온프레미스 환경내에서 게임 사이트 자동화 배포 작업 보고서

프로젝트 교과 : 인프라 및 애플리케이션 자동화 프로젝트

프로젝트 부제 : *코오롱 사내 게임 사이트 배포 및 자동화*

팀원 : 배민준, 김동인, 이준호, 이우정, 이승훈

개요
- ✅ 프로젝트 목표
- ✅ 주요 구성 요소
- ✅ 자동화 흐름 요약

아키텍처 설명
- 🖥️ 논리적 아키텍처
- ✅ 1. 서비스 계층
- ✅ 2. 관리 계층
- ✅ 3. CI/CD 계층
- 🖥️ 물리적 아키텍처
- ✅ Server 1 – 네트워크 및 보안 인프라
- ✅ Server 2 – 관리 및 제어 서버
- ✅ Server 3 – 서비스 및 운영 자동화 서버
- Storage
- 네트워크 구성 요약

프로젝트 진행 활용 기술
- 네트워크 설정

게임 자동화 배포 인프라 네트워크 설정
- 1. 네트워크 구성 개요
- 3. 📁 저장소 및 워커 연결망

기본설정

📘 BIND DNS를 활용한 내부 도메인 설정 및 관리
- ✅ 1. BIND 메인 설정 파일 구성 (`/etc/named.conf`)
- ✅ 2. 도메인 데이터베이스 설정

🔒 WAF 구축 – ModSecurity with NGINX
- 1. 시스템 구성 사전 준비
- 3. ModSecurity 소스 빌드 및 설치

4. NGINX 설정 및 모듈 연동
5. 보안 규칙 설정 및 테스트
HAProxy 로드밸런서 설정 작업
1. HAProxy 설치 및 초기 설정
2. 게임 배포 및 HAProxy 연동 흐름
5. 작업 결과 및 기대 효과
⚛️ Kubernetes 설치 작업 보고서
🧱 설치 작업 요약
⚙️ 작업 상세 절차
1. 기본 시스템 설정 (모든 노드 공통) master0, worker0~2
🖥️ **마스터 노드 (master0) 초기화**
🔗 워커 노드에서 kubectl join 명령 수행 (worker0~3)
🌐 **네트워크 플러그인(Calico) 설치**
✅ 설치 검증
Ansible을 이용한 게임 자동화 배포
디렉터리 구조
설치 전 작업
작업 모듈 요약
주요 템플릿 파일 (Jinja2)
실행 결과 확인
Master-Slave 복제를 활용한 Kubernetes Pod 로그 저장
✅ 작업 목적
🏗️ 시스템 구성
🛠️ 작업 절차
FreeNAS 기반 NAS 스토리지 구축 및 Kubernetes 연동
✅ 1. FreeNAS 가상머신 생성 및 네트워크 설정
Dataset 생성하기
Kubernetes에 NFS 연동
📍 PersistentVolume(PV) 및 PersistentVolumeClaim(PVC) 설정
Ansible 연동 및 자동화 백업
Helm 차트를 활용한 Prometheus & Grafana 기반 Kubernetes 모니터링 설정
✅ Helm이란?
🛠️ 1. Helm 설치 및 저장소 등록
📡 2. Prometheus & Grafana 설치
✅ 접속 확인
📊 5. Grafana 대시보드 확인
✅ 확인 가능한 항목
✅ 1번 대시보드 – **노드 리소스 모니터링 (Node Exporter)**
🔍 주요 모니터링 항목:
✅ 2번 대시보드 – **노드 리소스 모니터링 (Node Exporter)**
✅ 3번 대시보드 – **노드 리소스 모니터링 (Node Exporter)**
✅ 4번 대시보드 – **노드 리소스 모니터링 (Node Exporter)**
✅ 5번 대시보드 – **게임 서비스(Pod) 리소스 모니터링**
🔍 주요 모니터링 항목:
✅ 6번 대시보드 – **클러스터 전체 상태 모니터링 (Kubernetes Cluster Overview)**
🔍 주요 모니터링 항목:
CI/CD 파이프라인 구축 작업
🛠️ 파이프라인 구축 절차
1. **Gitea 설치 및 프로젝트 레포지토리 생성**
2. **Jenkins 설치 및 Gitea 연동**
3. **Docker Private Registry 설정**
4. **Pipeline 구성 (Jenkinsfile)**
주요 단계:
5. **Ansible을 통한 배포 작업**
6. **SSH Key 기반 인증 설정**
구현과정
2. Jenkins Pipeline 구현 (Jenkinsfile)
🧪 테스트 및 결과 확인

```
    1. Jenkins 빌드 성공 확인
    2. Kubernetes 상태 확인
    3. HAProxy 설정 확인
    4. WAF 설정 확인
    5. 외부 접속 테스트
마무리
```

팀명 : yaml얌얌

팀원: 배민준(팀장), 김동인, 이준호, 이우정, 이승훈

팀원 세부 역할

배민준 : Kubernetes 클러스터 구축

김동인 : Ansible을 이용한 게임 배포 자동화

이승훈 : DB,모니터링 작업

이우정 : WAF,DNS 작업

이준호 : CI/CD 작업

개요

VMware 기반 온프레미스 환경에서 **Kubernetes, Ansible, Jenkins, Prometheus, Grafana** 등 다양한 오픈소스 도구를 활용하여 **게임 사이트의 배포와 운영을 자동화**하는 것을 목표로 하였습니다.

✅ 프로젝트 목표

- 사내 인프라 자원을 활용한 **자체적인 온프레미스 클라우드 환경 구성**
- **Kubernetes 기반 게임 서비스 자동 배포**
- **Ansible을 활용한 인프라 구성 및 운영 자동화**
- **CI/CD 파이프라인 구축을 통한 지속적인 배포 자동화**
- **모니터링 및 로그 관리 자동화**
- **보안 강화를 위한 WAF 및 DNS 구성**

✅ 주요 구성 요소

- **가상화 환경**: VMware Workstation 기반 VM 인프라 구성
- **컨테이너 오케스트레이션**: Kubernetes + Docker
- **자동화 도구**: Ansible, Jenkins, Gitea
- **모니터링**: Prometheus, Grafana
- **보안 및 네트워크**: WAF(ModSecurity), Bind DNS, HAProxy
- **데이터베이스 구성**: MySQL Master-Slave 복제 환경
- **스토리지**: NAS를 활용한 백업 및 로그 저장

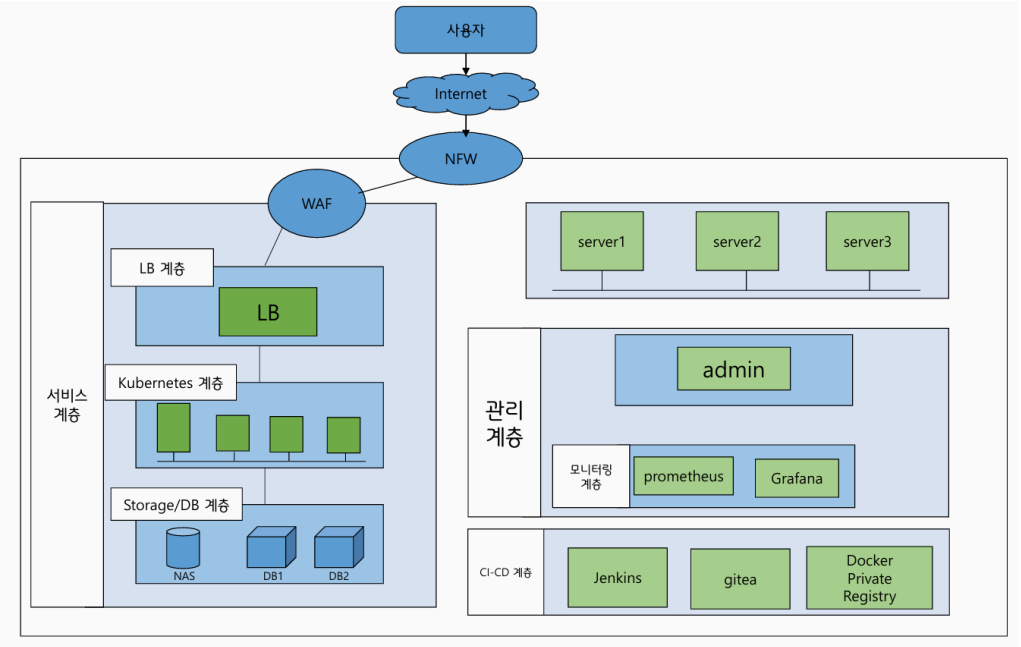
✅ 자동화 흐름 요약

1. **코드 변경 시**: Gitea → Jenkins Webhook 트리거
2. Jenkins가 소스코드 체크아웃 후 **Docker 이미지 생성 및 푸시**
3. **Ansible이 Kubernetes 리소스 생성** (Deployment, Service 등)

