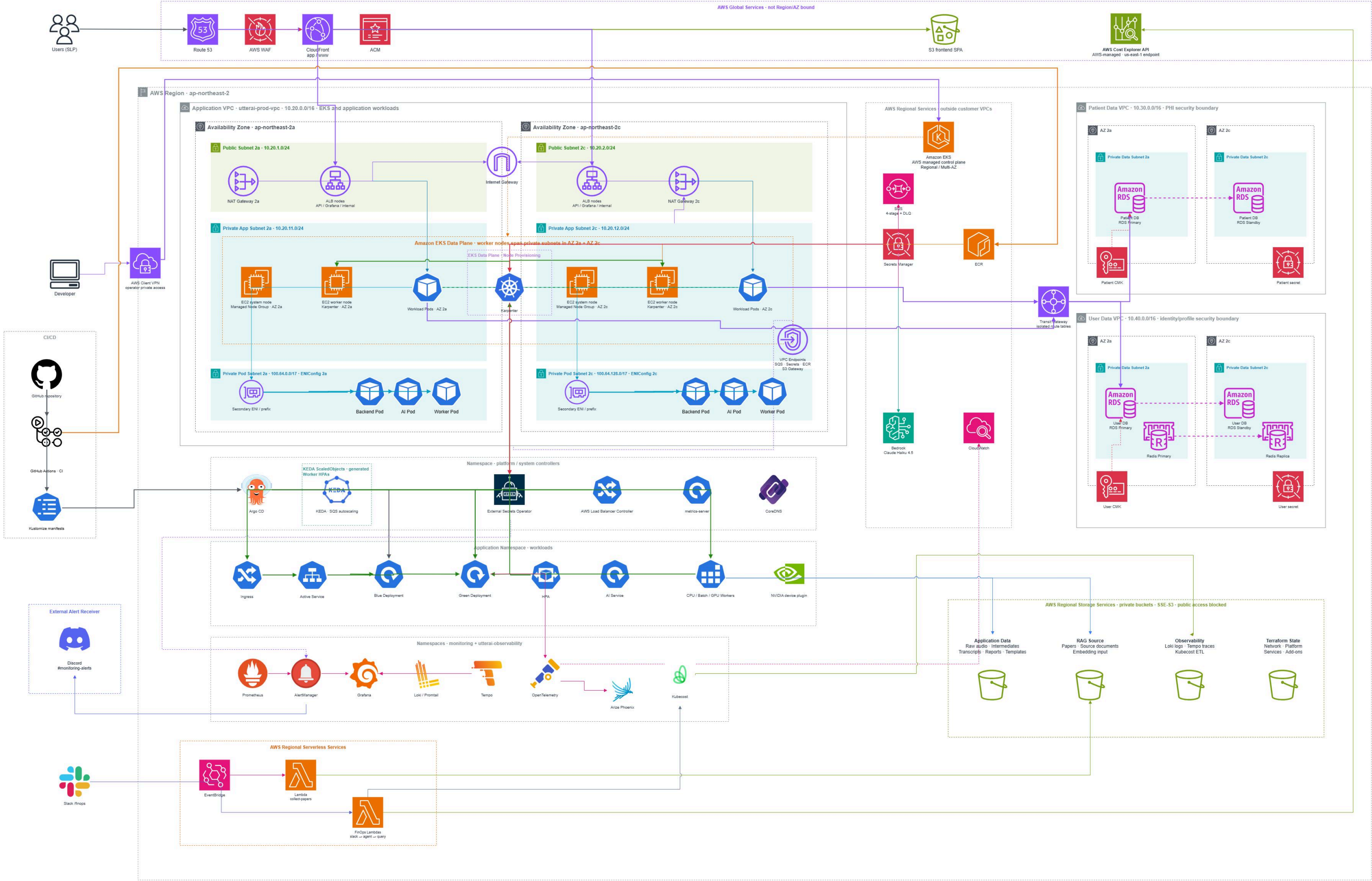


AWS 네트워크부터 EKS 워크로드 · SQS 파이프라인 · GitOps · 관측성까지 전체 운영 흐름을 하나의 아키텍처로 연결했습니다.



아키텍처 구성 요약

각 구역은 뒤 페이지에서 자세히 다룹니다.

1 글로벌 진입 & 인증

Route 53 → WAF → CloudFront가 정적 화면(S3 Frontend)과 API 요청(ALB)을 나누고, Cognito JWT로 사용자 인증합니다.

2 네트워크 격리

영상 음성(PHI) · 계정 · 서비스 영역을 서로 다른 VPC로 나누고, Aurora · Redis는 외부 경로가 없는 Private Data 서브넷에 둡니다.

상세 → Data Security

3 EKS 컴퓨트

System · API · CPU Worker · GPU Worker NodeGroup을 분리하고, Karpenter가 대기 중인 Pod를 보고 노드를 프로비저닝합니다.

