⚡ AWS SAA-C03 통합 가이드 - Part 3: 고성능 아키텍처 (24%)

> **성능 병목 제거 및 최적화 - 캐싱이 핵심!**

📋 성능 도메인 핵심 개념

성능 최적화 3원칙

1. **캐싱 전략**: 읽기 속도 향상 (CloudFront, ElastiCache, DAX)
2. **올바른 스토리지 선택**: 워크로드 특성 파악
3. **스케일 아웃 > 스케일 업**: 수평 확장 우선

병목 지점

- **스토리지 I/O**: EBS 유형 최적화
- **네트워크**: Enhanced Networking, Placement Group
- **데이터베이스**: Read Replica, 캐싱
- **글로벌 지연**: CloudFront, Global Accelerator

💽 스토리지 성능 최적화

🧠 WHY

- 스토리지는 애플리케이션 성능의 **최대 병목**
- 워크로드(랜덤 I/O, 순차 I/O, 공유 접근)에 따라 다른 스토리지 필요
- IOPS vs 처리량(Throughput) 이해 필수

⚙️ HOW

EBS 볼륨 유형 비교

유형	타입	IOPS	처리량	지연시간	GB당 비용	사용 사례
gp3	SSD	16,000	1,000 MB/s	단일 밀리초	$0.08	**범용 (권장)**
gp2	SSD	16,000	250 MB/s	단일 밀리초	$0.10	레거시
io2 BE	SSD	256,000	4,000 MB/s	서브 밀리초	$0.125	**초고성능 DB**
io2	SSD	64,000	1,000 MB/s	단일 밀리초	$0.125	미션 크리티컬
io1	SSD	64,000	1,000 MB/s	단일 밀리초	$0.125	레거시
st1	HDD	-	500 MB/s	-	$0.045	빅데이터, 로그
sc1	HDD	-	250 MB/s	-	$0.015	콜드 데이터

gp3 vs gp2 차이점

gp2: IOPS = 3 × 볼륨 크기 (GB)
  예: 100GB = 300 IOPS, 334GB = 1,000 IOPS

gp3: IOPS와 처리량 독립 설정
  기본 3,000 IOPS, 125 MB/s
  추가 비용으로 16,000 IOPS, 1,000 MB/s까지

사용 사례별 선택

워크로드	권장 볼륨	이유
일반 앱 서버	**gp3**	비용 효율적, 충분한 성능
프로덕션 DB	io2 Block Express	최고 성능, 99.999% 내구성
개발/테스트 DB	gp3	적절한 성능, 저렴
빅데이터 (순차 읽기)	st1	높은 처리량, 저렴
로그 아카이브	sc1	최저 비용
NoSQL DB (MongoDB)	io2	높은 IOPS

EBS 최적화

EBS Optimized Instance

- EC2와 EBS 간 전용 네트워크 대역폭
- 대부분 최신 인스턴스는 기본 활성화
- 네트워크 병목 제거

EBS Multi-Attach

- io1/io2만 지원
- 동일 AZ 내 최대 16개 인스턴스
- 파일시스템: Cluster-aware (GFS2, OCFS2)
- 사용 사례: Windows 클러스터, 공유 애플리케이션

스냅샷 최적화

- **증분 백업**: 변경된 블록만 저장
- **Fast Snapshot Restore (FSR)**: 즉시 사용 가능 (초기화 불필요)
- **교차 리전 복사**: 재해 복구
- **스냅샷 아카이브**: 75% 비용 절감, 24-72시간 복원

EBS 암호화

- 기본 암호화 활성화 권장

- KMS CMK 사용
- 성능 영향 미미
- 스냅샷도 자동 암호화

Instance Store

특징

- 물리적으로 연결된 **임시** 스토리지
- **매우 높은 성능** (밀리초 미만 지연)
- **휘발성**: 인스턴스 중지/종료 시 데이터 손실
- 추가 비용 없음 (인스턴스 비용에 포함)

성능 비교

```
Instance Store: 수십만 IOPS, 수 GB/s 처리량
gp3: 16,000 IOPS, 1,000 MB/s
io2 BE: 256,000 IOPS, 4,000 MB/s
```

사용 사례

- 캐시 (Redis, Memcached)
- 버퍼 (Kafka)
- 임시 파일 (렌더링, 트랜스코딩)
- 복제된 데이터 (Cassandra, HBase)