4. NAS에 **로그 자동 수집**, WAF/HAProxy 설정 자동화
5. **모니터링 도구로 서비스 상태 실시간 확인 및 알림 설정**

아키텍처 설명

🖥️ 논리적 아키텍처



본 프로젝트는 온프레미스 환경에서의 **게임 서비스 배포 및 운영 자동화**를 목표로, 다양한 계층으로 구성된 아키텍처를 기반으로 구축하였습니다. 전체 아키텍처는 **서비스 계층**, **관리 계층**, **CI/CD 계층**으로 나뉘며, 각 계층은 다음과 같은 구성 요소로 이루어져 있습니다.

✅ 1. 서비스 계층

- **사용자 → Internet → NFW → WAF → LB**
  - 외부 사용자는 인터넷을 통해 접속하며, **네트워크 방화벽(NFW)**과 **웹 애플리케이션 방화벽(WAF)**을 거쳐 **보안이 강화된 경로**로 요청이 전달됩니다.
  - 해당 요청은 **로드밸런서(LB)**를 통해 내부의 **Kubernetes 클러스터**로 분산됩니다.
- **LB 계층**
  - HAProxy를 활용하여 외부에서 유입된 트래픽을 내부의 Kubernetes 서비스로 라우팅합니다.
- **Kubernetes 계층**
  - 여러 워커 노드로 구성된 Kubernetes 클러스터에서 게임 애플리케이션을 컨테이너 기반으로 실행합니다.
  - 서비스 디스커버리, 오토스케일링, 롤백 등의 기능을 통해 **유연하고 안정적인 서비스 운영**이 가능합니다.
- **Storage/DB 계층**
  - **NAS**는 게임 데이터와 로그를 백업·저장하는 용도로 활용됩니다.
  - *DB1(Master)**와 **DB2(Slave)**는 MySQL 기반의 **Master-Slave 구조**로 구성되어, 실시간 데이터 복제 및 고가용성을 지원합니다.

✅ 2. 관리 계층

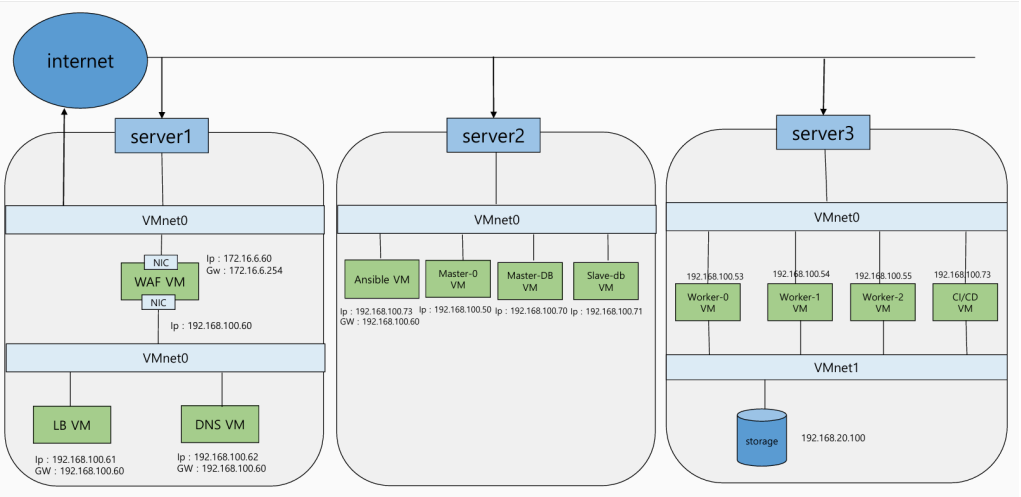
- **서버**
  - `server1`, `server2`, `server3` 등의 물리 서버는 다양한 백엔드 및 관리 기능을 수행합니다.
- **Admin**
  - 시스템 제어와 설정을 담당하는 중앙 관리자 노드로, 운영자가 직접 접근하여 인프라를 관리합니다.
- **모니터링 계층**
  - **Prometheus**는 메트릭 수집 및 저장을 담당하며, **Grafana**는 대시보드를 통해 클러스터 상태 및 성능 지표를 실시간으로 시각화합니다.

✅ 3. CI/CD 계층

- **Jenkins**
  - 소스 코드 변경이 발생하면 자동으로 빌드, 테스트, 배포까지 수행하는 **CI/CD 파이프라인**을 구성하여 배포 자동화를 실현합니다.
- **Gitea**
  - Git 기반의 자체 호스팅 코드 저장소로, Jenkins와 Webhook으로 연동되어 자동 배포 트리거 역할을 합니다.
- **Docker Private Registry**
  - 생성된 Docker 이미지를 내부에서 관리하는 사설 저장소로, 이미지의 안정적인 저장 및 배포를 지원합니다.

이와 같은 아키텍처를 통해 **외부 사용자 요청부터 내부 서비스 배포, 운영, 모니터링, 그리고 코드 변경에 따른 자동 배포**에 이르기까지 모든 과정을 **자동화된 파이프라인으로 통합**하여, **신속하고 안정적인 게임 서비스 운영 환경**을 구축할 수 있었습니다.

🖥️ 물리적 아키텍처



VMware 기반의 온프레미스 환경을 활용하여 서버 및 네트워크 자원을 가상화하였으며, 이를 기반으로 **게임 서비스의 배포 및 운영을 자동화**하였습니다. 전체 인프라는 총 세 대의 물리 서버(`server1`, `server2`, `server3`)를 중심으로, 다양한 기능을 수행하는 가상머신(VM)으로 구성되어 있습니다. 각 VM은 **VMnet0 및 VMnet1 네트워크**를 통해 상호 연결되어 있습니다.

✅ Server 1 – 네트워크 및 보안 인프라

- **WAF VM**
  - Dual NIC 구성으로, 외부망(172.16.6.60)과 내부망(192.168.100.60)을 동시에 연결하고 있습니다.
  - 외부 사용자의 요청을 필터링한 후 내부 인프라로 안전하게 전달합니다.
- **LB VM**
  - IP: 192.168.100.61 / GW: 192.168.100.60
  - HAProxy를 기반으로, 외부 트래픽을 내부 Kubernetes 서비스로 분산시키는 로드밸런서 역할을 수행합니다.
- **DNS VM**
  - IP: 192.168.100.62
  - 내부 도메인 해석과 외부 DNS 포워딩 기능을 담당하며, Bind DNS 기반으로 구성되어 있습니다.

