**DevOps 엔지니어 채용 과제**

**(Option 2: 운영 시나리오 기반 심층 분석)**

# 장애 대응 시나리오

● **상황: 월말 피크 타임에 DB Connection Pool 고갈 발생**

● **요구사항:**

- 즉시 대응 절차 (5분 내)

- 근본 원인 분석 방법

- 재발 방지를 위한 인프라/코드 개선안

**1.1 즉시 대응 절차 (5분 내)**

- 우선 모니터링의 slack 알림이나 로그등을 확인하여 에러를 파악한 후 관련 엔지니어와 개발자에게 상황을 전파 합니다.

**-** 특정 pod의 커넥션풀 고갈시 istio등의 circuit breaker기능을 사용하여 문제 있는 pod로의 호출을 차단하고 복구가 되면 다시 호출을 허용합니다.

- App의 커넥션풀 부족시 pod를 추가 생성하여 빠르게 커넥션풀을 확보합니다.

- DB 서버에 대한 Auto-scaling 또는 Read Replica를 활성화 합니다.

- DB서버가 다운되어 다른 서비스로의 장애전파를 예방하기 위해 pool이 고갈된 서비스의 네트워크 레벨인 WAF, ELB, Ingress Controller에서 rate limit와 connection limit를 설정합니다.

**1.2 근본 원인 분석 방법**

- connection 반환 누락이 있는지 점검합니다. (예: exception 발생 후 close 미호출)

- Slow Query를 파악 및 쿼리 튜닝을 통해 Connection pool 장기 점유 발생하지 않도록 합니다.

- Pool 설정이 적절히 되었는지 확인합니다. DB의 가용 connection 수량에 비해 app에서 적거나 크게 사용하는등 Pool 설정 불일치를 확인합니다.

**1.3 재발 방지를 위한 인프라/코드 개선안**

- Timeout정책 강화 및 app레벨의 메트릭과 모니터링 도구의 대시보드를 활용하여 Connection leak 모니터링을 추가하는등 Connection Pool 관리를 개선합니다.

- connection pool 사용량 메트릭을 기반으로 하는 서비스 오토스케일링 정책을 도입합니다.

- RDS의 Read Replica를 도입하여 읽기 전용 요청을 Replica로 분산 처리합니다.

- 빠른 Read성능과 DB connection pool 절약을 위해 Redis등 캐싱 시스템을 도입합니다

- 비효율적이고 처리가 느린 SQL쿼리 및 소스코드를 개선합니다. (N+1, Full Scan등)

# 비용 최적화 시나리오

● **상황: AWS 월 비용이 예상보다 30% 초과**

● **요구사항:**

- 비용 분석 접근법 (어떤 도구/지표 확인)

- 최적화 우선순위 (예: 컴퓨팅, 스토리지, 네트워크)

- 구체적 개선안 3가지 이상

**2.1 비용 분석 접근법 (어떤 도구/지표 확인)**

- AWS Cost Explorer를 사용하여 전체 비용 트렌드, 리소스별/서비스별 지출을 분석 합니다.

**-** AWS Trusted Advisor의 Cost Optimization 기능을 사용해서 리소스의 비용 효율성 평가, 불필요한 리소스 제거 권고등의 기능을 활용합니다.

- AWS CloudWatch Metrics 및 Prometheus/Grafana등을 활용해 프로비저닝된 스펙대비 리소스 활용률을 확인합니다.

- MSP사에서 제공하는 비용관리 서비스를 활용해 유휴리소스를 탐지합니다.

- 위 도구들을 활용하여 확인할 주요 지표는 CPU, Memory, EBS, S3, Data Transfer비용 입니다.

**2.2 최적화 우선 순위 및 구체적 개선안**

|  |  |  |
| --- | --- | --- |
| **우선 순위** | **대상** | **설명** |
| 1 | 컴퓨팅 | 1.실제 사용량을 분석하여 너무 과도하게 설정된 스펙을 scale down  2. RI혹은 spot instance를 활용해 컴퓨팅 비용 절감 |
| 2 | 네트워크 | 1. CDN 캐시를 도입하여 정적 콘텐츠 호출시 발생하는 네트워크 비용 절약  2. VPC내 혹은 EKS 클러스터내 내부 통신이 가능함에도 퍼블릭 DNS 주소 사용등으로 외부 인터넷으로 리소스가 호출되는 부분을 개선하여 비용절감  3. Istio등 ServiceMesh를 도입하여 같은 AZ에 있는 리소스들을 우선적으로 호출하도록 Locality-Aware Load Balancing 설정 활용 |
| 3 | 스토리지 | 1. 스토리지Lifecycle 정책을 수립하여 사용빈도가 낮은 파일등은 S3의 저렴한 class로 이동  2. 인프라나 app에서 발생하는 불필요한 중복성 로그나 debug레벨의 로깅등을 제거하여 스토리지 비용 절감 |

# 보안 강화 시나리오

● **상황: 보안 감사에서 IAM 권한이 과도하게 설정되어 있다는 지적**

● **요구사항:**

- 최소 권한 원칙(Principle of Least Privilege) 적용 계획

- Secrets 관리 개선안

- 네트워크 보안 강화 방안

**3.1 최소 권한 원칙(Principle of Least Privilege) 적용 계획**

- AdministratorAccess권한이 적용되어있는 사용자가 있는지 식별하여 제거

- 개별 인원 및 팀별 필수로 접근해야 하는 리소스를 파악하여 계정/그룹별 세밀한 권한 부여

- 실제 사용 API 기준으로 Custom Policy를 생성하여 Managed Policy → Custom Policy 전환

**3.2 Secrets 관리 개선안**

- DB접속정보, 외부API키등 민감한정보는 소스 코드나 GitHub에 노출되지 않도록 Vault를 활용

- Vault에 저장된 시크릿을 사용하기 위해서 Vault서버로의 API 호출이나 k8s Secret으로 생성후 시크릿 마운트를 하는 방법을 통해 안전하게 시크릿을 저장하고 사용

- Vault 저장소에 감사로그 기록을 활성화하여 사용 현황을 주기적으로 모니터링 및 slack 알람을 전송하는 프로세스 도입

**3.3 네트워크 보안 강화 방안**

- app이나 리소스별 필요한 inbound/outbound를 파악후 필요한 포트만 허용 정책을 추가하여 Security Group 최소화

- Network ACL을 사용하여 불필요한 outbound를 차단하여 VPC레벨의 전반적인 네트워크 보안 강화

- 보안이 중요한 RDS, 백엔드시스템 그리고 불필요하게 Public subnet에서 운영중인 서비스가 있는지 파악하여 Private Subnet으로 이동시키고 Bastion Host를 도입하여 허용된 인원과 장소에서만 private subnet의 자원에 접근할 수 있도록 구성

- VPC Flow Logs, GuardDuty, CloudTrail등 AWS에서 제공하는 보안 서비스들을 적극 활용하여 보안강화

- ELB, CloudFront에 WAF를 적용하여 SQL Injection, XSS등 웹공격을 차단해 L7(애플리케이션 계층) 보안을 강화