4 비동기 분석 & 자동 확장

API가 Job을 SQS Analysis Queue에 넣으면 Worker가 ASR · 화자 분리 · RAG 검색 · Bedrock 리포트 생성을 처리하고, KEDA가 큐 길이 기준으로 Worker를 0에서 확장합니다.

상세 → Autoscaling

5 데이터 & 보안

Aurora PostgreSQL(Writer + Read Replica) · ElastiCache Redis · S3 단계별 버킷에 데이터를 두고, KMS 암호화와 서비스별 IRSA · S3 정책으로 도메인마다 접근을 제한합니다.

6 배포 & 관측

GitHub Actions가 이미지를 빌드해 ECR에 올리면 Argo CD가 GitOps로 배포하고, OpenTelemetry → Prometheus → Grafana로 노드 · 큐 · 워커를 관측합니다.

상세 → CI/CD · Monitoring · Cost

상세 → Data Security

임상 음성·전사본(PHI)과 계정·프로필 정보를 같은 신뢰 경계에 두지 않고, 데이터 도메인과 접근 경로를 분리해야 했습니다.

2026.05 — 2026.07



구성

- Patient Data VPC와 User Data VPC를 Application VPC와 분리하고, Transit Gateway isolated route table에는 필요한 방향만 명시적으로 연결했습니다.
- EKS Private App subnet에는 S3 Gateway VPC Endpoint를 연결해 각 버킷 접근을 인터넷이나 NAT 경유 없이 처리했습니다.
- 각 데이터 계층은 Private subnet에 두고 외부에서 직접 접근하는 public 경로를 차단했습니다.

핵심 통제

- VPC 분리만으로는 Pod가 잘못된 S3 bucket이나 Secret·DB에 접근하는 것을 막지 못했습니다. 특히 음성 원본과 전사·리포트 파일이 같은 권한으로 처리되면 데이터 도메인 분리의 의미가 약해질 수 있었습니다.
- S3 bucket policy와 VPC Endpoint policy를 함께 적용해 특정 VPC·role·prefix에서만 접근하도록 이중으로 제한했습니다.

어려웠던 점

네트워크·전송 경로 분리

문제 : Patient·User Data VPC를 Application VPC와 분리해도 VPC 간 통신 방향과 S3 접근 경로가 잘못 연결되면 Private 데이터 계층이 무너질 수 있었습니다. TGW route와 Endpoint 경로가 섞이면 우회 경로를 놓칠 위험이 있었습니다.

해결 : Transit Gateway isolated route table로 필요한 방향만 연결하고, EKS Private App subnet에는 S3 Gateway Endpoint만 연결해 인터넷·NAT 경로를 제거했습니다. Route table·Endpoint 정책을 실제 호출 흐름 기준으로 점검했습니다.

결과 : 외부 public 경로 없이 데이터 VPC와 Application VPC의 경계를 유지하고, S3 접근 경로를 Private 영역으로 고정했습니다.

