 질문이 대표적이다. pharmacogenomics에서는 같은 질문으로 확장될 수 있다. variant interpretation에서도 ClinVar , , annotation 처럼 사람이 여러 화면과 파일을 오가며

찾던 정보를 더 빠르게 묶어 줄 수 있다. 중요한 점은 이런 답이 자연어로 보이더라도, 실제로는 구조화된 테이블과 규칙 기반 annotation에서 나와야 한다는 사실이다.

또한 Bedrock의 역할은 variant interpretation에만 국한되지 않는다. Sonrai 사례처럼 scRNA-seq cluster annotation과 텍스트 보고서 생성을 보조하는 해석 계층으로도 쓸 수 있다. 그러나 여기서도 메시지는 동일하다. raw omics를 AI가 곧바로 대체하는 것이 아니라, 전처리와 구조화가 끝난 뒤의 interpretation layer를 보조한다는 것이다. 학생에게는 이 공통점을 보게 하는 것이 중요하다. 유전체든 single-cell이든, AI가 강한 지점은 - 이지 원시 데이터 처리 계층이 아니다.

안전성, provenance, human-in-the-loop

변이 해석 장에서 반드시 남겨야 할 문장은 는 것이다. 생성형 AI는 질의 인터페이스와 설명 보조에는 강하지만, 최종 clinical interpretation을 자동화하는 시스템으로 과장하면 안 된다. 특히 병리성 분류, 유전상담, 약물 반응 해석, 환자별 권고 같은 부분은 human review와 governance가 필수다. 자연어 답변이 편리하다고 해서 해석 책임이 자동으로 모델이나 플랫폼으로 옮겨가는 것은 아니다. 오히려 AI가 들어갈수록 provenance와 승인 체계를 더 엄격히 설계해야 한다.

좋은 운영 원칙은 세 가지다. 첫째, 모든 자연어 응답이 어떤 테이블과 어떤 버전의 annotation을 근거로 생성되었는지 남긴다. 둘째, 자유 텍스트 답변보다 구조화된 근거와 함께 답하게 한다. 셋째, 임상적 혹은 연구적 중요 판단은 human-in-the-loop 단계에서 승인하도록 한다. 이렇게 설계된 시스템은 생성형 AI를 무서워할 필요도, 맹신할 필요도 없다. Bedrock 장의 결론은 결국 는 사실이다.

핵심 개념 정리

- Bedrock은 모델 운영 계층이고, 신뢰할 수 있는 답의 기반은 annotation과 구조화된 데이터 계층이다.
- 유전체 자연어 질의 시스템은 `VCF -> annotation -> table -> Athena query -> natural language interface` 구조로 이해하는 것이 가장 정확하다.
- AI는 variant interpretation을 보조할 수 있지만, 최종 clinical judgment를 대체해서는 안 된다.
- provenance와 human-in-the-loop 설계는 생성형 AI를 쓸수록 더 중요해진다.

복습 질문

1. Bedrock과 Claude, AgentCore는 각각 어떤 층의 역할을 하는가?
2. 자연어 질의 시스템이 raw VCF 위에서 직접 답하도록 설계하면 왜 위험한가?
3. variant interpretation에서 human-in-the-loop가 필요한 이유를 세 가지 이상 들어 보라.

Further Reading

- AWS Machine Learning Blog. Accelerating genomics variant interpretation with AWS HealthOmics and Amazon Bedrock AgentCore.
- Anthropic. Claude on Amazon Bedrock.
- AWS. Model parameters for Anthropic Claude models.

References

- Anthropic. 2026a. Claude on Amazon Bedrock.
- Anthropic. 2026b. Claude Code on Amazon Bedrock.
- AWS. 2026a. Model parameters for Anthropic Claude models.
- AWS Machine Learning Blog. 2025. Accelerating genomics variant interpretation with AWS HealthOmics and Amazon Bedrock AgentCore.
- AWS. 2026b. What is AWS HealthOmics?.
- AWS. 2026c. What is Amazon Athena?.

19장. 산업계 사례와 데이터 브로커로 배우는 AWS 오믹스 운영

학습 목표

- 제약사, 바이오텍, 플랫폼 기업이 AWS를 어떻게 다르게 사용하는지 분류할 수 있다.
- Seven Bridges, Basepair, Research Gateway 같은 데이터 브로커 또는 협업 플랫폼의 역할을 설명할 수 있다.
- AWS Marketplace가 무엇이며, SaaS/private offer 방식으로 어떤 유전체 AI 플랫폼을 조달할 수 있는지 설명할 수 있다.
- 연구실이 산업계 사례에서 무엇을 바로 배울 수 있는지 운영 원칙으로 정리할 수 있다.

핵심 질문

- 왜 어떤 조직은 직접 EC2와 Batch를 운영하고, 어떤 조직은 HealthOmics나 외부 플랫폼을 얹는가
- 데이터 브로커는 단순 저장 대행이 아니라 무엇을 대신해 주는가
- AWS Marketplace에서 사는 것과 직접 구축하는 것은 무엇이 다른가
- 산업계 사례에서 반복해서 보이는 공통 아키텍처는 무엇인가

산업계는 무엇을 AWS에 올리는가

산업계 사례를 볼 때 가장 먼저 버려야 할 오해는 “모두가 같은 이유로 AWS를 쓴다”는 생각이다. 제약사, 바이오텍, 진단 기업, 플랫폼 회사는 비슷한 서비스를 쓰더라도 전혀 다른 병목을 해결하려는 경우가 많다. 어떤 조직은 RNA-seq이나 WGS 파이프라인의 throughput을 높이기 위해 클라우드를 택하고, 어떤 조직은 공개 데이터와 임상 메타데이터를 한곳에 모으기 위해, 또 어떤 조직은 협업형 분석 플랫폼을 제공하기 위해 AWS를 선택한다. 즉 산업계의 AWS 사용 사례는 기술 그 자체보다 를 먼저 읽어야 이해가 된다. 이 장에서는 사례 나열보다 운영 모델의 차이를 보는 것이 더 중요하다.