EFS vs FSx 선택

항목	EFS	FSx for Windows	FSx for Lustre	FSx for NetApp ONTAP
프로토콜	NFS v4.1	SMB	Lustre	NFS, SMB
OS	Linux	Windows	Linux	둘 다
성능	최대 10 GB/s	최대 2 GB/s	최대 1 TB/s	최대 4 GB/s
확장	자동	수동	수동	수동
비용	중간	높음	매우 높음	높음
사용 사례	공유 웹 콘텐츠	AD 통합	HPC, ML	엔터프라이즈

EFS 성능 모드

- **General Purpose**: 저지연 (99.9% < 1ms) - 권장
- **Max I/O**: 높은 처리량, 높은 지연

EFS 처리량 모드

- **Bursting**: 크기 기반 버스트 (기본)
- **Provisioned**: 고정 처리량 지정
- **Elastic**: 자동 확장 (권장)

FSx for Lustre - S3 통합

```
S3 Bucket ↔ FSx for Lustre ↔ EC2 Compute
        (자동 동기화)    (HPC 워크로드)
```

🧩 SCENARIO

상황	정답	이유
일반 웹 서버	**gp3**	비용 효율적, 충분한 성능
고성능 DB (수십만 IOPS)	**io2 Block Express**	최고 성능
빅데이터 순차 읽기	**st1**	높은 처리량
여러 Linux EC2 공유	**EFS**	NFS, 다중 AZ
Windows 공유 드라이브	**FSx for Windows**	SMB, AD 통합
HPC (고속 병렬 I/O)	**FSx for Lustre**	테라바이트급 처리량
Redis 캐시 (최고 성능)	**Instance Store**	휘발성 허용
드물게 접근하는 공유 파일	**EFS IA**	90% 비용 절감

🎯 시험 키워드 매칭

- **"범용"** → gp3
- **"고성능 DB"** → io2 Block Express
- **"빅데이터"** → st1 (HDD)
- **"최저 비용"** → sc1 (Cold HDD)
- **"여러 EC2 공유"** → EFS
- **"Windows"** → FSx for Windows
- **"HPC/ML"** → FSx for Lustre
- **"최고 성능 임시"** → Instance Store

🚫 TRAP

- ❌ EBS Multi-Attach는 모든 유형 지원 (io1/io2만)
- ❌ Instance Store는 중지 시 데이터 유지 (휘발성)
- ❌ gp2에서 IOPS 증가 = 볼륨 크기 증가 (gp3는 독립)
- ❌ EFS는 Windows 지원 (Linux만)

🚀 캐싱 전략

🧠 WHY

- **읽기 속도 향상**은 대부분 캐시가 답
- DB/오리진 서버 부하 감소 → 비용 절감
- 지연시간 단축 (밀리초 → 마이크로초)

⚙️ HOW

CloudFront (CDN)

핵심 개념

- **엣지 로케이션**: 200+ 글로벌 POP (Point of Presence)
- **Regional Edge Cache**: 엣지 로케이션과 오리진 사이
- **TTL (Time To Live)**: 캐시 만료 시간
- **Cache Hit Ratio**: 캐시 적중률 (높을수록 좋음)

Origin 유형

- S3 Bucket
- ALB / NLB
- EC2 Instance
- API Gateway
- MediaStore / MediaPackage
- Custom HTTP Server

Cache Key 구성요소

- URL 경로
- Query String (선택적)
- Headers (선택적)
- Cookies (선택적)

Cache Key 최적화

> 나쁜 예: 모든 헤더 포함 → 캐시 적중률 낮음
> 좋은 예: 필수 헤더만 포함 → 캐시 적중률 높음

Cache Invalidation

- 캐시 강제 무효화
- 비용 발생 (처음 1,000개 무료)
- 경로 지정: `/images/*`, `/path/to/file.jpg`

Cache Behaviors

- 경로 패턴별 설정
- 우선순위 지정
- 예:

```
/api/* → TTL 0 (캐시 안 함)
/images/* → TTL 86400 (1일)
/* → TTL 3600 (1시간)
```