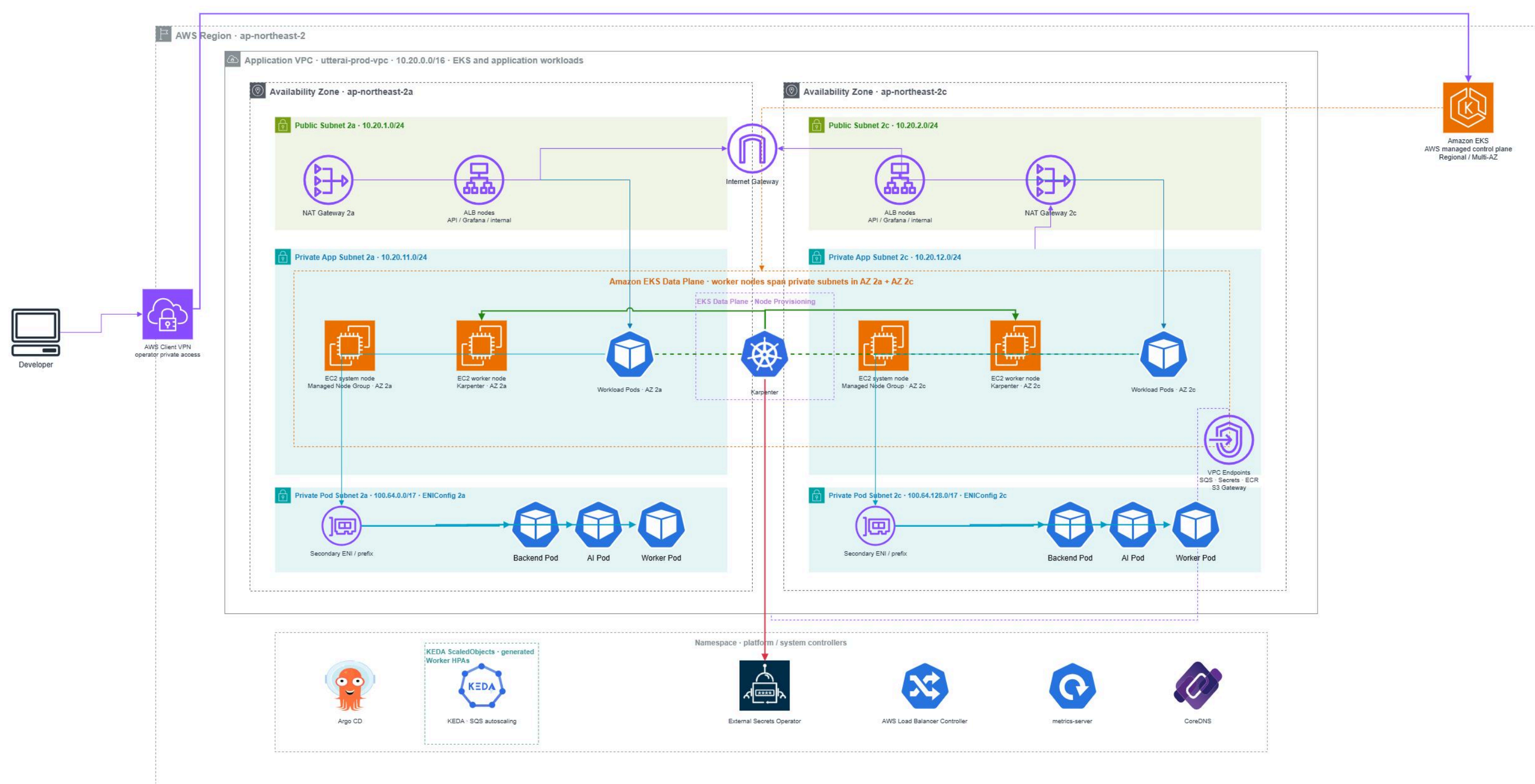
저장소·워크로드 권한 최소화

문제 : 음성 원본·전사·리포트·RAG 데이터에 같은 권한을 쓰면 하나의 IAM 오류가 전체 데이터 노출로 이어질 수 있었습니다. Worker가 늘수록 공통 role로 묶고 싶은 유혹도 커졌습니다.

해결 : S3 Block Public Access·SSE-S3(AES256)·Lifecycle을 적용하고, 서비스별 IRSA와 Secrets Manager → ESO, prod default-deny NetworkPolicy를 구성했습니다. role matrix와 bucket/object 정책을 대조했습니다.

결과 : 서비스별 bucket/object Get·Put 권한만 허용하고 Pod의 장기 AWS access key를 제거해 데이터 접근 범위를 최소화했습니다.

2개 AZ의 EKS에서 system 애드온과 애플리케이션 워크로드를 분리하고, VPN·IRSA·ESO로 운영 경계를 구성했습니다.



EKS 클러스터 구축

- 각 NodePool에 taint·label을 적용해 Pod의 requests와 workload 요구사항에 맞는 노드를 선택하도록 했습니다.
- 큐가 늘어 Pending Pod가 생기면 Karpenter가 NodeClaim을 생성하고, 큐가 비면 consolidation으로 유휴 노드를 회수하도록 구성했습니다.
- 외부 트래픽과 개발자 관리 트래픽을 분리 후 VPN을 통과한 운영자만 Private 영역의 클러스터·데이터 계층에 접근하도록 했습니다.

어려웠던 점

네트워킹 (VPC CNI)

증상 : CPU 워커 노드에서 16시간 동안 FailedCreatePodSandBox가 4,569회 발생. 서브넷에 여유 IP가 112개 남았는데도 할당 실패했습니다.

원인 : ENABLE_PREFIX_DELEGATION=true로 IP를 /28(16개 연속) 블록 단위로 할당하는데, VPC Endpoint ENI와 노드 warm pool이 IP 공간을 단편화시켜 연속 블록을 못 찾았습니다.

