포트폴리오

엔지니어 김동욱

- VMware 기반 리눅스, 윈도우 서버 운영 및 네트워크 시뮬레이션 경험
- 클라우드(AWS, GCP) 환경에서의 인프라 설계 및 트래픽 대응 아키텍처 구성
- Shell Script 및 Crontab을 활용한 자동화와 운영 효율화 실습 수행
- 고가용성 설계, 백업 자동화, DB 성능 최적화까지 시스템 운영의 핵심 요소 실습
- 팀 프로젝트 내에서 설계, 구현, 테스트, 장애 대응 등 전 과정 주도

Skill Set

클라우드 및 인프라

AWS	EC2, RDS, ALB, Auto Scaling, S3, CloudWatch, IAM, VPC, EKS, Route 53, Aurora MySQL, DynamoDB, SNS
GCP	VPC, Load Balancer, Cloud SQL, HA VPN
VMware	vSphere 기반 리눅스 가상 서버 구축 및 운영

인프라 자동화 및 운영 도구

Terraform	VPN 구성 및 인프라 코드 관리
GitHub	코드 관리 및 CI/CD 연동 (YAML 기반)
Cloud Shell	GCP 환경 CLI 기반 인프라 운영
Shell Script	Bash Shell을 통한 서버 운영 및 관리

네트워크 및 보안

Packet Tracer	네트워크 시뮬레이션 (NAT, RIP, DHCP)
Site-to-Site VPN	AWS-GCP 간 보안 터널 구성 (정적 라우팅)
IAM	사용자 및 서비스 최소 권한 정책 적용

운영체제 및 데이터베이스

Linux	CentOS 7.9, Ubuntu 서버 운영 및 관리
Windows Server	Windows Server 2022 운영 경험
DB	RDS (Aurora MySQL, MariaDB), Oracle(DBeaver), NoSQL(DynamoDB), 인덱스 설정 및 모니터링

웹/애플리케이션

Docker	컨테이너 이미지 생성 및 EKS 클러스터 배포
Apache + PHP	웹 서버 구성 및 PHP 연동
CI/CD	AWS CodePipeline을 활용한 자동화된 배포 파이프라인

스크립트 및 기타

Bash Shell	사용자 계정 생성, 디스크 관리, 권한 설정, 자동화 스크립트 작성
Python	간단한 자동화 및 스크립트 작성

Project1

VMWare 기반 온프레미스 인프라 구축 프로젝트

- 프로젝트 기간 : 2024.11.13 ~ 2024.11.27
- 구축 환경 : VMWare, Packet Tracer, CentOS 7.9, Windows Server

프로젝트 솔루션

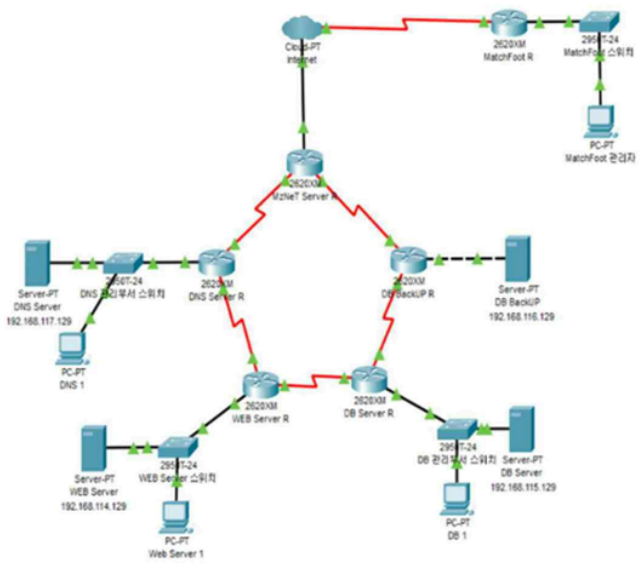
트래픽 증가로 기존 시스템의 성능 한계를 극복하기 위해 초기 구축 비용을 절감하고 신속히 운영 가능한 단일 서버 도입 방안을 제안하였습니다. 이는 VMware 기반의 단일 서버를 통해 웹 서버와 데이터베이스 서버를 통합하여 초기 운영을 간단하고 빠르게 가능하게 하였습니다.