ElastiCache

Redis vs Memcached

항목	Redis	Memcached
데이터 구조	고급 (List, Set, Hash, Sorted Set)	단순 (키-값)
영구 저장	✅ (AOF, RDB)	❌
복제	✅ (Primary-Replica)	❌
Multi-AZ	✅ (Cluster Mode)	❌
백업	✅	❌
Pub/Sub	✅	❌
트랜잭션	✅	❌
Lua 스크립트	✅	❌
멀티스레드	❌ (단일 스레드)	✅
샤딩	Cluster Mode	자동
사용 사례	**대부분 (권장)**	단순 캐시

Redis Cluster Mode

- **Disabled**: 단일 샤드, 복제 지원
- **Enabled**: 다중 샤드 (최대 500개 노드), 수평 확장

Redis 사용 패턴

1. Lazy Loading (Cache-Aside)

1. 애플리케이션이 캐시 조회
   ↓ (Miss)
2. DB에서 데이터 읽기
   ↓
3. 캐시에 저장
   ↓
4. 애플리케이션에 반환

- 장점: 필요한 데이터만 캐시
- 단점: 첫 요청은 느림 (Cache Miss)

2. Write-Through

1. 애플리케이션이 DB에 쓰기
   ↓
2. 동시에 캐시에도 쓰기
   ↓
3. 완료

- 장점: 캐시 항상 최신
- 단점: 쓰기 지연 증가

3. TTL (Time To Live)

- 자동 만료로 stale 데이터 방지
- Lazy Loading과 함께 사용 권장

Redis 사용 사례

- 세션 저장소 (로그인 정보)
- DB 쿼리 캐싱
- 실시간 리더보드 (Sorted Set)
- 채팅 메시지 (Pub/Sub)
- Rate Limiting (Incr, Expire)
- 분산 락 (Redlock)

DynamoDB Accelerator (DAX)

특징

- **DynamoDB 전용** 인메모리 캐시
- 완전 관리형 (패치, 장애 조치 자동)
- **마이크로초 지연** (10배 이상 향상)

- DynamoDB API 완전 호환 (코드 변경 최소)

DAX vs ElastiCache

항목	DAX	ElastiCache
대상	DynamoDB 전용	범용 (모든 데이터)
API	DynamoDB API	Redis/Memcached API
통합	완벽 통합	애플리케이션 레벨
사용 사례	DynamoDB 읽기 최적화	복잡한 캐싱 로직

DAX 작동 방식

```
Application → DAX Cluster → DynamoDB
              ↓ (Cache Hit)
            Return from Cache
              ↓ (Cache Miss)
            Query DynamoDB → Cache Result
```

DAX 캐시 유형

- **Item Cache**: GetItem, BatchGetItem
- **Query Cache**: Query, Scan

TTL

- Item Cache: 기본 5분
- Query Cache: 기본 5분
- 설정 가능: 1초 ~ 1시간

캐싱 레이어 설계

3-Tier 캐싱

```
Client
  ↓
CloudFront (CDN, 글로벌 캐시)
  ↓
ALB → EC2
  ↓
ElastiCache (애플리케이션 캐시)
  ↓
RDS (데이터베이스)
```

읽기 흐름

1. CloudFront에서 정적 콘텐츠 제공
2. 동적 콘텐츠는 ALB → EC2
3. EC2가 ElastiCache 조회 (Cache Hit)
4. Cache Miss 시 DB 조회 → 캐시 저장

🧩 SCENARIO

상황	정답	이유
전 세계 정적 콘텐츠	**CloudFront**	엣지 캐싱
DB 쿼리 응답 지연	**ElastiCache (Redis)**	복잡한 쿼리 캐싱
DynamoDB 읽기 과다	**DAX**	DynamoDB 전용
세션 저장	**ElastiCache (Redis)**	영구 저장 + Multi-AZ
단순 키-값 캐시	**ElastiCache (Memcached)**	멀티스레드
실시간 리더보드	**ElastiCache (Redis Sorted Set)**	순위 지원
API 응답 캐싱	**API Gateway Cache**	내장 캐시

🎯 시험 키워드 매칭

- **"전 세계 속도"** → CloudFront
- **"DB 부하 감소"** → ElastiCache
- **"DynamoDB 읽기 최적화"** → DAX
- **"세션 저장"** → Redis
- **"단순 캐시"** → Memcached
- **"마이크로초"** → DAX