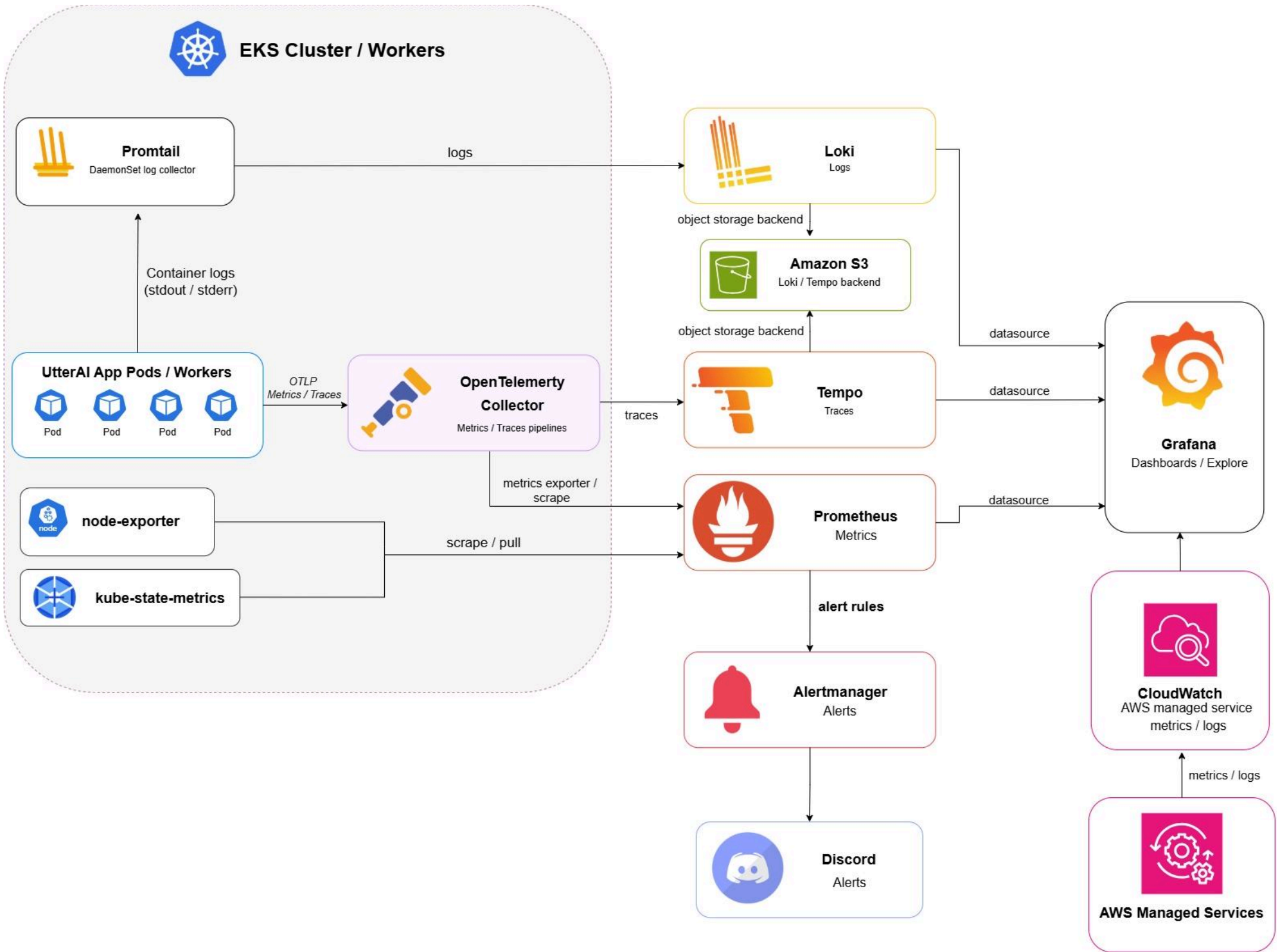
해결 : Secondary CIDR(100.64.0.0/16) + AZ별 /17 서브넷(AZ당 32,766 usable IP) + Custom Networking(ENIConfig) 전환으로 IP 고갈을 해소했습니다.

오토스케일링 (CA + HPA의 구조적 한계)

증상 : SQS 큐에 100개가 쌓여도 워커 Pod가 즉시 늘지 않고 전체 스케일아웃까지 5~10분 소요. GPU 워커가 최소 1개 상시 기동돼 비용 문제도 있었습니다.

원인 : HPA가 CPU 사용률(70%)로만 작동해 SQS 큐 깊이를 못 봤고, 노드 증설도 CA→ASG desired→Launch template→kubelet 기동의 간접 경로라 2~4분 추가. CA+HPA는 minReplicaCount:0 미지원이었습니다.

해결 : KEDA(SQS 큐 깊이 30초 주기 직접 폴링) + Karpenter(Pending Pod 감지 즉시 EC2 프로비저닝)로 교체. 전체 스케일아웃 48초로 단축, minReplicaCount:0으로 유휴 시 워커를 0까지 축소해 비용을 절감했습니다.



Monitoring 구축과 검증

1 로그 수집

컨테이너 stdout/stderr 로그를 각 노드의 Promtail DaemonSet이 수집해 Loki로 보냅니다.

앱은 저장 방식을 몰라도 되고, CloudWatch Logs 중복 적재 없이 Loki로 단일화합니다.

2 지표·트레이스 수집

App Pod·Worker가 OTLP로 지표·트레이스를 OpenTelemetry Collector에 보내면 지표는 Prometheus, 트레이스는 Tempo로 분기합니다.

Collector 파이프라인은 memory_limiter·batch와 redact processor를 거칩니다.

3 클러스터 지표 수집

node-exporter는 노드 리소스를, kube-state-metrics는 오브젝트 상태를 노출하고 Prometheus가 주기적으로 scrape/pull 합니다.