가장 크게 보면 다섯 가지 운영 모델이 보인다. 첫째는 이다. EC2, Batch, FSx, S3를 직접 조합해 파이프라인과 데이터 계층을 운영하는 방식이다. 둘째는 다. HealthOmics처럼 workflow 운영 부담을 줄이는 계층을 얹는다. 셋째는 또는 data broker 모델이다. Basepair, Seven Bridges처럼 데이터는 고객 환경에 두고 orchestration과 UI를 제공하는 경우가 많다. 넷째는 `hybrid cloud`다. 기관 내부 자원, science cloud, AWS를 함께 쓴다. 다섯째는 `marketplace procurement`다. 직접 플랫폼을 만들지 않고 AWS Marketplace를 통해 SaaS나 in-silico lab 플랫폼을 조달하는 방식이다. 이 다섯 가지를 구분하면 산업계 사례가 훨씬 덜 복잡해 보인다.

Takeda, ImmunoScape, Goldfinch로 보는 scale-out 패턴

Takeda 사례는 관리형 서비스가 대규모 오믹스 운영에서 어떤 차이를 만드는지 보여 주는 대표 예다. AWS 소개에 따르면 Takeda는 on-prem Slurm 기반 RNA-seq 운영을 HealthOmics로 옮기면서 20,000 samples를 2일 만에 처리하고, 일일 10,000 samples 규모의 throughput과 70% 이상의 비용 절감 효과를 얻었다고 설명한다. 여기서 중요한 점은 속도 그 자체만이 아니다. immutable run log, scientist self-service, tag 기반 데이터 관리처럼 운영과 재현성의 개선이 함께 언급된다는 사실이 더 중요하다. 즉 클라우드는 단지 더 빨리 도는 서버가 아니라, 더 예측 가능하게 운영되는 분석 시스템을 만들기 위한 수단이 된다.

ImmunoScape 사례는 startup형 패턴을 보여 준다. 이 회사는 Nextflow와 AWS Batch를 중심으로 필요할 때만 확장하는 구조를 택했고, 샘플이 들어올 때만 인프라를 키우는 방식으로 운영했다. 이 구조의 핵심은 `spiky workload`다. 항상 큰 클러스터를 사 두는 것보다, 분석 수요가 몰릴 때만 확장하는 것이 더 합리적인 경우가 많다. Goldfinch와 Loka가 보여 준 구조적 변이 분석 사례도 비슷한 교훈을 준다. 많은 파일과 높은 I/O가 필요한 파이프라인에서 `Cromwell + AWS Batch + FSx for Lustre` 같은 조합은 단순히 S3만으로 운영할 때보다 훨씬 더 적절할 수 있다. 즉 산업계 사례는 서비스 이름보다 를 보여 주는 교육 재료다.

Table 1. 산업계 사례에서 보이는 대표 운영 모델

운영 모델	대표 사례	주된 병목	핵심 AWS 계층
직접 구축	Loka/Goldfinch	파일 수, I/O, 구조적 변이 파이프라인	Batch, FSx, S3
관리형 서비스	Takeda	throughput, 운영 표준화, self-service	HealthOmics, S3

운영 모델	대표 사례	주된 병목	핵심 AWS 계층
startup scale-out	ImmunoScape	spiky workload, 빠른 생산화	Nextflow, Batch, S3
협업 플랫폼	Basepair, Seven Bridges	인프라 진입 장벽, 공유와 provenance	IAM role 연결, managed UI
조달형 플랫폼	Helical	foundation model 기반 in-silico lab	AWS Marketplace, SaaS

Seven Bridges, Basepair, Research Gateway로 보는 data broker 패턴

data broker를 단순히 로 설명하면 본질을 놓치게 된다. 이런 플랫폼의 핵심은 데이터 이동을 최소화하면서, 사용자가 인프라를 직접 엔지니어링하지 않고도 분석과 협업을 수행하게 해 주는 데 있다. Basepair는 고객 계정의 S3나 HealthOmics store에 IAM Role로 연결되는 connected cloud 모델을 강조한다. 즉 데이터는 고객 AWS 계정 안에 남아 있고, SaaS는 orchestration과 visualization 계층만 제공한다. 이 구조는 민감 데이터와 협업 플랫폼을 동시에 다루려는 조직에게 매우 매력적이다.

Seven Bridges는 더 오래된 세대의 대표적 data broker다. AWS 위에서 협업형 분석 플랫폼을 제공하며, CWL portability, NIH 데이터 접근, provenance 추적, GUI 기반 파이프라인 실행을 강하게 내세워 왔다. 여기서 학생이 배워야 할 점은, 모든 연구자가 인프라 엔지니어일 필요는 없다는 사실이다. 플랫폼은 자유도를 조금 줄이는 대신, 재현성과 협업과 접근성을 높이는 경우가 많다. Relevance Lab Research Gateway는 또 다른 변형으로, self-service catalog, shared S3 storage, AWS Batch/ParallelCluster, budget과 cost reporting을 묶어 기관 포털처럼 제공한다. 이 사례는 데이터 브로커가 단순 분석 도구를 넘어 기관 운영 플랫폼이 될 수도 있음을 보여 준다.

AWS Marketplace와 private offer로 조달하는 유전체 AI 플랫폼

AWS Marketplace는 third-party software, data, services를 찾고 사고 배포하고 관리하는 curated catalog다. 유전체 책에서는 이를 단순한 앱 스토어보다 으로 설명하는 편이 더 정확하다. 즉 사용자는 AWS bill과 계약 구조 안에서 외부 솔루션을 도입할 수 있고, 때로는 private offer를 통해 맞춤 가격과 조건을 협상할 수 있다. 학생이 이 구조를 이해하면, 모든 연구 조직이 직접 플랫폼을 만들 필요는 없다는 현실을 자연스럽게 받아들일 수 있다. 연구 인프라는 기술만이 아니라 조달과 계약의 문제이기도 하다.

Helical의 `Bio Foundation Model Platform for In-Silico Labs` listing은 이 조달형 모델을 설명하기 좋은 사례다. listing 기준 이 제품은 SaaS이며, private offer, deployed on AWS, 추가 인프라 비용 가능성 같은 구조를 가진다. 또한 `Model Library`, `Model Personalization`, `In-Silico Experiments` 세 모듈을 통해 single-cell, mRNA, DNA foundation model을 조합해 target identification, biomarker discovery, patient stratification, RNA design을 지원한다고 설명한다. 이 사례를 책에서는 "자동화 실험실"을 물리 실험실의 대체가 아니라 `foundation model + proprietary data adaptation + large-scale virtual experiment orchestration` 계층으로 설명하는 편이 정확하다. 다만 AWS Marketplace listing은 제품 개요와 계약 구조를 보여 주는 자료이지, 성능에 대한 독립 검증 자료가 아니라는 caveat도 반드시 함께 적어야 한다.