✅ Server 2 – 관리 및 제어 서버

- **Ansible VM**
  - IP: 192.168.100.73
  - 인프라 전반에 대한 배포, 설정, 운영 자동화를 수행합니다.
- **Master-0 VM**
  - IP: 192.168.100.50
  - Kubernetes 클러스터의 control plane으로서, 클러스터의 제어 및 조율을 담당합니다.
- **Master-DB VM**
  - IP: 192.168.100.70
  - MySQL Master 노드로, 로그 및 게임 데이터의 쓰기 작업을 처리합니다.
- **Slave-DB VM**
  - IP: 192.168.100.71
  - MySQL Slave 노드로서 Master와의 실시간 복제를 통해 읽기 요청 처리 및 데이터 백업을 지원합니다.

✅ Server 3 – 서비스 및 운영 자동화 서버

- **Worker-0, Worker-1, Worker-2 VM**
  - IP: 192.168.100.53 / .54 / .55
  - Kubernetes 워커 노드로 구성되어 있으며, 게임 서비스와 관련된 다양한 애플리케이션 컨테이너를 실행합니다.
- **CI/CD VM**
  - IP: 192.168.100.73
  - Jenkins, Gitea, Docker Registry 등 CI/CD 파이프라인 구성 요소가 포함되어 있어, 자동화된 빌드 및 배포가 가능합니다.

Storage

- **NAS 스토리지**
  - IP: 192.168.20.100
  - VMnet1이라는 별도의 네트워크를 통해 연결되어 있으며, 게임 서비스의 로그 및 데이터를 안정적으로 저장하고 백업하는 역할을 수행합니다.

네트워크 구성 요약

- **VMnet0**: 모든 주요 VM이 연결된 내부 네트워크 (192.168.100.0/24)
- **VMnet1**: NAS 전용 스토리지를 위한 별도 네트워크 (192.168.20.0/24)
- **외부 인터넷 접근**: WAF VM이 이중 NIC를 통해 외부망과 내부망을 중계함으로써 외부 요청에 대한 보안을 강화하고 있습니다.

프로젝트 진행 활용 기술

분류	기술 스택	설명
인프라 구축 및 관리	VMware workstation	VM 생성
	Ansible	Kubernetes 클러스터 및 서버 환경 구성 자동화
네트워크 및 보안	WAF	웹 애플리케이션 보호
	NFW (Network Firewall)	네트워크 보안 계층
	Load Balancer	트래픽 부하 분산
Kubernetes 기반 게임 배포	Kubernetes	게임 컨테이너 오케스트레이션
	Docker	게임 컨테이너 이미지 빌드 및 배포
	Helm	kubernetes 애플리케이션 패키징 및 관리
	kubeadm/kubectl	Kubernetes 클러스터 설치 및 관리
스토리지 및 데이터 관리	NAS	컨테이너 및 게임 데이터 저장소
	PV / PVC	Kubernetes 내 데이터 지속성 관리
	DB 서버	게임 파드 기록 저장 및 관리(Master, Slave)
CI/CD 파이프라인	Jenkins	CI/CD 파이프라인 구축 및 자동화
코드 및 이미지 관리	Gitea	게임 사이트 코드 관리
	Docker private registry	게임 사이트 코드 컨테이너화
모니터링 및 운영 관리	Prometheus	메트릭 수집 및 모니터링
	Grafana	모니터링 시각화

네트워크 설정

게임 자동화 배포 사이트와 관련된 **네트워크 설정표** 입니다. 각 VM의 네트워크 인터페이스, IP 정보, 게이트웨이, DNS 서버, 검색 도메인, 연결된 내부 네트워크(VMnet) 정보를 정리하여, 전체 인프라 환경의 네트워크 구조를 한눈에 파악할 수 있도록 작성되었습니다.

게임 자동화 배포 인프라 네트워크 설정

1. 네트워크 구성 개요

- **외부 접속(WAN)**: 인터넷 → WAF VM (172.16.6.60) → 내부망 (192.168.100.0/24)
- **내부망 게이트웨이**: WAF VM (192.168.100.60)
- **내부 DNS**: DNS VM (192.168.100.62)
- **내부 검색 도메인**: `game.local`
- **두 개의 가상 네트워크 사용**

- `VMnet0`: 기본 내부망
- `VMnet1`: 스토리지 및 워커 노드 내부망 (2nd NIC)

2. 🖥️ VM별 네트워크 상세 정보

VM 이름	IP Address	Gateway	DNS 서버	검색 도메인		내부네트워크
WAF VM	172.16.6.60	172.16.6.254	192.168.100.62	game.local	eth0	VMnet0
WAF VM	192.168.100.60				eth1	VMnet0
LB VM	192.168.100.61	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
DNS VM	192.168.100.62	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
ansible VM	192.168.100.72	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
Master-db VM	192.168.100.70	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
Slave-db VM	192.168.100.71	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
master0 VM	192.168.100.50	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
worker0 VM	192.168.100.53	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
worker0 VM	192.168.20.50				eth1	VMnet1
worker1 VM	192.168.100.54	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
worker1 VM	192.168.20.51				eth1	VMnet1
worker2 VM	192.168.100.55	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0
worker2 VM	192.168.20.52				eth1	VMnet1
ci-cd VM	192.168.100.75	192.168.100.60	192.168.100.62	game.local	eth0	VMnet0

3. 📁 저장소 및 워커 연결망

- **스토리지 IP**: `192.168.20.100` (VMnet1에 연결)
- **Worker-0/1/2 VM**들은 모두 VMnet1에 보조 NIC로 연결되어 스토리지와 직접 통신 가능
- 워커들은 CI/CD 파이프라인의 결과물을 저장하거나 불러올 때 이 경로를 사용

기본설정

우선 각 VM별 네트워크 상세 정보를 확인한 후 VMware workstation에서 각 server별 VM을 생성한다.

server1

- WAF VM
- LB VM
- DNS VM

server2

- ansible VM
- master0 VM
- Master-DB VM
- Slave-db VM

server3

- worker0 VM
- worker1 VM
- worker2 VM
- ci-cd VM

📘 BIND DNS를 활용한 내부 도메인 설정 및 관리

본 프로젝트에서는 온프레미스 환경 내의 각 VM에 대해 **도메인 기반 네트워크 통신을 지원**하기 위해, **BIND DNS**를 활용한 내부 DNS 서버를 구축하고 설정하였습니다. 이를 통해 각 VM은 고정 IP 대신 **도메인 네임으로 쉽게 접근 가능**하며, 외부 DNS 포워딩도 함께 설정하여 **내부-외부 통신의 안정성**을 확보하였습니다.

✅ 1. BIND 메인 설정 파일 구성 (`/etc/named.conf`)

- **DNS 요청 허용 대역 설정**: 내부망 IP(`192.168.100.0/24`)와 외부망 IP(`172.16.6.0/24`)에 대해서만 요청 허용
- **외부 DNS 포워딩 설정**: Google Public DNS (`8.8.8.8` , `8.8.4.4`)를 통해 외부 도메인 조회 가능
- **도메인 존 설정**:
  - 정방향 도메인 `game.local`
  - 역방향 도메인 `192.168.100.0/24` , `172.16.6.0/24`

```
zone "game.local" IN {
    type master;
    file "game.local.db";
    allow-update { none; };
};

zone "100.168.192.in-addr.arpa" IN {
    type master;
    file "rev.192.168.100.db";
    allow-update { none; };
};

zone "6.16.172.in-addr.arpa" IN {
    type master;
    file "rev.172.16.6.db";
    allow-update { none; };
};
```

✅ 2. 도메인 데이터베이스 설정

📌 2.1. 정방향 조회 파일 (`/var/named/game.local.db`)