OTel Collector의 Prometheus exporter도 같은 방식으로 수집됩니다.

4 저장

지표는 Prometheus TSDB, 로그는 Loki, 트레이스는 Tempo에 저장하고, Loki·Tempo는 object storage backend로 Amazon S3를 씁니다.

로컬 PVC 없이 S3 직접 저장. 보존 Loki 7·14일 / Tempo 1·3일 (dev·prod).

5 알림

Prometheus alert rule이 이상을 감지하면 Alertmanager가 묶고 중복을 제거해 Discord로 전달합니다.

감지 — Pending Pod·CrashLoopBackOff·재시작·오류율·p95·수집 실패(up==0).

6 조회 & 관리형

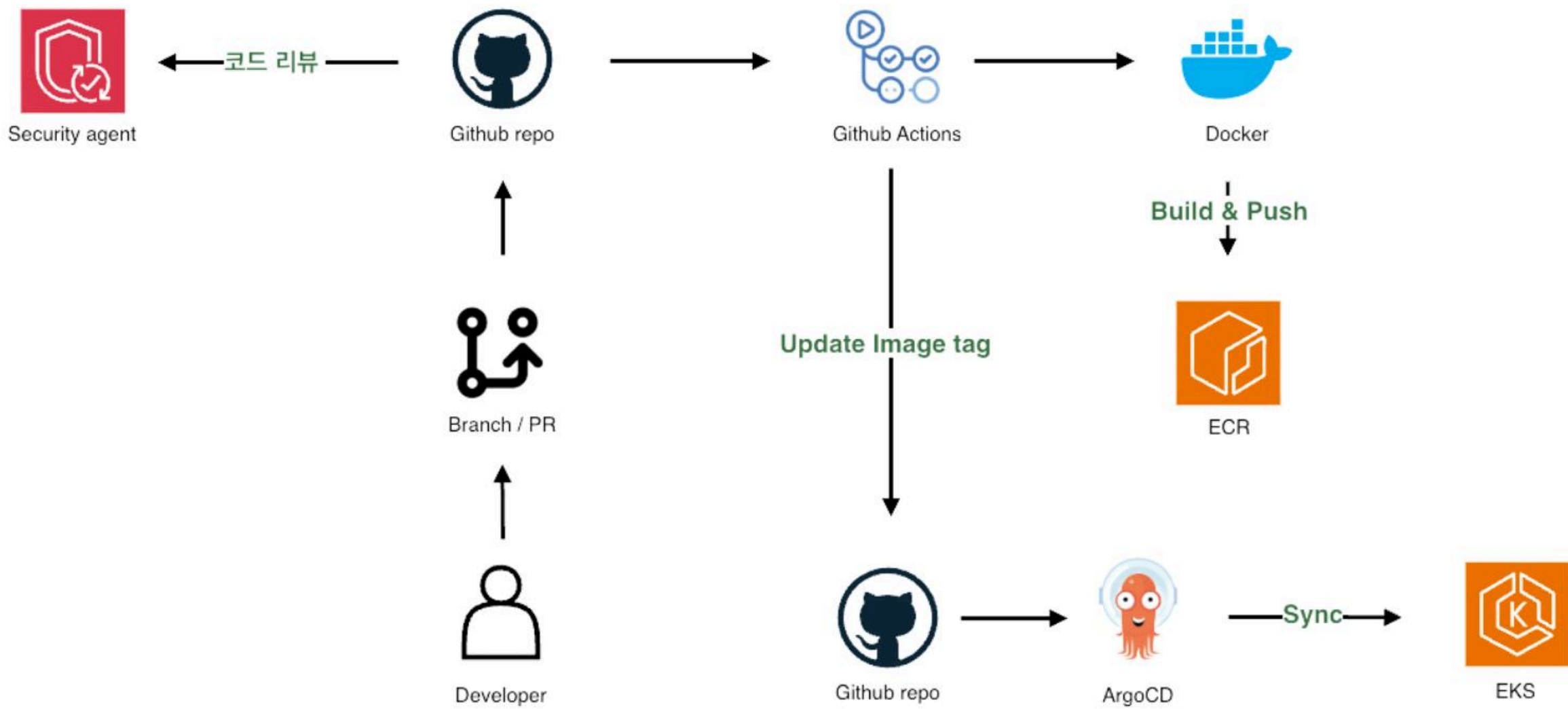
Grafana가 Prometheus·Loki·Tempo를 datasource로 등록해 지표·로그·트레이스를 한 시간축에서 대조합니다. 관리형 서비스 지표는 CloudWatch로 봅니다.

Grafana는 외부 미노출 — port-forward(dev) / VPN·SSO(prod).

7 민감정보 Redaction

임상 음성 SaaS라 관측 데이터에 PHI가 남지 않게 파이프라인에서 제거합니다. 로그는 Promtail이 토큰·비밀번호·presigned URL을 정규식으로 가리고, 트레이스는 OpenTelemetry Collector가 URL·S3 key·audio key attribute를 삭제합니다. 지표 label에는 낮은 cardinality 값만 허용합니다.