사용자 증가로 서버 자원의 한계가 발생하면서 성능 저하, 확장성 부족, 보안 취약성 문제가 발생할 수 있다는 점을 확인하였고 이를 해결하기 위해 웹 서버와 DB 서버를 역할별로 분리하는 방안을 제안하여 각 서버의 성능을 최적화할 수 있도록 구성하였습니다. 네트워크 구성은 Packet Tracer를 활용하여 NAT, RIP, DHCP 설정을 통해 트래픽을 분산하고 안정성을 확보할 수 있도록 제안하였습니다.

또한 데이터 손실 및 복구 지연 문제를 방지하기 위해 Crontab을 활용한 주기적인 백업과 SCP를 사용한 원격 백업 방안을 제안하여 데이터 보호와 신속한 복구가 가능하도록 하였습니다.

기술 스택

기술 스택	설명
가상화 서버	VMWare 기반 Linux (CentOS7.9) 서버 환경
네트워크	Packet Tracer를 활용한 네트워크 시뮬레이션 (NAT, RIP, DHCP)
웹 서버	CentOS 기반 Apache (PHP연동)
DB 서버	MariaDB (데이터 저장 및 관리), PHPMyAdmin (테이블 설정)
백업 서버	Crontab, Shell Script(bash)



담당 역할

담당 역할	세부 내용
네트워크 구성	Packet Tracer를 통해 NAT, RIP, DHCP 설정
웹 및 DB 서버 구축	VMWare CentOS 7.9 기반 웹서버 (Apache) 및 DB 서버 (MariaDB) 설치
백업 스크립트 구현	SCP 를 통해 원격 백업 서버로 자동 백업 파일 전송

성과 및 기대효과

항목	개선 전 (로컬 테스트)	개선 후 (가상화 기반 인프라)
확장성	물리적 서버 확장 불가능	가상 서버를 통해 서버 추가 가능
네트워크 구성	단일 네트워크	Packet Tracer 기반 다중 서브넷 및 트래픽 분산 구조
DB 관리	수동 백업	Crontab + SCP로 백업 및 원격 전송
시스템 복구	수동 복구	VMWare Snapshot을 통한 복구 가능

프로젝트 후기

1. 네트워크 구성

- **목표** : 가상화 환경에서 실무형 네트워크 아키텍처 구성
- **결과** : Packet Tracer를 활용하여 NAT, RIP, DHCP 설정을 통해 네트워크 트래픽 분산 및 라우팅 관리

2. 웹 서버 및 DB 서버 구축

- **목표** : 안정적인 웹 서비스 및 데이터베이스 운영 환경 구성
- **결과** : 웹 서버와 DB 서버 간 연결 설정 및 PHP 연동 확인

3. 백업 스크립트 구현

- **목표** : 데이터 보호 및 백업 관리
- **결과** : SCP로 원격 서버로 백업 파일 전송 및 Shell Script로 백업 파일 생성, 압축, 전송 구현

4. 느낀점

이번 프로젝트를 통해 **가상화 기반의 인프라 환경 구성과 네트워크 시뮬레이션의 중요성**을 깊이 느꼈습니다.

Packet Tracer를 사용하여 실제 네트워크와 유사한 환경을 시뮬레이션하며 **서브넷 분리, NAT, RIP 설정**을 통해 네트워크 트래픽 분산과 라우팅을 이해할 수 있었습니다.

Crontab과 SCP를 통한 백업 자동화는 서버 운영의 안정성을 보장하는 핵심 요소임을 실감했고 주기적인 백업 관리와 안전한 파일 전송의 중요성을 깨달았습니다.

추후에는 이러한 자동화 기술을 클라우드 환경에서도 확장 적용하여 **보다 효율적인 백업 및 복구 체계**를 구축하고자 합니다.