공개 데이터 민주화와 hybrid cloud의 공존

산업계 사례를 읽다 보면 한쪽에서는 데이터를 AWS 안에서 바로 읽는 공개 데이터 민주화가 진행되고, 다른 한쪽에서는 여전히 hybrid cloud가 중요한 현실이라는 사실이 동시에 드러난다. Amazon Science의 SRA 기사는 NIH Sequence Read Archive가 AWS Open Data Sponsorship Program을 통해 네이티브하게 접근 가능해졌다고 설명한다. 이 흐름은 방식이 공개 데이터 생태계에서 얼마나 중요해졌는지를 보여 준다. 그러나 모든 조직이 이 모델 하나로 수렴하는 것은 아니다. 법적, 윤리적, 네트워크적 이유로 일부 데이터는 여전히 기관 내부나 다른 science cloud에 남을 수 있다.

Kyoto University의 hybrid cloud 논문은 이런 현실을 잘 보여 준다. 대형 기관에서는 on-prem, 학술용 클라우드, 공용 클라우드 AWS를 함께 묶는 구성이 오히려 더 현실적일 수 있다. 학생은 여기서 "클라우드가 전통 인프라를 완전히 대체한다"는 단선적 서사를 버려야 한다. 실제 연구 인프라는 대개 혼합형이다. 중요한 것은 어느 한 플랫폼을 신앙처럼 택하는 것이 아니라, 데이터 위치를 유지하면서 계산과 워크플로 계층을 적절히 조합하는 능력이다.

연구실이 바로 가져올 수 있는 운영 원칙

산업계 사례를 연구실 수준으로 번역하면 몇 가지 운영 원칙이 남는다. 첫째, 모든 것을 직접 구축하려고 하지 말고, 팀의 역량과 반복 사용 패턴에 맞는 계층을 고른다. 둘째, 데이터는 가능한 한 원래 위치에 두고, orchestration과 해석 계층을 붙이는 방향을 우선 고려한다. 셋째, 플랫폼을 도입할 때는 기술보다 provenance, budget visibility, 협업 방식, access control이 더 중요한 경우가 많다. 넷째, 조달형 플랫폼을 사용할 때는 편의성만 보지 말고, 데이터가 어디에 남는지, 어떤 권한 모델을 쓰는지, 어떤 부분이 독립 검증되지 않았는지를 함께 점검해야 한다.

AWS 유전체 운영의 미래는 한 가지 서비스가 아니라, 데이터 위치를 유지하면서 compute와 workflow 계층, 그리고 필요할 경우 조달 계층까지 조합하는 능력에 있다. 어떤 조직은 HealthOmics로 갈 것이고, 어떤 조직은 Batch와 EKS를 유지할 것이며, 어떤 조직은 Basepair나 Seven Bridges 같은 data broker를 선택할 것이다. 학생은 이 다양성을 보고 혼란스러워할 필요가 없다. 오히려 같은 질문을 던지면 된다. ? 이 질문에 답할 수 있다면 산업계 사례는 단순한 홍보 자료가 아니라 좋은 교과서가 된다.

핵심 개념 정리

- 산업계에서 AWS는 한 가지 방식으로 쓰이지 않으며, 직접 구축, 관리형 서비스, 협업 플랫폼, hybrid cloud, 조달형 플랫폼이 공존한다.
- data broker의 핵심은 저장 대행보다 + + provenance + 에 있다.
- AWS Marketplace는 기술 선택이 아니라 조달 계층으로 이해하는 편이 맞다.
- 산업계 사례를 읽을 때는 서비스 이름보다 을 먼저 보는 습관이 중요하다.

복습 질문

1. Takeda, ImmunoScape, Basepair는 각각 어떤 병목을 풀기 위해 AWS를 사용했는가?
2. data broker 모델은 왜 모든 연구실이 직접 인프라를 운영하지 않아도 되게 만드는가?
3. AWS Marketplace를 단순한 앱 스토어보다 조달 계층으로 설명해야 하는 이유는 무엇인가?

Further Reading

- AWS Industries Blog. How Takeda accelerates large-scale bioinformatics with AWS HealthOmics.
- Seven Bridges. Amazon Web Services page.
- AWS Marketplace. What is AWS Marketplace?.

References

- Amazon Science. 2020. AWS democratizes access to the largest genomic sequences repository — NIH's Sequence Read Archive.
- AWS APN Blog. 2020. Accelerating genomics and high-performance computing on AWS with Relevance Lab Research Gateway Solution.
- AWS Industries Blog. 2024a. How Takeda Pharmaceuticals accelerates large-scale bioinformatics with AWS HealthOmics.
- AWS Industries Blog. 2024b. Biotech startup builds agile drug discovery pipeline with scalable compute.
- AWS Solutions. 2025. Basepair case study.
- AWS Marketplace. 2026a. What is AWS Marketplace?.
- AWS Marketplace. 2026b. SaaS products through AWS Marketplace.
- AWS Marketplace. 2026c. Private offers in AWS Marketplace.
- Helical. 2026. Bio Foundation Model Platform for In-Silico Labs.
- Seven Bridges. 2026. Amazon Web Services page.
- Yokoyama et al. 2023. Hybrid cloud for genome analysis.

20장. 비용이 새는 지점과 해결 전략 – 1PB WGS 운영에서 배우기

학습 목표

- 유전체 워크플로에서 자주 놓치는 숨은 비용 항목을 설명할 수 있다.
- 비용이 커지는 메커니즘을 , , , , 로 나누어 이해할 수 있다.
- AWS의 lifecycle rule, right-sizing, Spot, retention 설정 같은 구체적 대응책을 제시할 수 있다.