- 내부에서 사용하는 모든 VM에 대해 도메인 네임과 IP 주소를 매핑하였습니다.
- 예: `waf.game.local → 192.168.100.60` , `master0.game.local → 192.168.100.50`

```
$TTL 86400
@   IN  SOA  dns.game.local. root.game.local. (
        2024032201  ; Serial
        3600        ; Refresh
```

```
        1800        ; Retry
        604800      ; Expire
        86400 )     ; Minimum TTL

@       IN  NS      dns.game.local.
dns     IN  A       192.168.100.62
waf     IN  A       192.168.100.60
dong    IN  A       172.16.6.60
lb      IN  A       192.168.100.61
ansible IN  A       192.168.100.73
master0 IN  A       192.168.100.50
masterdb IN A       192.168.100.70
slavedb IN A        192.168.100.71
worker0 IN A        192.168.100.53
worker1 IN A        192.168.100.54
worker2 IN A        192.168.100.55
cicd    IN A        192.168.100.75
storage IN A        192.168.20.100
```

📌 2.2. 역방향 조회 파일

- **192.168.100.0/24** 대역 → `/var/named/rev.192.168.100.db`
- **172.16.6.0/24** 대역 → `/var/named/rev.172.16.6.db`

`/var/named/rev.192.168.100.db`

```
$TTL 86400
@   IN  SOA  dns.game.local. root.game.local. (
        2024032201  ; Serial
        3600        ; Refresh
        1800        ; Retry
        604800      ; Expire
        86400 )     ; Minimum TTL

@       IN  NS      dns.game.local.
60      IN  PTR     waf.game.local.
61      IN  PTR     lb.game.local.
62      IN  PTR     dns.game.local.
50      IN  PTR     master0.game.local.
70      IN  PTR     masterdb.game.local.
71      IN  PTR     slavedb.game.local.
53      IN  PTR     worker0.game.local.
54      IN  PTR     worker1.game.local.
55      IN  PTR     worker2.game.local.
75      IN  PTR     cicd.game.local.
```

`/var/named/rev.172.16.6.db`

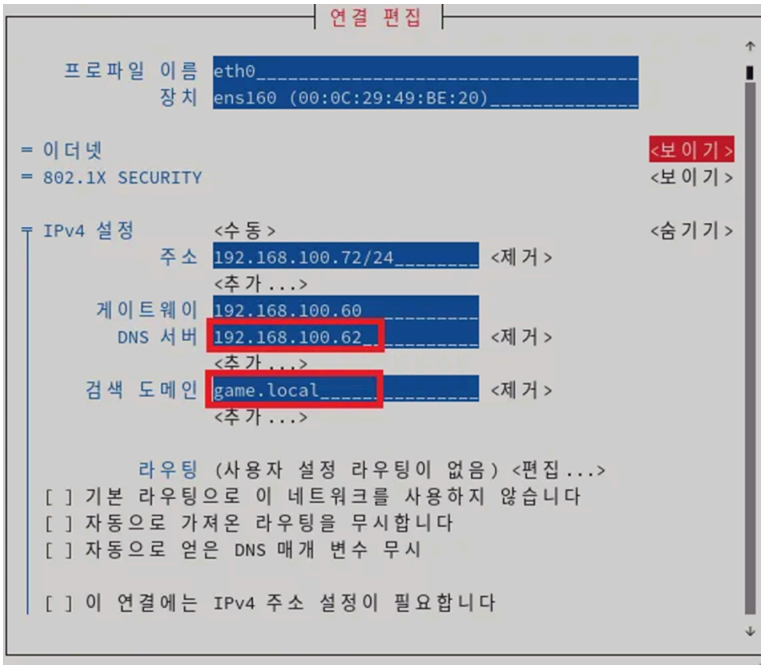
```
$TTL 86400
@   IN  SOA  dns.game.local. root.game.local. (
        2024032202  ; Serial
        3600        ; Refresh
```

```
        1800       ; Retry
        604800     ; Expire
        86400 )    ; Minimum TTL

@       IN  NS     dns.game.local.
60      IN  PTR    dong.game.local.
```

Bind DNS 서비스 적용

```
systemctl restart named
systemctl enable --now named
```

- 서비스 재시작 및 활성화



- nmtui 명령어로 모든 VM에 DNS 서버(192.168.100.62) 등록

명령어로 DNS 조회 결과 확인

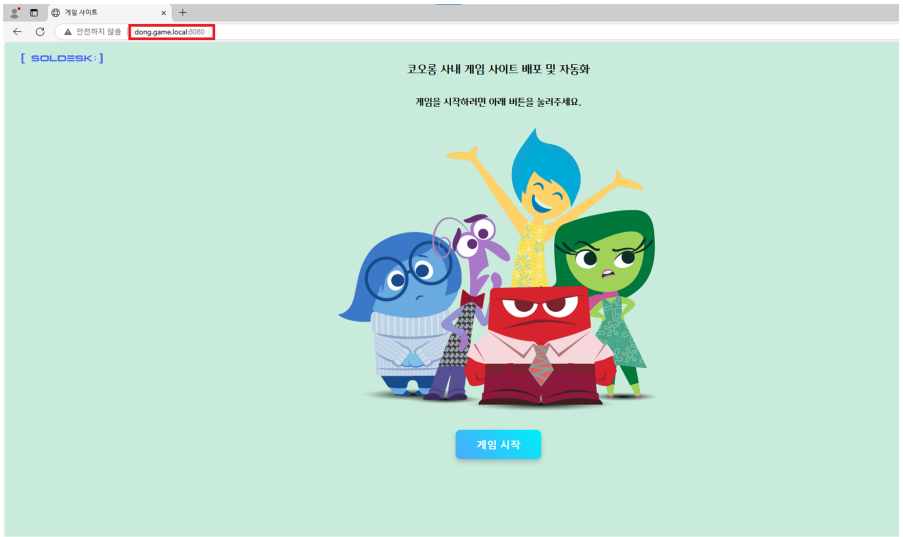
```
[root@master0 ~]# nslookup dong.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   dong.game.local
Address: 172.16.6.60

[root@master0 ~]# nslookup waf.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   waf.game.local
Address: 192.168.100.60

[root@master0 ~]# nslookup lb.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   lb.game.local
Address: 192.168.100.61

[root@master0 ~]# nslookup dns.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   dns.game.local
Address: 192.168.100.62
```

```
[root@master0 ~]# nslookup worker0.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   worker0.game.local
Address: 192.168.100.53

[root@master0 ~]# nslookup worker1.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   worker1.game.local
Address: 192.168.100.54

[root@master0 ~]# nslookup worker2.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   worker2.game.local
Address: 192.168.100.55

[root@master0 ~]# nslookup cicd.game.local
Server:         192.168.100.62
Address:        192.168.100.62#53

Name:   cicd.game.local
Address: 192.168.100.75
```

위 명령어로 각 ip에 대한 dns가 설정된 것을 확인 할 수 있습니다.

외부에서 게임 사이트 접속
내부 DNS 설정을 통해 외부에서도 아래와 같은 방식으로
도메인 기반 접속이 가능하게 되었습니다:

- 외부에서 `http://dong.game.local:8080` 로 접속



위 BindDNS 설정을 통해서 기대효과로는

- **IP 기억 불필요**: 도메인 기반 접근으로 사용자 및 운영자의 편의성 향상
- **유지보수 용이성**: VM의 IP가 변경되더라도 DNS만 수정하면 서비스 영향 없이 대응 가능
- **외부 도메인 포워딩**: 내부망에서도 외부 도메인 접근 가능 (Google DNS 활용)
- **안정적이고 체계적인 네트워크 구성 기반 마련**

이와 같이 BIND DNS 서버를 구성함으로써,
온프레미스 환경에서의 내부 네트워크 가시성, 관리 효율성, 운영 안정성을 모두 확보할 수 있었습니다.

🔒 WAF 구축 – ModSecurity with NGINX

본 작업은 온프레미스 인프라 환경에서 외부 공격으로부터 웹 서비스를 보호하기 위해, **ModSecurity**를 NGINX와 연동하여 **웹 애플리케이션 방화벽(WAF)**을 구성한 내용입니다. 해당 작업은 CentOS 기반의 WAF 서버에서 진행되었으며, 주요 보안 위협인 **SQL Injection**, **XSS**, **비정상 User-Agent** 요청을 탐지 및 차단하는 것을 목표로 하였습니다.