Project 2

EC2 & RDS 기반 고가용성 및 재해 복구 설계 프로젝트

- 프로젝트 기간 : 2024.12.27 ~ 2025.01.17
- 구축 환경 : AWS, Ubuntu

프로젝트 솔루션

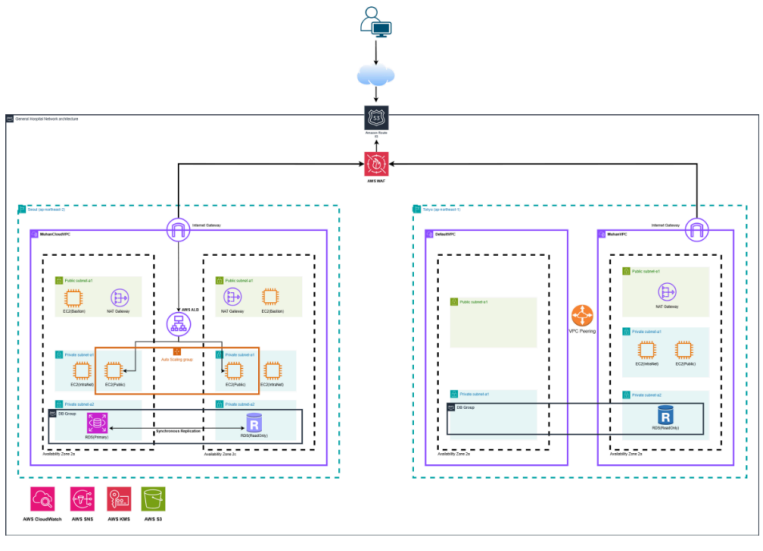
트래픽 증가로 인한 시스템 성능 저하를 해결하기 위해 클라우드 기반의 고가용성 인프라 도입 방안을 제안하였습니다. 초기에는 AWS 클라우드에서 EC2 인스턴스를 활용하여 웹 서버와 데이터베이스 서버를 통합하여 운영함으로써 신속한 서비스 운영을 가능하게 하였습니다.

사용자 증가에 따라 서버 자원의 한계와 성능 저하가 발생할 수 있음을 확인하였고 이를 해결하기 위해 Auto Scaling을 통해 웹 서버의 자동 확장을 가능하도록 하였습니다. 또한 ALB를 도입하여 사용자 요청을 웹 서버에 분산하고 Route53을 통해 도메인 관리와 장애 조치 (Failover) 구성을 제안하였습니다.

보안 강화를 위해 IAM을 활용하여 사용자 및 권한 관리를 수행하고 VPC 네트워크를 통해 퍼블릭 및 프라이빗 서브넷을 분리하여 외부 접근을 통제할 수 있도록 구성하였습니다. DB는 RDS의 자동 백업 기능을 통해 데이터 보호를 강화하고 CloudWatch 모니터링으로 시스템 성능을 실시간으로 확인할 수 있도록 제안하였습니다.

기술 스택

기술 스택	설명
EC2	PHP 웹 서버 운영
DB	RDS (MariaDB) Multi-AZ
Auto Scaling	사용자 사용률 기반 서버 자동 확장
ALB	트래픽 분산 및 헬스체크 설정
Route 53	멀티 리전 장애 대응을 위한 Failover 설정



담당 역할

담당 역할	세부 내용
웹 서버 및 DB 구축	EC2 PHP 웹 서버 구성, RDS (MariaDB) Multi-AZ 설정
Auto Scaling 및 ALB 설정	사용자 사용률 기반 Auto Scaling 설정 및 ALB로 트래픽 분산
CloudWatch 모니터링 및 알람	EC2, RDS, ALB 모니터링, 지정 임계치 초과 시 SNS 알람 설정
재해 복구 및 백업	Route 53 Failover 구성으로 멀티 리전 자동 전환

성과 및 기대효과

항목	개선 전 (단일 서버 환경)	개선 후 (HA 및 자동화 아키텍처)
트래픽 대응	고정 서버 수 (수동 확장)	Auto Scaling으로 서버 확장
장애 대응	DNS 수동 전환	Route 53 Failover 시 자동 전환
DB 가용성	단일 AZ	Multi-AZ 자동 장애 복구
서버 모니터링	수동 개별 서버 점검	CloudWatch 실시간 모니터링 및 알람