핵심 질문

- 왜 "컴퓨트만 많이 썼다"고 생각했는데 실제 청구서는 다른 곳에서 커지는가
- 1PB급 WGS 프로젝트에서 비용은 어느 한 항목이 아니라 어떤 상호작용으로 만들어지는가
- 숨은 비용은 피할 수 없는가, 아니면 설계와 운영으로 줄일 수 있는가

데이터 크기보다 데이터 생애주기가 더 중요하다

대규모 오믹스 프로젝트의 비용을 이해할 때 가장 먼저 버려야 할 생각은 라는 단순한 설명이다. 실제로는 같은 1PB라도 무엇을 얼마나 오래, 어떤 방식으로 읽고, 얼마나 자주 옮기고, 중간 산물을 어떻게 남겨 두느냐에 따라 비용 구조가 완전히 달라진다. 즉 데이터 크기 자체보다 데이터 생애주기(data lifecycle)가 더 중요하다. 학생은 이 장에서 청구서를 보는 법보다, 비용이 만들어지는 메커니즘을 보는 법을 먼저 배워야 한다. 그렇게 해야 숨은 비용이 갑자기 튀는 이유를 이해할 수 있다.

교육용으로 1,000명 WGS, 총 데이터 약 1PB 수준을 생각해 보자. raw FASTQ가 400–600 TB, processed BAM/CRAM이 300–500 TB, intermediate가 300–800 TB, derived data가 10–50 TB 정도라고 가정할 수 있다. 여기서 중요한 것은 최종 보관량이 아니라 workflow 중간의 순간 최대 저장량이다. 실제 운영에서는 intermediate가 원본보다 더 커지는 경우가 흔하며, 순간적으로는 2–3PB에 가까운 저장량이 보일 수도 있다. 따라서 비용 설계는 최종 산물만 보고 세울 수 없고, 워크플로 전체의 생애주기를 함께 봐야 한다.

Incomplete multipart upload와 보이지 않는 유령 바이트

대형 FASTQ, BAM, CRAM 업로드의 기본 방식은 S3 multipart upload다. 이 방식 자체는 효율적이지만, 업로드가 중간에 실패하거나 파이프라인이 멈추면 완성되지 않은 part가 S3에 남을 수 있다. 문제는 사용자가 파일이 "아직 없다"고 느껴도, 저장 비용은 이미 발생하고 있다는 점이다. 이 때문에 incomplete multipart upload는 대규모 오믹스 프로젝트에서 대표적인 이 된다. 파일이 눈에 잘 보이지 않아도 바이트는 이미 남아 있기 때문이다.

AWS는 이를 줄이기 위해 `AbortIncompleteMultipartUpload` lifecycle rule을 제공한다. 교육용으로는 7
같은 정책을 기본값처럼 가르치는 것이 좋다. 완성되지 않은 파일도 비용을 만든다는 점이 핵심이다. 따라서 업로드 실패가 잦거나, 샘플 반입 파이프라인이 길거나, 외부 기관이 데이터를 자주 전달하는 환경일수록 이 규칙은 사실상 필수다. 비용 최적화는 화려한 절약 기술보다 먼저 이런 기본 hygiene에서 시작한다.

실제 현장에서 이 문제가 얼마나 자주 발생하는지는 외부 엔지니어링 문서에서도 확인할 수 있다. AWS Storage Blog의 "Discovering and deleting incomplete multipart uploads to lower Amazon S3 costs"는 List-MultipartUploads API로 잔존 part를 찾아내고 lifecycle로 정리하는 절차를 설명한다. Corey Quinn은 Last Week in AWS에서 "Quieting Noisy Multipart Upload Costs" 같은 글로 이 문제를 "S3 청구서에서 가장 많이 간과되는 항목"이라고 반복해서 지적한다. 또 AWS Config 관리형 규칙 `s3-lifecycle-policy-check`나 `s3-version-lifecycle-policy-check`는 이런 정책이 버킷에 실제로 설정되어 있는지를 계정 차원에서 감시할 수 있게 해 준다. 유전체 연구실이 수십 수백 TB를 다루기 시작하면, 이 작은 규칙 하나가 매달 눈에 띄는 청구서 차이를 만든다.

Small object, random access, request cost

S3 비용은 저장 단가만으로 결정되지 않는다. request cost는 유전체 분석에서 자주 과소평가되지만, 실제로는 매우 중요할 수 있다. BAM과 CRAM의 random access, index 기반 반복 조회, 수십만 개 small object, intermediate 재읽기, console 탐색에 따른 LIST/GET 요청이 겹치면 저장 비용보다 요청 비용이 더 크게 느껴질 수도 있다. 특히

보다 이 비용을 좌우한다는 점이 중요하다. 따라서 데이터 구조를 잘못 설계하면 저장 자체는 싸도 운영 비용은 빠르게 커진다.

이 문제는 클라우드 네이티브라는 이름 아래 small object를 지나치게 많이 만드는 구조에서 더 자주 드러난다. 예를 들어 chunk를 너무 잘게 나눈 배열 데이터, 지나치게 세분화된 intermediate shard, 정리되지 않은 임시 산물은 모두 요청 비용과 listing overhead를 키운다. 해결책은 무조건 하나의 큰 파일로 돌아가는 것이 아니라, `object granularity`를 잡는 것이다. S3를 단순한 하드디스크로 떠올리는 순간 요청 비용은 눈에 띄지 않게 불어난다. 객체 스토리지는 저장 방식만이 아니라 비용 구조도 다르다.

Over-provisioned compute와 right-sizing

compute 비용은 보통 가장 눈에 잘 띄기 때문에, 사람들은 종종 그것만을 문제로 생각한다. 그러나 compute는 동시에 가장 통제하기 쉬운 비용이기도 하다. 낭비의 주된 원인은 , , `Spot` , `shard orchestration overhead`다. 예를 들어 실제 병목이 디스크 I/O인데 CPU 코어만 늘리거나, 메모리 사용량은 낮은데 `r` 계열을 쓰면 비용은 금방 새어 나간다. 즉 compute는 비싸서 문제가 되는 경우보다, 잘못 배치해서 문제가 되는 경우가 더 많다.

해결의 핵심은 right-sizing이다. 처음부터 가장 큰 인스턴스를 고르기보다 `m -> c/r/i/f`처럼 병목에 따라 점진적으로 맞추는 습관이 중요하다. 샘플 단위 병렬화가 가능하면 작은 인스턴스를 많이 쓰는 편이 나을 수 있고, Spot이 잘 맞는 작업이라면 On-Demand만 고집할 이유가 없다. 또한 workflow의 resume, retry, call caching, checkpoint가 잘 설계되어 있으면 더 공격적으로 저렴한 자원을 활용할 수 있다. 학생은 여기서 비용 최적화가 곧 성능 최적화와 다르지 않다는 사실을 배우게 된다.