---

1. 시스템 구성 사전 준비

EPEL 저장소 활성화 및 NGINX 설치

```
sudo dnf install -y epel-release
sudo dnf install -y nginx
```

- EPEL 저장소는 NGINX 및 보안 관련 패키지를 설치하기 위해 필수적으로 활성화하였습니다.

개발 및 의존성 패키지 설치

```
sudo dnf groupinstall "Development Tools"
sudo dnf install -y gcc-c++ flex bison yajl yajl-devel curl-devel curl doxygen zlib-devel pcre-devel libtool g
it automake autoconf libmaxminddb libmaxminddb-devel
```

- ModSecurity 및 관련 모듈 빌드를 위한 필수 도구 및 라이브러리를 설치하였습니다.

2. ModSecurity 및 NGINX 모듈 설치

ModSecurity-nginx 모듈 다운로드 및 빌드

```
cd /usr/local/src
sudo git clone --depth 1 https://github.com/SpiderLabs/ModSecurity-nginx.git
```

설치된 NGINX 버전 확인 및 소스 다운로드

```
nginx -v
# 예: nginx/1.20.1

cd /usr/local/src
sudo wget http://nginx.org/download/nginx-1.20.1.tar.gz
sudo tar -xvzf nginx-1.20.1.tar.gz
cd nginx-1.20.1
```

Nginx 모듈 빌드 및 설치

```
sudo ./configure --with-compat --add-dynamic-module=/usr/local/src/ModSecurity-nginx
sudo make modules
sudo mkdir -p /usr/share/nginx/modules/
sudo cp objs/ngx_http_modsecurity_module.so /usr/share/nginx/modules/
```

- NGINX와 ModSecurity 연동을 위한 동적 모듈을 빌드 후 설치 경로에 복사

3. ModSecurity 소스 빌드 및 설치

```
cd /usr/local/src
sudo git clone --depth 1 https://github.com/SpiderLabs/ModSecurity
cd ModSecurity
sudo ./build.sh
sudo ./configure
sudo make -j$(nproc)
sudo make install
```

4. NGINX 설정 및 모듈 연동

모듈 로드 설정 (`/etc/nginx/nginx.conf`)

```
load_module /usr/share/nginx/modules/ngx_http_modsecurity_module.so;

http {
    modsecurity on;
    modsecurity_rules_file /etc/nginx/modsec/main.conf;

    ...
}
```

설정 적용

```
sudo nginx -t
sudo systemctl restart nginx
```

5. 보안 규칙 설정 및 테스트

보안 룰 파일 작성 (`/etc/modsecurity/modsecurity.conf`)

```
SecRuleEngine On
SecAuditEngine On
SecAuditLog /var/log/nginx/modsec_audit.log
SecDebugLog /var/log/modsec_debug.log
SecDebugLogLevel 9

# SQL Injection 룰
SecRule ARGS|REQUEST_HEADERS "@rx select|insert|drop|update" "id:1001,phase:2,deny,log,status:403,
msg:'SQL Injection Detected'"

# XSS 룰
SecRule ARGS "@rx <script>" "id:1002,phase:2,deny,log,status:403,msg:'XSS Detected'"

# 특정 User-Agent 차단
SecRule REQUEST_HEADERS:User-Agent "Minipunk" "id:1003,phase:1,deny,log,status:403,msg:'Blocked U
ser-Agent'"
```

서비스 재시작

```
sudo systemctl restart nginx
```

6. 보안 테스트 및 로그 확인

테스트 명령어

```
# SQL Injection 테스트
curl -A "Mozilla/5.0" "http://localhost/?id=1' OR '1'='1"
```

```
# XSS 공격 테스트
curl -A "Mozilla/5.0" "http://localhost/?id=<script>alert('XSS')</script>"
```

```
[root@WAF ~]# curl -A "Mozilla/5.0" "http://localhost/?id=<script>alert('XSS')</
script>"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.20.1</center>
</body>
</html>
```

```
# User-Agent 차단 테스트
curl -A "Minipunk" "http://localhost/?id=1"
```

```
[root@WAF ~]# curl -A "Minipunk" "http://localhost/?id=1"
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.20.1</center>
</body>
</html>
```

로그 확인

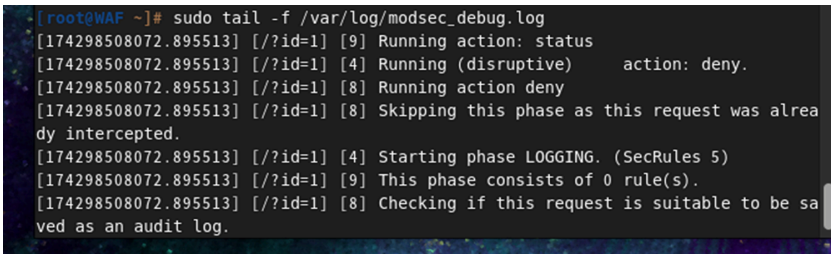
```
# Audit 로그
sudo tail -f /var/log/nginx/modsec_audit.log
```

```
[root@WAF ~]# sudo tail -f /var/log/nginx/modsec_audit.log

---qirELobE---H--
ModSecurity: Access denied with code 403 (phase 1). Matched "Operator `Rx' with
parameter `Minipunk' against variable `REQUEST_HEADERS:User-Agent' (Value: `Mini
punk' ) [file "/etc/modsecurity/modsecurity.conf"] [line "20"] [id "123457"] [re
v ""] [msg "Blocked User-Agent: Minipunk"] [data ""] [severity "0"] [ver ""] [ma
turity "0"] [accuracy "0"] [hostname "127.0.0.1"] [uri "/"] [unique_id "17429850
8072.895513"] [ref "o0,8v48,8"]
```

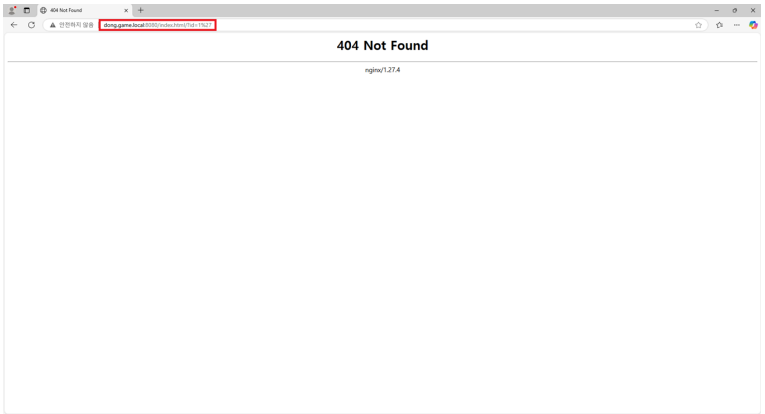
```
# 에러 로그
sudo tail -f /var/log/nginx/error.log
```

```
[root@WAF ~]# sudo tail -f /var/log/nginx/error.log
2025/03/26 19:21:28 [error] 1290#1290: *3 [client 127.0.0.1] ModSecurity: Access
 denied with code 403 (phase 2). Matched "Operator `Rx' with parameter `select|i
nsert|union|drop|update|delete|--|;|/*' against variable `ARGS_NAMES:id' (Value:
 `id' ) [file "/etc/modsecurity/modsecurity.conf"] [line "21"] [id "1002"] [rev
""] [msg "SQL Injection Attempt Detected"] [data ""] [severity "0"] [ver ""] [ma
turity "0"] [accuracy "0"] [hostname "127.0.0.1"] [uri "/"] [unique_id "17429844
8869.758015"] [ref "o0,0v9,12o0,0v37,9o0,0v59,11o0,0v79,3o0,0v6,2"], client: 127
.0.0.1, server: 172.16.6.60, request: "GET /?id=1' OR '1'='1 HTTP/1.1", host: "l
```

```
# 디버깅 로그
sudo tail -f /var/log/modsec_debug.log
```