프로젝트 후기

1. 웹 서버 및 DB 구축

- **목표** : AWS 기반 웹 서비스 및 DB 환경 구성
- **결과** : EC2 인스턴스에 PHP 웹 서버 구축, RDS (MariaDB)로 데이터베이스 분리

2. Auto Scaling 및 ALB 설정

- **목표** : 웹 서비스 자동 확장 및 트래픽 분산
- **결과** : Auto Scaling과 ALB를 통해 트래픽 급증 상황에서도 안정적인 서비스 제공

3. CloudWatch 모니터링 및 알람 설정

- **목표** : 서버 상태 모니터링 및 장애 발생 시 대응
- **결과** : CloudWatch를 통해 EC2 및 RDS의 CPU 사용률, 네트워크 트래픽 실시간 모니터링과 임계치 초과 시 SNS 알람 설정

4. 데이터 백업 및 재해 복구

- **목표** : 데이터 손실 방지 및 신속한 복구 보장
- **결과** : RDS Multi-AZ 구성으로 자동 장애 복구, Route 53 Failover Policy로 자동 장애 전환

5. 느낀점

이번 프로젝트를 통해 **클라우드 기반 인프라의 고가용성과 자동화된 장애 복구의 중요성**을 깊이 느꼈습니다.

Auto Scaling과 ALB 설정을 통해 트래픽 증가에 자동으로 대응할 수 있었으며 Route 53을 활용한 Failover 구성으로 **멀티 리전 장애 대응**도 경험할 수 있었습니다.

CloudWatch를 통해 서버 상태를 실시간으로 모니터링하며 임계치 초과 시 자동으로 알람을 받아 **문제 발생 시 즉각적인 대응**이 가능했습니다.

S3를 활용한 정적 파일 관리와 RDS Multi-AZ로 데이터 손실 위험을 줄인 점이 가장 큰 성과였습니다.

향후 계획으로는 **멀티 클라우드 환경에서도 고가용성 인프라 설계**를 이어나가며 클라우드 자동화 역량을 더욱 강화하고자 합니다.

Project 3

컨테이너 환경 구축 및 배포 자동화 프로젝트

- 프로젝트 기간 : 2025.02.17 ~ 2025.03.14
- 구축 환경 : AWS, GitHub, Ubuntu

프로젝트 솔루션

트래픽 증가와 빠른 배포 요구를 해결하기 위해 AWS EKS 기반의 컨테이너 환경 구축 및 배포 자동화 방안을 제안하였습니다. 초기에는 온프레미스 환경에서 수동 배포로 인해 시간이 소요되고 운영 효율성이 떨어지는 문제가 발생하였습니다.

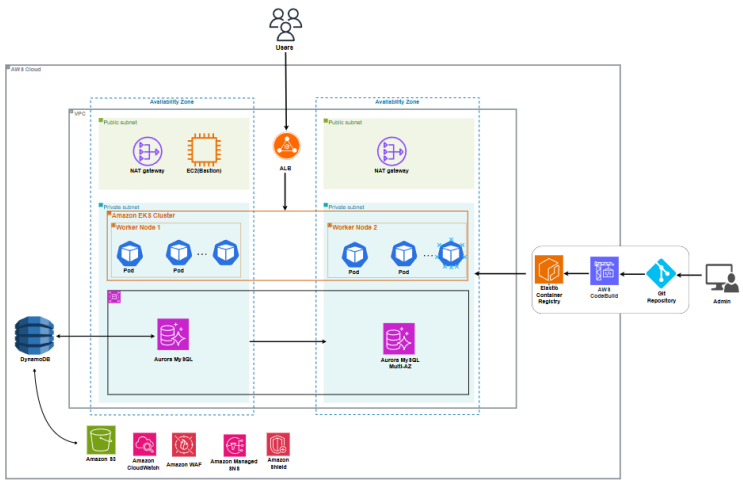
이를 해결하기 위해 Docker를 통해 애플리케이션을 컨테이너화하고 AWS EKS를 활용하여 컨테이너를 관리할 수 있도록 구성하였습니다. EKS의 Auto Scaling 기능을 통해 사용자 증가에 따른 자동 확장이 가능하도록 설정하여 고가용성을 확보할 수 있었습니다.