Intermediate data explosion과 lifecycle 설계

유전체 분석에서 intermediate data는 자주 과소평가된다. sorted BAM, recalibrated BAM, temporary shard, feature matrix, 중간 summary table은 최종 결과보다 더 큰 저장량을 차지할 수 있다. 프로젝트를 시작할 때는 raw FASTQ와 최종 VCF만 떠올리기 쉽지만, 실제 청구서를 키우는 것은 종종 분석 중간의 잠깐 존재하는 대형 파일들이다. 문제는 이 intermediate가 분석이 끝난 뒤에도 아무 규칙 없이 남아 있는 경우가 많다는 점이다. 이때 storage cost는 조용히, 그러나 꾸준히 누적된다.

따라서 lifecycle 설계는 필수다. 어떤 intermediate는 7일 뒤 지워도 되지만, 어떤 것은 재현성을 위해 한 달 보관해야 할 수 있다. 어떤 결과는 바로 Parquet나 요약 테이블로 변환한 뒤 원본 intermediate를 없애도 되지만, 어떤 단계는 재분석을 위해 더 오래 남겨야 할 수 있다. 핵심은 를 미리 정하는 것이다. 생애주기 규칙이 없는 data lake는 쉽게 data swamp가 되고, 비용은 그 늪의 가장 빠른 신호로 나타난다.

Egress, cross-AZ, cross-region data movement

협업 단계에서 가장 위험한 비용 가운데 하나가 데이터 이동이다. 많은 사용자가 같은 AWS 안에서는 이동이 다 무료라고 생각하지만, 실제로는 그렇지 않다. 인터넷으로 나가는 egress는 명확한 비용 항목이고, 같은 리전 안에서도 가용 영역이 다르면 비용이 생길 수 있으며, 리전 간 이동은 더욱 분명한 과금 대상이다. 유전체 데이터는 한 번 움직일 때 TB 단위가 되기 쉽기 때문에, 작은 설계 실수가 곧 큰 청구서로 이어진다. 따라서 compute를 데이터가 있는 곳 가까이에 배치하는 원칙은 성능뿐 아니라 비용 측면에서도 매우 중요하다.

이 문제는 특히 외부 협업에서 자주 드러난다. 데이터를 collaborator에게 넘기거나, 다른 클라우드로 이동하거나, 로컬로 대량 다운로드하는 순간 비용 구조가 바뀐다. 따라서 좋은 운영 원칙은 , 는 것이다. 공개 데이터에서 `bring compute to data`가 중요했던 이유가 여기서 다시 나타난다. 공개 데이터 접근과 자체 데이터 협업은 결국 같은 비용 원리 위에 놓여 있다.

CloudWatch logs, metrics, metadata retention

로그는 작아 보여서 종종 무시되지만, 장기 운영에서는 결코 작은 비용이 아니다. CloudWatch Logs는 기본적으로 로그를 무기한 보관할 수 있으므로, retention을 설정하지 않으면 Batch, EMR, HealthOmics, Lambda, Step Functions에서 나온 로그가 계속 쌓인다. 개별 로그 이벤트는 작더라도, 대량 샘플 처리와 긴 기간이 겹치면 누적 비용은 무시하기 어려워진다. 게다가 Logs Insights로 반복 스캔하는 비용까지 더해지면, “로그는 작으니 괜찮다”는 말은 더 이상 맞지 않는다. 로그도 data lifecycle의 일부로 봐야 한다.

해결은 명확하다. workflow 성격에 따라 7일, 30일, 90일 같은 retention 정책을 명시하고, 장기 보관이 정말 필요한 로그만 따로 남기는 것이다. 학생에게는 이것을 단순 절약 기술보다 로 가르치는 편이 좋다. 좋은 운영은 문제가 생겼을 때 필요한 로그는 충분히 남기되, 의미 없는 로그를 영구 보관하지 않는다. 결국 로그도 intermediate data와 다르지 않다. 무엇을 얼마나 오래 남길지 미리 정하지 않으면, 조용히 비용을 만들어 낸다.

1,000 WGS, ~1PB 환경을 비용 구조로 읽는 법

교육용으로 1,000명 WGS, 약 1PB 규모의 프로젝트를 생각해 보자. 여기서 비용은 하나의 큰 항목이 지배하는 경우보다, storage growth와 request frequency, inefficient compute usage, data movement가 상호작용하며 생겨난다. raw FASTQ를 오래 남기면 storage cost가 생기고, 이를 작은 shard로 계속 읽으면 request cost가 커지고, 분석을 위해 다른 리전의 compute를 쓰면 transfer cost가 붙고, 병목과 맞지 않는 인스턴스를 쓰면 compute cost가 늘어난다. 즉 청구서는 기술 요소의 목록이 아니라, 운영 설계의 결과물이다. 비용은 곧 시스템이 어디에서 비효율적인지를 드러내는 지표다.

이 예제에서 학생에게 남겨야 할 핵심 메시지는 세 가지다. 첫째, `Data size is not the real problem - data lifecycle is.` 둘째, `Compute is controllable - storage and transfer are where costs silently accumulate.` 셋째, `The biggest costs often come from things you did not explicitly plan.` 이 세 문장은 단순한 슬로건이 아니라, 실제 운영 경험을 요약한 원칙이다. 학생이 이 원칙을 이해하면, 아직 자기 프로젝트가 1PB가 아니더라도 큰 규모를 설계할 감각을 미리 배울 수 있다.

비용 누수 진단표와 해결 플레이북

비용 문제를 해결할 때 중요한 것은 공포가 아니라 진단이다. incomplete multipart upload가 보인다면 lifecycle rule부터 확인한다. request cost가 크다면 small object와 random access pattern을 의심한다. compute 비용이 높다면 인스턴스 right-sizing과 Spot 사용 여부를 본다. storage가 급격히 늘어난다면 intermediate retention을 점검한다. data transfer가 크다면 cross-region, cross-AZ, egress 패턴을 추적한다. logs 비용이 계속 늘면 retention 설정이 없는지부터 본다.