관측 스택 Secret(Grafana·Discord webhook·Loki·Tempo IRSA)은 전부 AWS Secrets Manager → External Secrets Operator로 주입합니다.



CI/CD 아키텍처와 배포 흐름

CI · 소스 → 이미지

1 Branch / PR

Backend·AI·Infra 저장소를 나눠 독립 배포하고, feature 브랜치는 CI 검증만 수행합니다.

2 PR CI

언어별 테스트와 Docker 빌드 가능 여부를 검증하며, 이 단계에서 EKS 반영은 하지 않습니다.

3 이미지 빌드 · ECR

OIDC로 AWS 권한을 받아 빌드한 뒤, 환경+커밋 태그를 붙여 이미지 저장소에 올립니다.

4 변경 전파

빌드가 끝나면 새 이미지 정보를 인프라 저장소로 넘겨 배포 워크플로를 트리거합니다.

CD · GitOps → EKS

5 GitOps PR

매니페스트의 이미지 태그를 바꾸는 PR을 만들고 다시 검증하며, merge가 배포 승인입니다.

6 Argo CD 동기화

인프라 저장소의 main 브랜치만 감시해 클러스터에 반영하고, 운영은 수동 승인입니다.

7 EKS 배포

dev는 롤링 업데이트, 운영 백엔드는 블루-그린으로 무중단 전환합니다.

● 환경 · 배포 방식

dev → 롤링 / 운영 → 블루-그린·일반 배포 / feature → 검증만 수행합니다.