또한 CI/CD 파이프라인 구축을 통해 배포 자동화를 제안하였습니다. AWS CodePipeline과 CodeBuild를 사용하여 코드 변경 시 자동으로 Docker 이미지를 빌드하고 EKS 클러스터에 자동 배포되도록 설정하여 수동 작업을 최소화할 수 있었습니다.

데이터베이스는 Aurora MySQL과 DynamoDB를 도입하여 안정적인 데이터 관리를 가능하게 하였으며 CloudWatch를 통해 애플리케이션 로그와 시스템 성능을 모니터링하여 운영 안정성을 확보할 수 있도록 제안하였습니다.

기술 스택

기술 스택	설명
Docker	애플리케이션 컨테이너화
AWS EKS	컨테이너 오케스트레이션 및 관리
DB	RDS(MySQL, MariaDB 등), NoSQL(DynamoDB)
CI/CD	AWS CodePipeline을 활용한 배포 자동화
CloudWatch	로그 및 모니터링 시스템



담당 역할

담당 역할	세부 내용
IAM 정책 설정	각 사용자 생성 및 서비스 별 정책 설정
CI/CD 전체 관리	AWS Pipeline 구축 및 CodeBuild 배포
Docker 이미지 빌드 및 배포	Docker를 이용한 이미지 빌드 및 푸시
데이터베이스 관리	Aurora MySQL, DynamoDB구성 및 자동 확장 설정

성과 및 기대효과

항목	개선 전 (온프레미스)	개선 후 (클라우드 전환)
확장성	단일 서버 확장 불가능	컨테이너 기반 Auto Scaling 자동 확장
장애 대응	복구 시간 상당 소요	컨테이너 자동 복구 (Self-Healing)
배포 방식	수동 배포 (SSH 접근)	CI/CD 자동 배포 적용
운영 비용	고정 비용 부담	필요한 리소스만 사용으로 비용 절감

프로젝트 후기

1. IAM 정책 설정

- **목표** : 보안 강화를 위한 최소 권한 원칙 적용
- **결과** : 팀원 및 서비스에 적절한 IAM 정책을 설정하여 리소스에 대한 안전한 접근 보장

2. CI/CD 파이프라인 구축

- **목표** : 배포 자동화 및 효율화
- **결과** : AWS CodePipeline을 활용하여 자동화된 배포 파이프라인 구축으로 수동 배포에서 발생할 수 있는 오류를 줄이고 배포 속도 개선

3. Docker 이미지 빌드 및 푸시

- **목표** : 애플리케이션의 일관된 배포 및 관리
- **결과** : 애플리케이션을 Docker로 컨테이너화, EKS 클러스터에 배포함으로써 개발 환경과 운영 환경 간의 일관성 유지 및 안정적인 배포

4. 데이터베이스 관리

- **목표** : 안정적인 데이터 저장 및 성능 최적화
- **결과** : Aurora MySQL 및 DynamoDB에서 인덱스 설정을 통해 쿼리 성능을 개선하고 데이터베이스의 빠른 응답성과 효율적인 데이터 조회 보장

5. 느낀점

이번 프로젝트를 통해 **컨테이너 기반 인프라의 유연성과 자동화의 중요성**을 깊이 느꼈습니다.

EKS 클러스터를 통해 애플리케이션을 컨테이너화하여 **개발 환경과 운영 환경 간의 일관성**을 유지할 수 있었으며 AWS CodePipeline을 통한 CI/CD 구축으로 배포 자동화의 효율성을 실감할 수 있었습니다.

Docker 이미지 빌드와 EKS 클러스터로의 자동 배포는 코드 변경 시 빠르게 새로운 버전을 배포할 수 있는 **DevOps 자동화 환경**의 중요성을 강조해 주었습니다.

향후 계획으로는 **MSA구축**을 위한 학습을 이어나가며 더욱 효율적인 클라우드 인프라를 설계하고 운영하고자 합니다.