숨은 비용은 피할 수 없는 숙명이 아니라, 파악하면 줄일 수 있는 설계 문제다. 유전체 분석은 데이터가 크기 때문에 비용이 생기는 것이 아니라, 큰 데이터를 어떤 습관으로 운영하는가에 따라 비용이 달라진다. 좋은 연구 인프라는 빠르기만 한 인프라가 아니라, 어디서 비용이 만들어지는지를 설명할 수 있는 인프라다. 이 감각이 생기면 청구서는 두려움의 대상이 아니라 설계를 고치는 피드백 도구가 된다.

핵심 개념 정리

- 오믹스 프로젝트의 비용은 데이터 크기 자체보다 데이터 생애주기와 접근 패턴에 더 크게 좌우된다.
- incomplete multipart upload, small object 요청, intermediate data, transfer, 로그는 대표적인 숨은 비용 축이다.
- compute 비용은 눈에 잘 띄지만 동시에 가장 통제하기 쉬운 항목이다.
- 비용 최적화는 절약 기술보다 `lifecycle retention` 에서 출발한다.

복습 질문

1. incomplete multipart upload가 왜 라고 불릴 수 있는가?
2. 저장 비용보다 request 비용이 더 커질 수 있는 유전체 작업의 예를 들어 보라.
3. 1PB급 WGS 환경에서 비용을 줄이기 위한 설계 원칙 세 가지를 제시하라.

Further Reading

- AWS. Aborting incomplete multipart uploads using a bucket lifecycle configuration.
- AWS. Amazon S3 pricing.
- AWS. CloudWatch pricing.

References

- AWS. 2026a. Aborting incomplete multipart uploads using a bucket lifecycle configuration.
- AWS. 2026b. Amazon S3 pricing.
- AWS. 2026c. Amazon EC2 On-Demand pricing.
- AWS. 2026d. Understanding data transfer charges in CUR.
- AWS. 2026e. Working with log groups and log streams.
- AWS. 2026f. CloudWatch pricing.

에필로그. 클라우드 이후의 오믹스 연구

이 책을 통해 우리는 AWS의 개별 서비스보다 더 큰 변화를 보았다. 오믹스 연구는 더 이상 파일을 한 번 내려받아 개인 서버에서 끝내는 작업이 아니라, 데이터가 생성되고, 등록되고, 구조화되고, 질의되고, 해석되고, 공유되는 하나의 플랫폼적 과정이 되었다. 이 변화는 단지 계산 장소가 연구실 밖으로 옮겨갔다는 뜻이 아니다. 데이터가 생성되는 곳, 협업이 이루어지는 방식, 재현성을 확보하는 방법, 그리고 연구자가 코드를 쓰고 질문을 던지는 방식 자체가 함께 바뀌고 있다는 뜻이다. 클라우드는 유전체학을 더 빠르게 만들었을 뿐 아니라, 무엇을 연구 인프라라고 부를 것인가를 다시 정의하게 만들었다.

앞으로의 연구자는 한 가지 역할만으로는 충분하지 않을 가능성이 크다. 데이터 해석 능력만으로도, 인프라 운영 능력만으로도, 생성형 AI 도구 활용 능력만으로도 부족할 수 있다. 대신 좋은 연구자는 데이터의 위치를 이해하고, 분석 계층을 구조화하고, 비용과 재현성을 함께 설계하며, 필요할 때는 AI를 빠른 초안 작성과 질의 인터페이스로 활용할 줄 아는 사람이 될 것이다. 그러나 이 변화가 연구자를 기계적으로 만들 필요는 없다. 오히려 좋은 인프라가 갖추어질수록 연구자는 더 중요한 과학적 질문과 해석에 집중할 수 있다.

이 책의 마지막 메시지는 크고 거창한 것이 아니다. 자신의 연구 주제에 맞는 작은 첫 프로젝트를 하나 설계해 보는 것에서 시작하면 된다. 공개 데이터를 S3에서 직접 읽어 보는 일, 작은 Nextflow 파이프라인을 HealthOmics나 Batch에서 돌려 보는 일, Hail로 cohort 질의를 한 번 해 보는 일, annotation 결과를 Athena로 요약해 보는 일, AI 도구로 작은 분석 스크립트와 문서를 함께 만들어 보는 일 중 하나면 충분하다. 클라우드 이후의 오믹스 연구는 모든 것을 한꺼번에 배우는 사람의 것이 아니라, 데이터와 계산과 해석의 관계를 조금씩 더 잘 설계해 가는 사람의 것이기 때문이다.

10년의 기록

2026년 4월, 덴마크의 Novo Nordisk가 OpenAI와 전사 차원의 전략적 파트너십을 발표했다. 이 협력은 특정 연구팀이 AI 도구를 도입하는 수준이 아니라, 신약 발굴·제조·임상·상업 운영에 이르는 조직 전체를 AI 인프라 위에 재구성하겠다는 선언에 가깝다. 경영진은 이 파트너십의 목적을 "과학자를 대체하는 것이 아니라 supercharge하는 것"이라고 표현했고, 기술 도입과 함께 전사 AI 리터러시 교육까지 계약에 포함되었다 (Reuters 2026). Eli Lilly, Novartis 등 주요 제약사들도 비슷한 방향으로 움직이고 있으며, 이는 바이오 산업이 "AI를 실험적으로 써 보는 단계"에서 "AI를 인프라 자체로 깔아 두는 단계"로 옮겨 가고 있다는 신호다. 이 소식을 접하면서, 나는 AWS 클라우드를 유전체 연구에 써 온 지난 10년을 자연스럽게 돌아보게 되었다.