🚫 TRAP

- ❌ CloudFront는 DB 캐시 불가 (정적 콘텐츠 전용)
- ❌ Memcached는 자동 페일오버 불가
- ❌ DAX는 RDS 캐시 불가 (DynamoDB 전용)
- ❌ Redis Cluster Mode Disabled는 샤딩 불가

⚙️ 데이터베이스 성능 최적화

🧠 WHY

- 워크로드(OLTP, OLAP, NoSQL)에 맞는 엔진 선택 필수
- 읽기/쓰기 부하 분리
- 인덱스 및 쿼리 최적화

⚙️ HOW

RDS 성능 최적화

Read Replica

- 읽기 부하 분산 (최대 5개)
- 비동기 복제 (약간의 지연)
- 쓰기는 Primary만
- 프로모션 가능 (수동)

Parameter Group

- 데이터베이스 엔진 설정
- 예:
  - `max_connections`: 최대 연결 수
  - `query_cache_size`: 쿼리 캐시 크기
  - `innodb_buffer_pool_size`: InnoDB 버퍼 풀

Enhanced Monitoring

- OS 레벨 메트릭 (CPU, 메모리, 디스크 I/O)
- 1초 단위 세밀한 모니터링
- CloudWatch Logs로 전송

Performance Insights

- 쿼리 성능 분석
- Top SQL 식별
- 대기 이벤트 분석
- 7일 무료 (최대 2년 보관)

Connection Pooling

- RDS Proxy 또는 애플리케이션 레벨
- 연결 재사용
- Lambda 최적화

Aurora 고급 기능

스토리지 아키텍처

- 자동 확장 (10GB → 128TB)

- 6개 복제본 (3 AZ)
- 4/6 쓰기, 3/6 읽기 쿼럼
- 자동 복구 (10초 이내)

Reader Endpoint

- 읽기 부하 자동 분산
- 연결 로드 밸런싱
- 최대 15개 Read Replica

Custom Endpoint

- 특정 인스턴스 그룹 지정
- 예:
  - 분석 쿼리용 그룹 (더 큰 인스턴스)
  - OLTP 쿼리용 그룹 (더 많은 인스턴스)

Aurora Serverless v2

- 즉시 확장 (0.5 ACU~)
- 프로비저닝 모드와 혼용 가능
- 간헐적 워크로드
- 개발/테스트 환경

Aurora Global Database

```
Primary Region (Read/Write)
  ↓ (< 1초 복제)
Secondary Region 1 (Read)
Secondary Region 2 (Read)
...
Secondary Region 5 (Read)
```

- RPO < 1초
- RTO < 1분
- 교차 리전 읽기 확장

Aurora Parallel Query

- 쿼리를 스토리지 레이어로 푸시
- 대량 분석 쿼리 가속 (100배)
- 자동 활성화 (특정 쿼리만)

Aurora Auto Scaling

- Read Replica 자동 추가/제거
- CPU, 연결 수 기반

DynamoDB 성능 최적화

용량 모드

모드	설명	비용	사용 사례
On-Demand	자동 확장, 요청당 과금	높음	예측 불가 트래픽
Provisioned	RCU/WCU 지정, 저렴	낮음	예측 가능 트래픽

Provisioned Mode + Auto Scaling

- Target Utilization: 70% (권장)
- 최소/최대 용량 지정
- 비용 효율적

인덱스

GSI (Global Secondary Index)

- 다른 파티션 키
- 언제든 추가/삭제
- 별도 RCU/WCU
- 최대 20개

LSI (Local Secondary Index)

- 같은 파티션 키, 다른 정렬 키
- **테이블 생성 시에만** 추가
- Primary 테이블의 RCU/WCU 공유
- 최대 5개

최적화 기법

1. Partition Key 설계

```
나쁜 예: user_id (핫 파티션 발생)
좋은 예: user_id + date (균등 분산)
```

2. Batch Operations

- BatchGetItem: 최대 100개

- BatchWriteItem: 최대 25개
- 네트워크 왕복 최소화

3. TTL (Time To Live)

- 자동 만료 (무료)
- Unix 타임스탬프
- 48시간 이내 삭제

4. DynamoDB Streams

- 24시간 보관
- Lambda 트리거
- Change Data Capture (CDC)