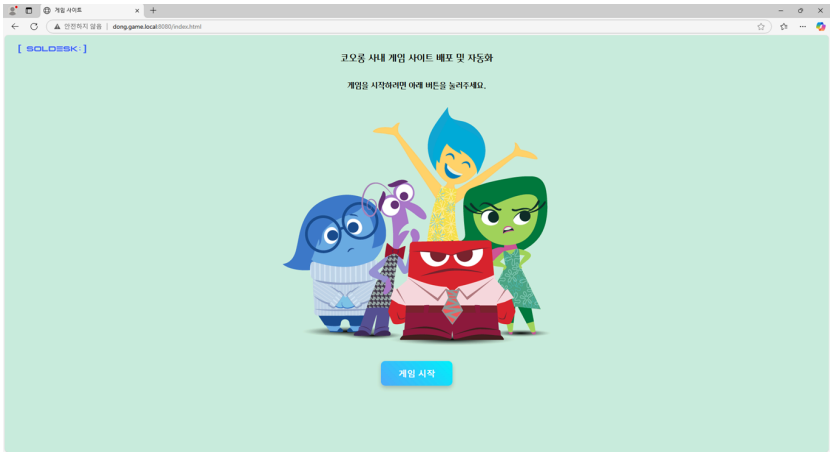


위 설정을 통해 차단 됐을 때는



404가 뜨게 되고

정상 접속 할 경우



게임 사이트가 접속됩니다.

ModSecurity는 NGINX와의 연동을 통해 **SQL Injection, XSS, 비정상 User-Agent 요청** 등 다양한 웹 공격을 실시간으로 탐지하고 차단할 수 있도록 구성되었습니다. 본 작업을 통해 구축된 WAF는 내부 서비스 보호를 위한 핵심 보안 계층으로 활용될

수 있으며, 향후 OWASP CRS 적용 등 고도화를 통해 보다 강력한 정책을 구성할 수 있습니다.

HAProxy 로드밸런서 설정 작업

본 작업은 외부 사용자의 요청을 내부 Kubernetes 클러스터의 게임 서비스에 안전하고 효율적으로 전달하기 위해, **HAProxy를 구성하여 로드밸런싱 기능을 적용**한 작업 내용입니다. 특히 HAProxy 설정은 **Ansible 자동화 도구를 통해 반복 가능하고 일관되게 적용**되도록 구성하였습니다.

---

1. HAProxy 설치 및 초기 설정

```
sudo yum install -y haproxy
```

설치 완료 후,
`/etc/haproxy/haproxy.cfg` 파일을 통해 프록시 및 백엔드 서버 설정을 구성하였습니다.

2. 게임 배포 및 HAProxy 연동 흐름

1단계: Ansible을 통한 게임 서비스 자동 배포

- Ansible Playbook을 이용하여 Kubernetes Deployment 및 Service(YAML)를 자동으로 생성하고 적용합니다.
- 각 게임 서비스는 NodePort 형식으로 외부에 포트를 개방하며, 워커 노드에서 해당 포트를 통해 접근할 수 있도록 설정됩니다.
  - 예시: `worker0:30001`, `worker1:30001`, `worker2:30001` 등

2단계: Ansible을 통한 HAProxy 설정 자동화

- 게임 서비스 배포 이후, Ansible을 통해 HAProxy 설정 파일을 자동으로 수정하고 서비스를 재시작합니다.
- 게임별로 프론트엔드와 백엔드 구성을 자동 적용하며, 서비스 상태까지 확인할 수 있는 통합 모니터링 페이지도 함께 설정합니다.

4. HAproxy 모니터링

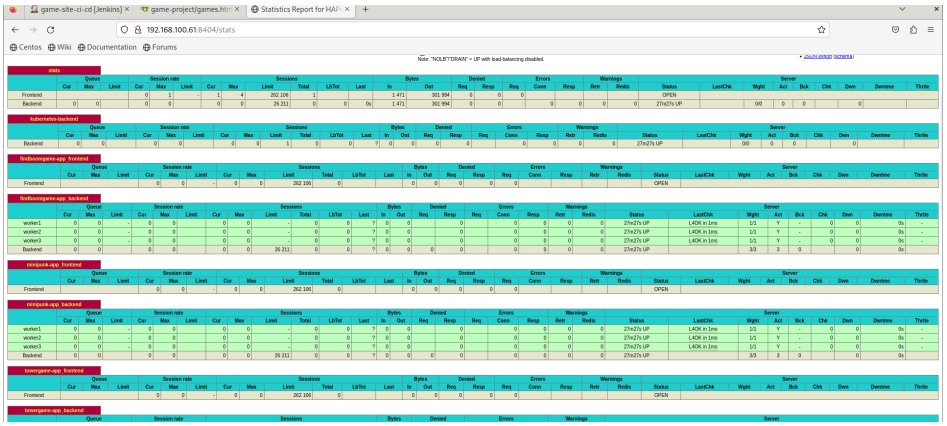
HAProxy 웹 UI를 통해 **부하분산 상태 및 서버 가용성**을 실시간으로 확인할 수 있습니다.
HAProxy 설정 (
`/etc/haproxy/haproxy.cfg`)

```
listen stats
    bind *:8404
    mode http
    stats enable
    stats uri /stats
    stats refresh 10s
    stats realm Haproxy\ Statistics
    stats auth admin:admin
```

설정 완료 후 서비스를 재시작

```
sudo systemctl restart haproxy
sudo systemctl enable haproxy
```

- 접속 주소 : `http://192.168.100.61:8404/stats`
- 로그인 정보:
  - ID: `admin`
  - PW: `admin`



웹 UI에서는 각 게임 서비스의 백엔드 상태, 응답 속도, 연결 수 등을 직관적으로 확인할 수 있습니다.

5. 작업 결과 및 기대 효과

- **게임 서비스 배포 직후, 자동으로 LB 설정이 적용**되어 사용자는 외부 주소 한 곳으로 접속 가능
- **NodePort와 연동된 로드밸런싱 구성**으로 서버 간 트래픽 분산 및 고가용성 확보
- **Ansible 자동화를 통해 반복 배포 시에도 일관된 LB 설정 가능**
- 통합 **HAProxy 모니터링 페이지**로 실시간 상태 확인 및 장애 대응 용이

☸️ Kubernetes 설치 작업 보고서

본 프로젝트에서는 게임 서비스를 안정적으로 배포하고 유연하게 확장할 수 있는 플랫폼으로 **Kubernetes**를 선택하여 클러스터를 구성하였습니다. Kubernetes는 컨테이너 기반 애플리케이션의 자동 배포, 확장, 운영을 지원하는 오픈소스 플랫폼으로, 온프레미스 환경에서도 효율적인 마이크로서비스 아키텍처 구현이 가능합니다.

🧰 설치 작업 요약

- **설치 대상**: Master 노드 1대(`master0`), 워커 노드 3대(`worker0~2`)
- **컨테이너 런타임**: containerd
- **네트워크 플러그인**: Calico
- **클러스터 구성 도구**: kubeadm

⚙️ 작업 상세 절차

1. 기본 시스템 설정 (모든 노드 공통) master0, worker0~2

Swap 비활성화, **SELinux Permissive**, **Firewalld 비활성화**

- **k8s 관련 커널 모듈 로드 및 sysctl 네트워크 설정**
- **Containerd 설치 및 SystemdCgroup 설정 적용**

1. Swap 비활성화

```
# Swap 비활성화
sudo swapoff -a

# 부팅 시 swap이 다시 활성화되지 않도록 설정
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

2. 네트워크 모듈 및 sysctl 설정

K8s 설치에 필요한 네트워크 모듈을 로드하고, 필요한 sysctl 파라미터를 설정합니다.