출발점은 박사후 연구원을 시작한 2015년 말이었다. UCSF 서버에 쌓여 있던 수백 TB의 유전체 데이터를 로컬에서 처리하다 대형 서버가 자주 멈췄고, 결국 James Simons 박사가 설립한 자폐 연구 재단(SFARI)의 지원 아래 AWS로의 이주를 결정했다. 처음에는 SGE cluster와 Intel Lustre를 직접 세우며 유전학을 전공한 연구자가 인프라 엔지니어 역할까지 겸해야 했지만, 시간이 흐르면서 htslib의 클라우드 네이티브 접근이 자리를 잡고, 관리형 서비스들이 하나둘 정리되면서, 연구자는 낯선 엔지니어링 문제 대신 본래의 과학적 질문에 다시 집중할 수 있게 되었다. 공동 연구자가 버킷 접근 권한만 받으면 같은 분석을 그대로 재실행할 수 있게 되면서, 재현성과 국제 협업의 감각도 함께 바뀌었다. 10년이 지난 지금, 그때 우리가 고생하며 쌓던 인프라는 대부분 한두 개의 관리형 서비스로 대체되었고, 산업계는 그 위에 foundation model과 오믹스 데이터를 묶는 다음 단계로 이미 넘어가 있다. 지금의 학생에게 건네고 싶은 말은 단순하다. 10년 뒤에 다루게 될 인프라는 이 책의 내용을 훨씬 넘어설 것이므로, 서비스 이름이 아니라 그 서비스가 풀려고 했던 문제를 읽는 감각을 만들어 두면 된다. 그 감각이 있으면 다음 10년의 바이오 인프라가 어떤 모양이 되든, 그 위에서 자기 연구 질문을 계속 던질 수 있다.

Further Reading

- AWS. Genomics on AWS resources.

- AWS. Genomics Data Transfer, Analytics, and Machine Learning on AWS.

References

- AWS. 2026a. Genomics on AWS resources.
- AWS. 2026b. Genomics Data Transfer, Analytics, and Machine Learning on AWS.
- Reuters. 2026. “Wegovy-maker Novo Nordisk partners with OpenAI to speed drug development.” April 14, 2026.

용어집 (Glossary)

이 용어집은 본문에 처음 등장하는 자리에서 간단히 정의하되, 책 전체에서 반복해서 나오는 핵심 용어를 한곳에 모은 것이다. 클라우드 인프라, 생물정보학, 오믹스 데이터 모델, AI·자동화 용어를 분야별로 정리했다. 각 장을 읽다 모호한 용어가 나오면 여기서 다시 확인하면 된다.

---

1. AWS 인프라 용어

용어	정의
Region	AWS가 서비스하는 물리적 지리 영역. 같은 리전 안에 두면 비용과 지연이 작다.
Availability Zone (AZ)	한 리전 안에서 전력·네트워크가 분리된 데이터센터 묶음.
EC2 (Elastic Compute Cloud)	원하는 크기의 가상 서버를 필요할 때 켜는 계산 서비스.
S3 (Simple Storage Service)	파일이 아닌 객체(object)로 데이터를 저장하는 클라우드 저장소.
EBS (Elastic Block Store)	EC2 인스턴스에 붙는 블록 스토리지. 운영체제 디스크, 임시 작업 공간에 적합.
FSx for Lustre	고성능 POSIX 공유 파일시스템. S3와 연동해 대규모 scratch 용도로 쓴다.
IAM User	사람 혹은 프로그램적 주체에 발급되는 자격 증명 단위.
IAM Role	실행 주체(인스턴스, 서비스 등)에 일시적 권한을 부여하는 방식. 액세스 키를 코드에 박는 대신 쓴다.
VPC	AWS 계정 안의 가상 네트워크. 서브넷, 라우팅, VPC endpoint를 포함한다.
VPC Endpoint	인터넷을 거치지 않고 AWS 서비스(S3 등)에 접근하는 내부 통로.
Spot Instance	여유 자원을 할인가로 쓰는 EC2. 2분 interruption notice 후 회수될 수 있다.
Batch	대규모 잡을 큐에 넣고 EC2/Fargate에서 실행시키는 관리형 서비스.
Lambda	이벤트에 반응해 짧게 돌아가는 서버리스 함수 실행 환경.
EMR	Spark, Hadoop 같은 분산 프레임워크를 AWS에서 관리형으로 돌리는 플랫폼.
HealthOmics	유전체·오믹스 전용 storage, workflow, analytics를 묶은 AWS 서비스.
SageMaker (SageMaker AI)	머신러닝 모델 학습과 배포를 위한 통합 서비스. 2024년 12월 `SageMaker AI`로 개명.
Bedrock	foundation model(Claude 등)을 AWS 계정 안에서 호출·관리하는 운영 계층.
AgentCore	Bedrock 위에서 도구 사용, 질의 orchestration을 담당하는 응용 계층.

용어	정의
Athena	S3 위의 데이터를 SQL로 질의하는 서버리스 쿼리 서비스.
Glue (Data Catalog)	S3 위 데이터의 스키마와 테이블 메타데이터를 관리하는 카탈로그.
DataSync	온프레미스, 다른 클라우드, 시퀀싱 센터에서 AWS로 데이터를 옮기는 관리형 서비스.
ECR	컨테이너 이미지를 저장·배포하는 AWS 레지스트리.
ECS / Fargate	컨테이너 실행 서비스. Fargate는 인스턴스를 직접 관리하지 않는 형태.
EKS	AWS에서 관리형으로 돌리는 Kubernetes.
ParallelCluster	Slurm/HPC 스타일 클러스터를 AWS에서 간편하게 띄우는 도구.
Step Functions	여러 서비스의 실행 흐름을 상태 기계(state machine)로 엮는 오케스트레이션.
MWAA	AWS가 관리하는 Apache Airflow 환경.
CloudWatch	로그, 메트릭, 이벤트를 수집·조회하는 관측 서비스.
S3 Tables	트랜잭션·스키마가 관리되는 S3 기반 Iceberg 테이블 스토리지.
Quick (Amazon Q / QuickSight)	자연어·시각화 기반 분석 인터페이스 계열.
Mountpoint for Amazon S3	AWS 공식 S3 FUSE 마운트 도구. 읽기 전용과 append 위주.
S3 Intelligent-Tiering	접근 빈도에 따라 S3 스토리지 계층을 자동 조정하는 저장 클래스.
S3 Glacier	장기 보관용 저비용 S3 스토리지 계층.
Lifecycle rule	S3 객체의 전이·삭제 규칙. incomplete multipart upload 정리에 중요.