5. Global Tables

- 다중 리전 복제 (Active-Active)
- 초 단위 복제
- 충돌 해결 (Last Writer Wins)

🧩 SCENARIO

상황	정답	이유
RDS 읽기 부하 급증	**Read Replica**	최대 5개까지
초당 수천 TPS	**Aurora**	고성능 아키텍처
비정형 데이터, 서버리스	**DynamoDB**	NoSQL, 자동 확장
예측 불가 트래픽	**DynamoDB On-Demand**	사용량 기반
여러 리전 동기화	**Aurora Global** 또는 **DynamoDB Global**	< 1초 복제
분석 쿼리 분리	**Aurora Custom Endpoint**	그룹 지정
간헐적 DB	**Aurora Serverless**	자동 스케일링
대량 분석 쿼리	**Aurora Parallel Query**	100배 가속

🎯 시험 키워드 매칭

- **"초고성능 RDB"** → Aurora
- **"밀리초 응답"** → DynamoDB
- **"글로벌 복제"** → Aurora Global 또는 DynamoDB Global
- **"서버리스 DB"** → Aurora Serverless
- **"예측 불가"** → DynamoDB On-Demand

🚫 TRAP

- ❌ Aurora Reader는 쓰기 불가
- ❌ DynamoDB는 JOIN 불가 (비정규화 필요)
- ❌ LSI는 나중에 추가 불가 (테이블 생성 시에만)
- ❌ Provisioned 모드 초과 시 스로틀링 (Auto Scaling 권장)

🧠 Lambda & API Gateway

🧠 WHY

- **서버리스** = 운영 오버헤드 제거 + 초 단위 확장
- 사용량 기반 과금 (유휴 비용 0)
- 이벤트 기반 아키텍처

⚙️ HOW

Lambda 제한 및 최적화

제한 사항

항목	제한	증가 가능
메모리	128MB ~ 10GB	❌
실행 시간	최대 15분	❌
동시 실행	1,000개 (리전별)	✅
패키지 크기	50MB (압축), 250MB (압축 해제)	❌
임시 디스크 (/tmp)	512MB ~ 10GB	❌
환경 변수	4KB	❌

콜드 스타트

- 새 실행 환경 초기화 시간
- 영향 요인:
  - 메모리 크기 (높을수록 빠름)
  - 패키지 크기 (작을수록 빠름)
  - VPC 구성 (ENI 생성 시간)
  - 런타임 (해석형 > 컴파일형)

콜드 스타트 최소화

- **Provisioned Concurrency**: 미리 초기화 (추가 비용)

- 메모리 증가: CPU도 함께 증가
- 패키지 최소화: Lambda Layers 사용
- VPC 피하기: 가능하면 VPC 외부

Lambda Layers

- 공통 라이브러리 재사용
- 최대 5개 레이어
- 총 크기 250MB 제한

Reserved Concurrency

- 특정 함수 동시 실행 수 예약
- 다른 함수 영향 방지
- Throttling 방지

환경 변수 vs Parameter Store

항목	환경 변수	Parameter Store
크기	4KB	4KB (Standard), 8KB (Advanced)
암호화	KMS	KMS
버전 관리	❌	✅
비용	무료	무료 (Standard)
동적 로드	❌	✅

VPC Lambda

- VPC 리소스 접근 (RDS, ElastiCache)
- ENI 생성 필요
- 콜드 스타트 증가 (Hyperplane으로 개선)
- NAT Gateway 필요 (인터넷 접근 시)

API Gateway

API 유형 비교

항목	REST API	HTTP API	WebSocket API
프로토콜	HTTP/HTTPS	HTTP/HTTPS	WebSocket
기능	완전한 기능	기본 기능	양방향 통신
성능	보통	**빠름**	보통
비용	$3.50/100만	**$1.00/100만** (70% 절감)	$1.00/100만
캐싱	✅	❌	❌
WAF 통합	✅	❌	❌
Usage Plans	✅	❌	❌
사용 사례	복잡한 API	**대부분 (권장)**	채팅, 실시간

인증 방법

방법	설명	사용 사례
IAM	AWS 자격 증명	AWS 서비스 간
Cognito	사용자 풀	모바일/웹 앱
Lambda Authorizer	사용자 정의	OAuth, SAML
API Key	간단한 키	파트너 API