실패 지점 · 책임 경계

· Docker 빌드 실패

→ ECR 푸시와 이후 배포 이벤트가 발생하지 않음

· AWS 권한 획득 실패

→ OIDC 신뢰 정책과 role ARN 확인 필요

· 변경 전파 실패

→ 이미지는 있으나 GitOps PR이 생성되지 않음

· 매니페스트 렌더 실패

→ GitOps PR 생성 또는 PR 검증 단계에서 실패

· Argo 동기화 불일치

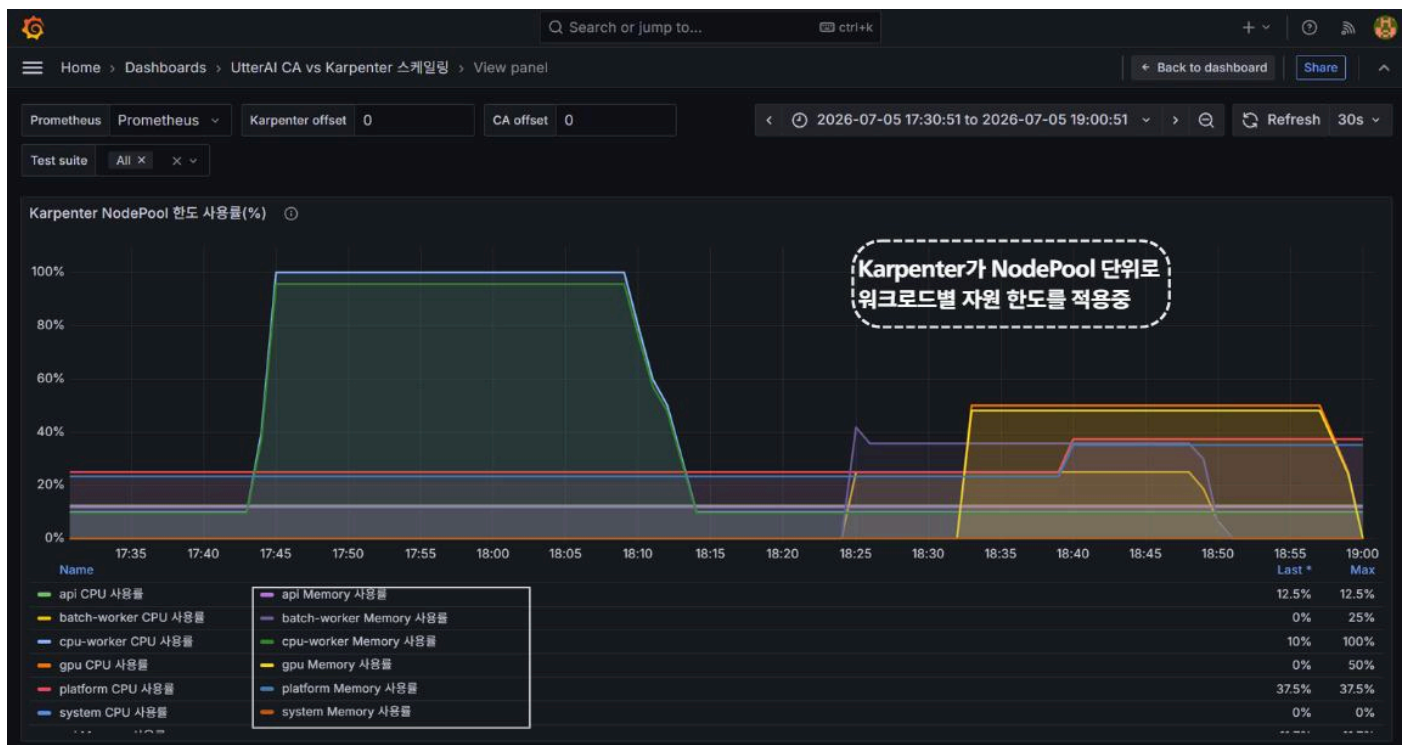
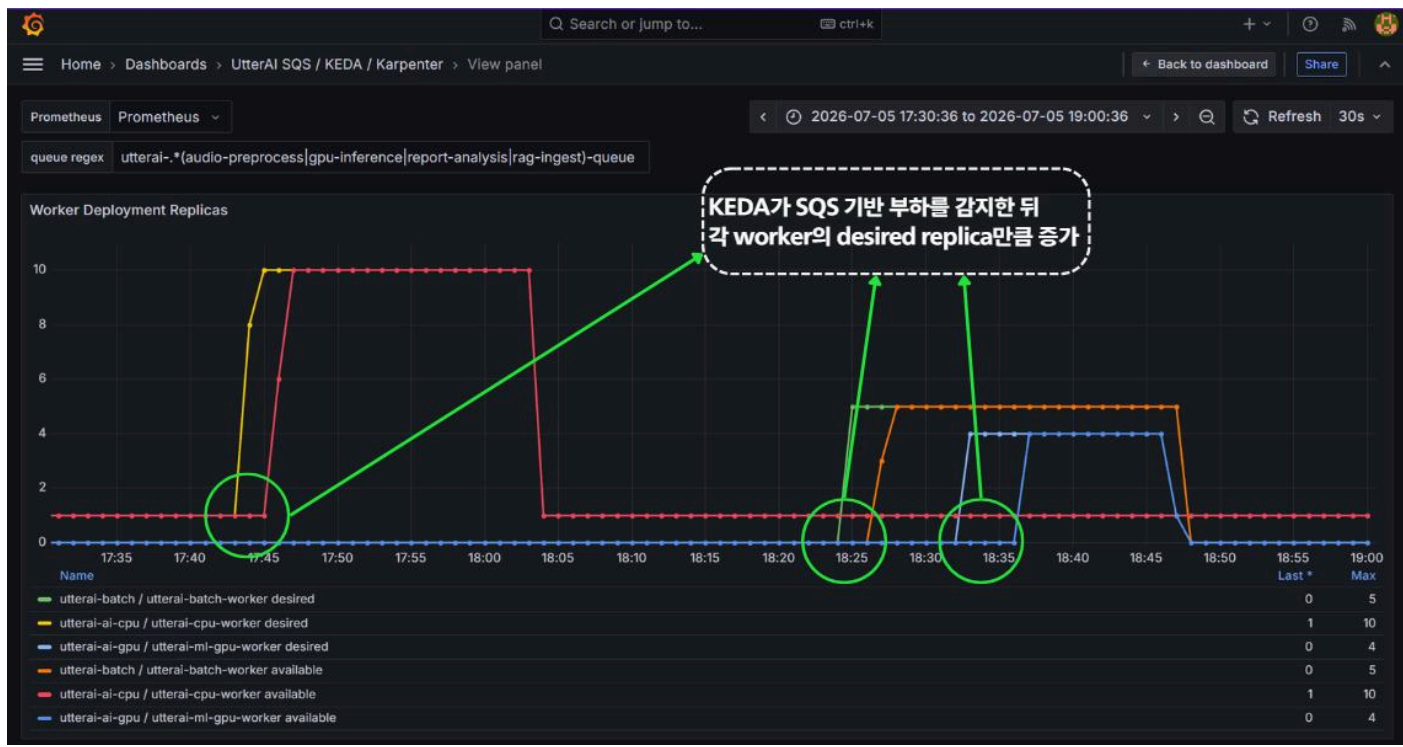
→ dev 자동 반영 / 운영은 담당자가 수동 동기화

· Pod 배포 실패

→ 이미지 pull·Secret·초기화 컨테이너·probe 확인

별도 보안 스캔 단계는 두지 않고, PR 필수 체크와 ECR 푸시 스캔으로 대체합니다.

SQS queue depth → KEDA replica → Pending Pod → Karpenter NodeClaim → idle scale-to-zero 복귀 검증



검증 목적 또는 가설

트래픽 증가 시 SQS queue depth를 기준으로 KEDA가 Worker Replica를 확장하고, Pending Pod 발생 시 Karpenter가 노드를 프로비저닝하는지 확인했습니다.