2. 유전체·변이 분석 용어

용어	정의
WGS (Whole-Genome Sequencing)	한 사람의 전체 유전체를 읽는 시퀀싱. 한 명당 수십~수백 GB 규모.
WES (Whole-Exome Sequencing)	단백질 코딩 영역을 집중해 읽는 시퀀싱.
FASTQ	시퀀싱 원시 리드와 품질값을 담은 텍스트 포맷.
BAM / CRAM	정렬된 리드를 담는 이진 포맷. CRAM은 참조 기반 압축으로 더 작다.
VCF	변이(variant)를 기록하는 표준 텍스트 포맷.
BGZF	BAM/CRAM/VCF에 쓰이는 블록 기반 gzip 변형. 랜덤 접근을 가능하게 한다.
Variant calling	정렬 결과에서 변이를 호출하는 단계.
Joint genotyping	여러 샘플의 중간 산물(g.vcf 등)을 합쳐 cohort 수준에서 변이를 재호출하는 단계.
VEP (Variant Effect Predictor)	변이의 기능적 효과를 예측하는 Ensembl 도구.
ClinVar	변이의 임상적 의미를 공유하는 NIH 공개 데이터베이스.
Pathogenic variant	질병과 인과적 연관이 확립된 변이.
Allele frequency (AF) / Allele number (AN)	코호트에서 대립유전자가 관찰된 비율과 전체 수.
Multiallelic site	한 위치에 여러 대립유전자가 있는 경우.
Burden test	유전자 단위로 rare variant의 누적 효과를 검정하는 방법.

용어	정의
gnomAD	Broad Institute가 관리하는 대규모 인구 변이 빈도 참조 데이터.
SRA (Sequence Read Archive)	NCBI가 관리하는 공개 시퀀싱 원시 데이터 저장소.
DRAGEN	Illumina의 FPGA 가속 유전체 분석 파이프라인.
Manifest	분석에 들어가는 입력 파일의 경로·메타데이터 목록 파일.

3. 분산 분석·데이터 모델 용어

용어	정의
Spark	분산 데이터 처리 프레임워크. EMR의 주된 실행 엔진.
Parquet	칼럼 기반 압축 저장 포맷. 대규모 질의에 유리.
Iceberg	S3 등 객체 스토리지 위에서 트랜잭션과 스키마 진화를 지원하는 테이블 포맷.
EMRFS	EMR이 S3를 HDFS처럼 쓰게 해 주는 파일시스템 계층.
Hail	Spark 위에서 유전체 데이터를 다루는 분산 분석 라이브러리.
MatrixTable	Hail의 행=변이, 열=샘플 행렬 자료구조.
VariantDataset (VDS)	reference block과 variant data를 분리한 sparse, split 표현. cohort-scale 분석 표현.
Local alleles (LGT, LAD, LPL, LA)	multiallelic site에서 전역 인덱스 대신 지역 인덱스를 쓰는 VDS 표현.
Nextflow / WDL / CWL	재현 가능한 워크플로 정의 언어. HealthOmics가 지원.

4. single-cell·spatial·이미징 오믹스 용어

용어	정의
scRNA-seq	세포 단위 RNA 발현 측정. 한 실험에서 수만~수백만 세포가 나온다.
Spatial transcriptomics	조직 절편의 위치 정보와 함께 RNA 발현을 측정하는 기술.
AnnData	세포×유전자 행렬과 `obs`, `var`, `obsm`, `layers`를 함께 다루는 데이터 모델.
h5ad	AnnData의 HDF5 기반 저장 backend. 단일 파일.
Zarr	N차원 배열을 chunk 단위로 저장해 객체 스토리지에서 부분 접근을 가능하게 하는 포맷.
Chunk	Zarr 배열의 최소 접근 단위. 크기 설계가 성능과 비용을 좌우한다.
Sharding (zarr v3)	작은 chunk를 묶어 객체 수를 줄이는 기법.
OME-Zarr	생물학 이미지와 다중 해상도 피라미드를 위한 Zarr 기반 표준.
SpatialData	Images, Labels, Points, Shapes, Tables를 한 좌표계에서 다루는 프레임워크.
TileDB-SOMA	larger-than-memory 질의와 corpus-wide API를 제공하는 플랫폼형 single-cell 저장 계층.
Vitessce	웹 기반 single-cell·spatial 시각화 컴포넌트 모음.

용어	정의
CELLxGENE Census	CZI가 제공하는 corpus-wide single-cell 질의 플랫폼. TileDB-SOMA 기반.
gimVI	짝지어지지 않은 scRNA-seq와 spatial 데이터를 통합해 미측정 유전자를 추정하는 모델.

5. AI·자동화 용어

용어	정의
Foundation model	범용 대규모 언어·멀티모달 모델. Claude, Llama 등이 포함된다.
Claude	Anthropic의 foundation model 계열. Bedrock을 통해 AWS에서 호출 가능.
Kiro	요구사항·설계·작업 파일을 명시적으로 남기는 spec-driven AI IDE.
RAG (Retrieval-Augmented Generation)	외부 데이터에서 관련 문서를 먼저 검색해 모델 답변을 근거 있게 만드는 방식.
Human-in-the-loop	AI 출력에 사람의 검토·승인을 결합해 책임 체계를 유지하는 설계.
Provenance	데이터가 어디서 왔고 어떤 처리를 거쳤는지 추적 가능한 기록.
Reproducibility	같은 입력·코드·환경에서 같은 결과가 나오도록 보장하는 성질.

6. 운영·비용 용어

용어	정의
Bring compute to data	데이터를 옮기지 않고 데이터가 있는 곳에서 계산을 돌리는 원칙.
Egress	리전·AZ를 벗어나는 데이터 전송. 상당한 비용 요인.
Cross-AZ / cross-region	가용 영역·리전 사이의 데이터 이동. 구성에 따라 과금된다.
Array job	Batch가 같은 정의를 N개 복제해 병렬 실행하는 잡.
Shard	큰 작업을 재실행 단위로 나누기 쉽게 쪼갠 묶음.
Retry / timeout	일시적 실패를 자동 복구하거나 무한 대기를 방지하는 설정.
Incomplete multipart upload	중단된 업로드가 남긴 "유령 바이트". 7일 lifecycle rule로 정리한다.
Right-sizing	병목에 맞는 인스턴스 패밀리·크기를 고르는 습관.
Presigned URL	제한된 시간 동안 유효한 서명 URL. 외부 공유에 쓴다.
Requester pays	접근 비용을 요청자(다운로드자)가 부담하는 S3 설정.

찾는 용어가 없다면

본문의 첫 등장 시 정의를 찾아보거나, 각 장 말미의 `Further Reading`과 서비스 문서 링크가 References에 정리되어 있다. AWS 공식 용어는 해당