```
# 네트워크 모듈 로드
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl 파라미터 설정
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# sysctl 파라미터 적용
sudo sysctl --system
```

3. SELinux 비활성화

```
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

4. firewalld 비활성화

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld
```

5. Containerd 설치 및 설정

```
# containerd 설치
# https://docs.docker.com/engine/install/centos/
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install -y containerd.io
sudo systemctl enable --now containerd

# 컨테이너 런타임 설정
# https://kubernetes.io/docs/setup/production-environment/container-runtimes/
containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
sudo systemctl restart containerd
```

쿠버네티스 도구 설치

```
# https://pwittrock.github.io/docs/setup/independent/install-kubeadm/
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.32/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.32/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
systemctl enable --now kubelet
```

🖥️ 마스터 노드 (master0) 초기화

마스터노드 1개(
`master0)` 사용하여 **Kubernetes 클러스터**를 구성

```
sudo kubeadm init --apiserver-advertise-address 192.168.100.50
```

코드 실행후

```
# non-root 사용자라면 다음 명령을 수행한다.
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# root 사용자라면 다음 명령을 수행한다.
# echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> $HOME/.bashrc
# source $HOME/.bashrc
```

- `kubeadm init` 명령어를 통해 Kubernetes 클러스터의 컨트롤 플레인을 구성하였습니다.
- 일반 사용자 접근을 위해 `admin.conf` 를 `.kube/config` 로 복사하여 인증 설정을 완료하였습니다.

🔗 워커 노드에서 kubectl join 명령 수행 (worker0~3)

```
# (명령어 형식)
# kubeadm join --token <token> <control-plane-host>:<control-plane-port>
#   --discovery-token-ca-cert-hash sha256:<hash>
# 예) kubeadm join 명령 예
# kubeadm join 192.168.10.10:6443 --token vb2n8m.5gnqo1mpbng8zdkh \
#   --discovery-token-ca-cert-hash \
#   sha256:03e1f0a96ab67f39c6769465b2200104a5da8cee83692805e8346026f9d7c462
#
# --token:
#    ⇒ kubeadm token list | tail -1 | awk '{print $1}'
# --discovery-token-ca-cert-hash:
#    ⇒ openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt \
#       | openssl rsa -pubin -outform der 2>/dev/null \
#       | openssl dgst -sha256 -hex | sed 's/^.* //'
#
# (주의) 다음은 실행 예제이다.
# master 노드에서 kubeadm init 명령 수행시 나온 출력 결과를 참고해야 실행해야 한다.
kubeadm join 192.168.100.50:6443 --token vb2n8m.5gnqo1mpbng8zdkh \
  --discovery-token-ca-cert-hash \
  sha256:03e1f0a96ab67f39c6769465b2200104a5da8cee83692805e8346026f9d7c462
```

🌐 네트워크 플러그인(Calico) 설치

- 마스터 노드에서 Calico YAML 매니페스트를 적용하여 네트워크 구성

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

- `kubectl get pod -A` 명령어로 모든 네트워크 구성 요소가 정상적으로 실행 중인 것을 확인함

✅ 설치 검증

```
kubectl get nodes
```

```
[root@master0 ~]# kubectl get nodes
NAME                  STATUS   ROLES           AGE   VERSION
master0.example.com   Ready    control-plane   13d   v1.32.2
worker0.example.com   Ready    <none>          13d   v1.32.3
worker1.example.com   Ready    <none>          13d   v1.32.3
worker2.example.com   Ready    <none>          13d   v1.32.3
```

- 모든 노드가 `Ready` 상태로 정상적으로 클러스터에 등록됨

```
kubectl get pods -A
```

```
NAME                                       READY   STATUS    RESTARTS          AGE
calico-kube-controllers-7498b9bb4c-rpt6t   1/1     Running   927 (3d21h ago)   13d
calico-node-4pgqb                          0/1     Running   14 (3h43m ago)    13d
calico-node-bhk8f                          1/1     Running   8 (3h38m ago)     13d
calico-node-jd8s7                          1/1     Running   8 (3h38m ago)     13d
calico-node-t88fb                          1/1     Running   8 (3h38m ago)     13d
coredns-668d6bf9bc-2t8c9                   1/1     Running   11 (3h43m ago)    13d
coredns-668d6bf9bc-g6r22                   1/1     Running   7 (3h38m ago)     13d
etcd-master0.example.com                   1/1     Running   19 (3h43m ago)    13d
kube-apiserver-master0.example.com         1/1     Running   15 (3h43m ago)    13d
kube-controller-manager-master0.example.com   1/1     Running   15 (3h43m ago)    13d
kube-proxy-4qww9                           1/1     Running   8 (3h38m ago)     13d
kube-proxy-fqqn6                           1/1     Running   8 (3h38m ago)     13d
kube-proxy-mdrgj                           1/1     Running   8 (3h38m ago)     13d
kube-proxy-njfg6                           1/1     Running   14 (3h43m ago)    13d
kube-scheduler-master0.example.com         1/1     Running   15 (3h43m ago)    13d
```

Kubernetes는 본 프로젝트에서 요구하는 **게임 서비스의 유연한 배포, 스케일링, 모니터링, 운영 자동화**를 충족할 수 있는 최적의 솔루션이었습니다. 설치 이후에는 Ansible, Jenkins, Helm 등을 연동하여 CI/CD, 모니터링, 로드밸런싱 등 다양한 운영 자동화 기능을 성공적으로 구성할 수 있었습니다.

이를 통해 복잡하고 반복적인 운영 작업을 자동화하고, 향후 신규 게임 서비스 추가 시에도 **신속하고 일관된 배포가 가능한 인프라 환경**을 완성하였습니다.

Ansible을 이용한 게임 자동화 배포

본 프로젝트에서는 게임 서비스의 배포 및 인프라 운영 자동화를 위해
Ansible을 도입하였습니다. Ansible은 에이전트 없이 SSH를 기반으로 작동하는 경량화된 자동화 도구로, 다음과 같은 이유로 본 프로젝트에 적합한 선택이었습니다.

Ansible은 Kubernetes, MySQL, DNS, WAF 등 다양한 시스템과의 연동이 가능하며, 본 프로젝트에서는 다음을 자동화하여 모듈화하였습니다:

- Kubernetes Deployment 및 Service YAML 자동 생성 및 적용
- NAS 로그 수집 및 게임 데이터 백업
- HAProxy, WAF 포트 및 정책 설정
- NodePort 기반 외부 접속 경로 설정

본 작업은 VMware 환경 server2(Ansible VM)에서 진행됩니다.

디렉터리 구조

```
[root@main ~/ansible]# tree ansible_module_game_project/
ansible_module_game_project/
├── README.md
└── game_ansible
    ├── ansible.cfg
    ├── group_vars
    │   └── all.yml
    ├── host_vars
    │   ├── master.yml
    │   └── worker.yml
    ├── inventory
    │   └── hosts.ini
    ├── playbook.yml
    └── roles
```

```
        ├── haproxy
        │   ├── tasks
        │   │   └── main.yml
        │   └── templates
        │       └── haproxy.cfg.j2
        ├── k8s_deploy
        │   ├── tasks
        │   │   └── main.yml
        │   └── templates
        │       ├── deployment.yaml.j2
        │       ├── ingress.yaml.j2
        │       └── service.yaml.j2
        ├── nas
        │   └── tasks
        │       └── main.yml
        └── waf
            └── tasks
                └── main.yml
```

설치 전 작업

vi /etc/hosts