Throttling

- 계정 레벨: 10,000 RPS
- API 레벨: 사용자 지정
- Stage 레벨: 사용자 지정
- Method 레벨: 사용자 지정

캐싱

- TTL: 0초 ~ 3,600초 (1시간)
- 크기: 0.5GB ~ 237GB
- 비용: 캐시 크기당 시간당
- 캐시 키: 경로, 쿼리, 헤더

Usage Plans & API Keys

- Rate: 초당 요청 수
- Burst: 순간 최대 요청 수
- Quota: 일/주/월 요청 수
- API Key로 클라이언트 식별

🧩 SCENARIO

상황	정답	이유
하루 몇 번 실행	**Lambda**	유휴 비용 0
HTTP API, 저비용	**API Gateway HTTP API**	70% 절감
WebSocket 필요	**API Gateway WebSocket**	양방향
인증 필요	**API Gateway + Cognito**	사용자 풀
콜드 스타트 제거	**Provisioned Concurrency**	밀리초 응답
VPC 리소스 접근	**Lambda VPC 구성**	ENI
API 응답 캐싱	**API Gateway Cache**	성능 향상
공통 라이브러리	**Lambda Layers**	재사용

🎯 시험 키워드 매칭

- **"서버리스"** → Lambda
- **"저비용 API"** → HTTP API
- **"실시간"** → WebSocket
- **"콜드 스타트"** → Provisioned Concurrency

🚫 TRAP

- ❌ Lambda는 15분 이상 불가 (Fargate/ECS 사용)
- ❌ Lambda VPC는 NAT Gateway 필요 (인터넷 접근)
- ❌ API Gateway REST가 항상 필요 (HTTP API로 충분한 경우 많음)

📦 EC2 성능 튜닝

🧠 WHY

- 인스턴스 선택이 성능/비용 결정
- 워크로드별 CPU, 메모리, I/O, GPU 요구 다름

⚙️ HOW

인스턴스 패밀리

패밀리	비율 (vCPU:RAM)	설명	사용 사례
T	1:2	Burstable, CPU 크레딧	저부하, 테스트
M	1:4	General Purpose	범용 웹 서버
C	1:2	Compute Optimized	배치, 게임, 과학
R	1:8	Memory Optimized	DB, 캐시
X	1:16	Extreme Memory	SAP HANA
I	높은 IOPS	Storage Optimized	NoSQL DB
D	높은 HDD	Dense Storage	Hadoop, HDFS
P	NVIDIA GPU	GPU	ML 학습
G	NVIDIA GPU	Graphics	렌더링, 스트리밍
Inf	AWS Inferentia	Inference	ML 추론
F	FPGA	프로그래머블 하드웨어	금융, 암호화

Burstable Instance (T3, T4g)

- CPU 크레딧 기반
- 기본 성능 + 버스트 성능
- 크레딧 소진 시 기본 성능으로 제한
- **Unlimited Mode**: 추가 비용으로 무제한 버스트

Nitro System

- 최신 하이퍼바이저
- 더 나은 성능 (EBS, 네트워크)
- 더 많은 인스턴스 크기
- Enhanced Networking 기본

Enhanced Networking

ENA (Elastic Network Adapter)

- 최대 100 Gbps
- 낮은 지연시간
- 높은 PPS (Packets Per Second)
- 대부분 최신 인스턴스 기본

EFA (Elastic Fabric Adapter)

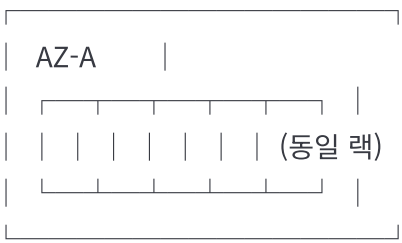
- HPC 전용
- OS 바이패스

- MPI (Message Passing Interface)
- 낮은 지연시간 (마이크로초)