검증 환경과 시나리오

부하를 순차적으로 주입해 정상 처리 → scale-out → 노드 프로비저닝 → 유휴 상태 scale-to-zero 복귀까지 하나의 시나리오로 관찰했습니다.

정량 검증 결과

검증 포인트

- queueLength trigger와 desired/available replica, Pending→Running, created/terminated activity가 같은 시간축에서 순서대로 맞물리는지 확인했습니다.

결과

- CPU worker는 queueLength=5, min 1/max 10으로 동작하고, GPU worker는 queueLength=1·min 0/max 4, Batch worker는 queueLength=3·min 0/max 5로 동작하도록 확인했습니다.
- Grafana에서 18:25 전후 batch-worker desired 증가와 created activity가 함께 발생했고, 이후 18:42~18:45 구간에도 created/terminated activity와 Pending spike가 관찰됐습니다. scale-out 뒤 Pending이 0으로 내려가고 Running 상태로 전환되는 흐름을 확인했습니다.
- 큐가 비면 GPU·Batch는 desired=0으로 복귀하고, cooldown 이후 Karpenter가 유휴 NodeClaim을 회수합니다. CPU worker는 min=1을 유지해 다음 요청을 받을 기본 처리 슬롯을 남깁니다.

EC2 Compute 일비용

-58.1%

\$28.73 → \$12.02 / day

2일 EC2 실측 비용

-33.8%

\$88.72 → \$58.73 (-\$29.99)

g4dn (GPU) 비용

-60.1%

\$45.20 → \$18.04

m5 (CPU) 비용

-93.7%

\$10.76 → \$0.67

인스턴스 사용시간은 오히려 +47% (449.15h → 660.95h) 늘었지만 EC2 비용은 감소 — 사용량이 준 게 아니라 같은 양을 더 싸게 처리한 결과입니다.

비용 최적화 상세

1. Total EC2 Instance Cost

Period	Dates	Total Cost	Daily Average
Before stabilization	Jun 18–19	\$88.7215	\$44.3607
After stabilization	Jun 21–22	\$58.7324	\$29.3662
Delta		-\$29.9891	-\$14.9945
Change		-33.8%	-33.8%

안정화 전(6/18–19) vs 후(6/21–22) 2일씩 EC2 실측 비용

1 실제 절감 메커니즘

스케일 기준을 CPU 사용률 70%에서 SQS 큐 깊이로 바꿔, 부하가 큐에만 쌓이는 구간에서도 워커를 즉시 확장합니다.

Karpenter는 Spot 인스턴스를 우선 프로비저닝하고, 워크로드별로 instance family를 나눕니다.

GPU 워커는 0→4로 확장하고 backlog가 없으면 0으로 회수합니다.

consolidateAfter — GPU 10분 / CPU 워커 5분 / batch·API 30초. Spot 중단 시 SQS visibility timeout 으로 자동 재처리.

AWS FinOps AgentPublic Preview

+ New chat

Tasks

Automations

Artifacts

Context files

Recent

AWS Cost Comparison

AWS Cost ComparisonShow ID

모두 5:20

Show thinking process (3 steps)

Here's the day-to-day comparison. All figures are UnblendedCost (USD), credits/refunds excluded. J

Jun 14 vs Jul 01 — Daily Cost Comparison

Service	Jun 14	Jul 01	Delta	Change
EC2 Compute	\$28.73	\$12.02	-\$16.71	-58.1%
ElastiCache	\$0.58	\$11.88	+\$11.31	+1,967.2%
RDS	\$2.78	\$6.91	+\$4.13	+148.5%
EC2-Other (EBS/NAT)	\$2.20	\$6.18	+\$3.98	+180.9%
VPC (NAT GW)	\$2.86	\$6.24	+\$3.38	+118.2%
EKS	\$14.40	\$17.64	+\$3.24	+22.5%
Claude Haiku (Bedrock)	\$0.01	\$0.58	+\$0.57	+5,700.0%
Route 53	\$0.00	\$0.50	+\$0.50	ne

AWS Cost ComparisonShow ID

Calculated from API data above.

Key observations

EC2 Compute dropped sharply (-\$16.71). This is the biggest mover. Jun 14 had heavy on-demand usage (g4dn.xlarge, m5.xlarge at ~\$28.73/day). By Jul 1, Karpenter-only is running a leaner on-demand fleet, with more Spot instances absorbing the workload at lower cost.

2 검증 방식

절감 효과는 안정화 전(6/18–19)과 후(6/21–22) 각 2일 구간을, AWS FinOps Agent에 자연어로 질의해 Cost Explorer 청구 데이터를 비교하는 방식으로 확인했습니다. 전체 EC2뿐 아니라 g4dn(GPU)·m5(CPU) 등 instance family별 비용을 따로 추적했습니다.

Kubecost의 "monthly savings available" 추정치는 적용 전 추천 상한이라 참고만 하고, 실제 절감은 청구 데이터 기준으로만 제시합니다.