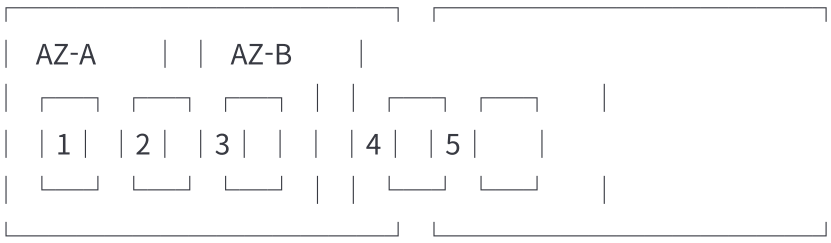
Placement Group

유형	설명	AZ	지연시간	사용 사례
Cluster	단일 AZ, 가까운 배치	단일	**최저**	HPC, 빅데이터
Spread	다른 하드웨어	다중 (최대 7개/AZ)	보통	고가용성
Partition	논리적 파티션 분리	다중	보통	Hadoop, Kafka, Cassandra

Cluster Placement Group



Spread Placement Group



(각각 다른 랙)

EC2 최적화 팁

- Compute Optimizer로 권장 크기 확인
- CloudWatch 메트릭 모니터링
- 적절한 인스턴스 패밀리 선택
- Burstable은 지속 부하 피하기

🧩 SCENARIO

상황	정답	이유
DB 서버, 메모리 중심	**R 시리즈**	1:8 비율
배치 처리, CPU 중심	**C 시리즈**	Compute Optimized
ML 학습	**P 시리즈**	NVIDIA GPU
저지연 HPC	**Cluster Placement Group**	단일 AZ

상황	정답	이유
고가용성	**Spread Placement Group**	다른 하드웨어
테스트 환경	**T 시리즈**	Burstable, 저렴
Hadoop 클러스터	**Partition Placement Group**	논리적 분리

🎯 시험 키워드 매칭

- **"메모리 집약"** → R 시리즈
- **"CPU 집약"** → C 시리즈
- **"ML/AI"** → P, Inf
- **"저지연"** → Cluster
- **"고가용성"** → Spread

🚫 TRAP

- ❌ Cluster는 Multi-AZ 불가
- ❌ Burstable은 지속 부하 부적합
- ❌ EFA는 일반 인스턴스 불가 (특정 인스턴스만)

🌎 CloudFront + Global Accelerator

🧠 WHY

- CloudFront: 콘텐츠 캐싱
- Global Accelerator: 네트워크 최적화

⚙️ HOW

CloudFront vs Global Accelerator

항목	CloudFront	Global Accelerator
목적	콘텐츠 캐싱	네트워크 가속
프로토콜	HTTP/HTTPS	**TCP/UDP**
고정 IP	❌	✅ (Anycast, 2개)
캐싱	✅	❌
사용 사례	정적/동적 콘텐츠	게임, VoIP, IoT

Global Accelerator

- Anycast IP: 2개 고정 IP
- Health Check: 자동 장애 조치

- AWS 백본망 사용 (60% 향상)
- TCP/UDP 트래픽

🧩 SCENARIO

상황	정답	이유
정적 파일	**CloudFront**	캐싱
TCP 성능 향상	**Global Accelerator**	네트워크 가속
고정 IP 필요	**Global Accelerator**	Anycast

🎯 시험 키워드 매칭

- **"캐싱"** → CloudFront
- **"고정 IP"** → Global Accelerator
- **"TCP/UDP"** → Global Accelerator

🚫 TRAP

- ❌ Global Accelerator는 캐싱 불가
- ❌ CloudFront는 TCP 가속 불가

📋 성능 도메인 최종 체크리스트

✅ 스토리지

- [ ] gp3 vs io2 Block Express
- [ ] EFS vs FSx
- [ ] Instance Store 사용 사례

✅ 캐싱

- [ ] CloudFront Cache Behaviors
- [ ] Redis vs Memcached
- [ ] DAX (DynamoDB 전용)

✅ 데이터베이스

- [ ] Aurora 특수 기능
- [ ] DynamoDB GSI vs LSI
- [ ] Read Replica

✅ 서버리스

- [ ] Lambda 제한
- [ ] API Gateway 유형
- [ ] Provisioned Concurrency

✅ EC2

- [ ] 인스턴스 패밀리
- [ ] Placement Group
- [ ] Enhanced Networking

🎯 성능 최적화는 캐싱이 핵심입니다! CloudFront → ElastiCache → DAX 순서로 기억하세요.

다음: Part 4 - 비용 최적화 (20%)