```
#
# WAF
#
192.168.100.60 WAF

#
# LB
#
192.168.100.61 LB

#
# DNS
#
192.168.100.62 DNS


#
# Kubernetes
#
192.168.100.50  master0.example.com     master0
192.168.100.53  worekr0.example.com     worker0
192.168.100.54  worker1.example.com     worker1
192.168.100.55  worker2.example.com     worker2
```

(ansible) private/public key 생성

ssh-keygen

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa): <ENTER>
Enter passphrase (empty for no passphrase): <ENTER>
Enter same passphrase again: <ENTER>
Your identification has been saved in /home/ansible/.ssh/id_rsa.
Your public key has been saved in /home/ansible/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:cP2+IlybAGI/LpkZqAHDDbwcxSAkRM3307izfijAK/Y ansible@ansible.example.com
The key's randomart image is:
+---[RSA 3072]----+
|*=++.           |
|..+o..   .      |
| o =. o + .     |
|  * .  *o...    |
|  .o   oSo ..   |
|   o. .o. o.. . |
|    oo  +* o.+ o|
| o ... o=.. +.o |
|. oE  o... ..   |
+----[SHA256]-----+
```

```
 ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.100.60
→ 출력내용 생략(암호: centos)
 ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.100.61
→ 출력내용 생략(암호: centos)
 ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.100.50
→ 출력내용 생략(암호: centos)
 ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.100.53
→ 출력내용 생략(암호: centos)
 ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.100.54
→ 출력내용 생략(암호: centos)
 ssh-copy-id -i ~/.ssh/id_rsa.pub 192.168.100.55
→ 출력내용 생략(암호: centos)
```

Ansible 설치

```
sudo yum install epel-release -y
sudo yum install yum-utils -y
sudo yum list installed 'python*'
sudo yum install ansible ansible-doc -y
```

`mkdir -p ansible/module_game_project/game_ansible/ && ansible/ansible_module_game_project/game_ansible`

이 디렉터리 안에 앤서블 관련 파일들을 넣을 거다.

작업 코드 내용은 아래 깃허브 사이트를 참고한다.

https://github.com/Dongin-Kim-ui/Ansible_Module_game_project

작업 모듈 요약

역할(Role)	설명
k8s_deploy	Deployment, Service, Ingress 리소스를 생성하여 게임 Pod 배포
nas	실행 중인 Pod에서 로그 수집 및 /app/data 디렉터리 백업
haproxy	HAProxy 설정 자동화 및 라운드로빈 방식 부하 분산 구성
waf	DNAT, MASQUERADE NAT 규칙 설정 및 방화벽 포트 자동 오픈

주요 템플릿 파일 (Jinja2)

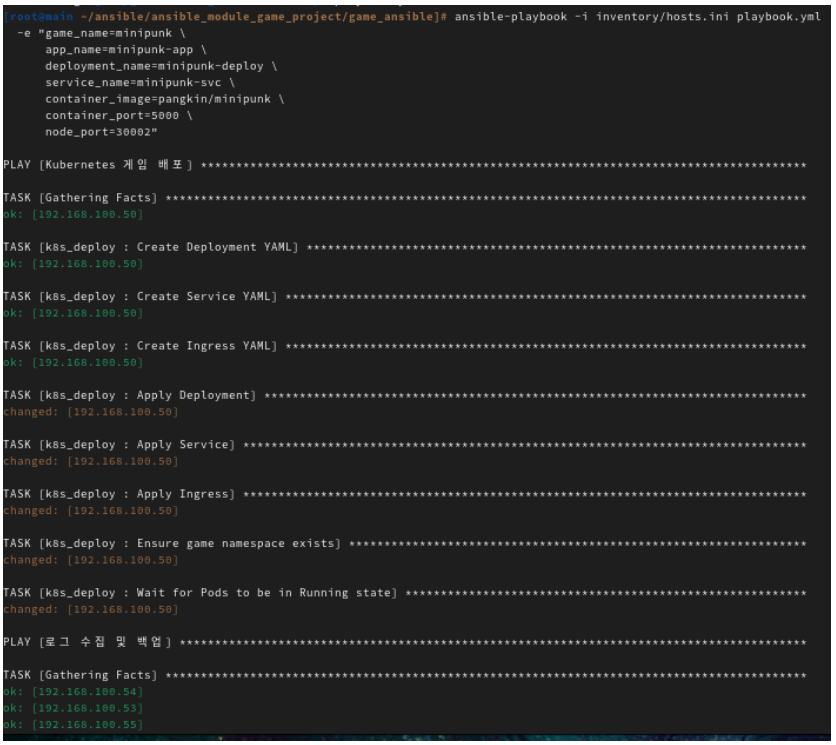
템플릿 파일	설명
deployment.yaml.j2	게임 Pod 정의 및 NAS 볼륨 마운트
service.yaml.j2	NodePort로 외부 접속 가능하도록 설정
ingress.yaml.j2	도메인 기반 Ingress 접근 허용 설정
haproxy.cfg.j2	프론트엔드/백엔드 자동 구성 (worker 연결 포함)

실행 결과 확인

실행 예시 (변수 직접 지정 방식) 위 코드로 다른 게임들도 배포 자동화를 할 수 있습니다.

```
---------------------------------------------------------------------
ansible-playbook ansible-game-playbook.yaml \
-e "game_name=findboomgame \
app_name=findboomgame-app \
deployment_name=findboomgame-deploy \
service_name=findboomgame-svc \
container_image=klm6280/findboomgame \
container_port=3000 \
node_port=30001"

---------------------------------------------------------------------
ansible-playbook ansible-game-playbook.yaml -e "game_name=minipunk \
app_name=minipunk-app \
deployment_name=minipunk-deploy \
service_name=minipunk-svc \
container_image=pangkin/minipunk \
container_port=5000 \
node_port=30002"

---------------------------------------------------------------------

ansible-playbook ansible-game-playbook.yaml -e "game_name=towergame \
app_name=towergame-app \
deployment_name=towergame-deploy \
service_name=towergame-svc \
```

```
container_image=sdylevelup/tg \
container_port=8082 \
node_port=30003"

-----------------------------------------------------------------

ansible-playbook ansible-game-playbook.yaml -e "game_name=clumsy-bird \
app_name=clumsy-bird-app \
deployment_name=clumsy-bird-deploy \
service_name=clumsy-bird-svc \
container_image=pangkin/clumsy-bird \
container_port=8001 \
node_port=30004"

-----------------------------------------------------------------
```



```
[root@main ~/ansible/ansible_module_game_project/game_ansible]# ansible-playbook -i inventory/hosts.ini playbook.yml
  -e "game_name=minipunk \
      app_name=minipunk-app \
      deployment_name=minipunk-deploy \
      service_name=minipunk-svc \
      container_image=pangkin/minipunk \
      container_port=5000 \
      node_port=30002"

PLAY [Kubernetes 게임 배포] ***********************************************************************************

TASK [Gathering Facts] ****************************************************************************************
ok: [192.168.100.50]

TASK [k8s_deploy : Create Deployment YAML] ********************************************************************
ok: [192.168.100.50]

TASK [k8s_deploy : Create Service YAML] ***********************************************************************
ok: [192.168.100.50]

TASK [k8s_deploy : Create Ingress YAML] ***********************************************************************
ok: [192.168.100.50]

TASK [k8s_deploy : Apply Deployment] **************************************************************************
changed: [192.168.100.50]

TASK [k8s_deploy : Apply Service] *****************************************************************************
changed: [192.168.100.50]

TASK [k8s_deploy : Apply Ingress] *****************************************************************************
changed: [192.168.100.50]

TASK [k8s_deploy : Ensure game namespace exists] **************************************************************
changed: [192.168.100.50]

TASK [k8s_deploy : Wait for Pods to be in Running state] ******************************************************
changed: [192.168.100.50]

PLAY [로그 수집 및 백업] **************************************************************************************

TASK [Gathering Facts] ****************************************************************************************
ok: [192.168.100.54]
ok: [192.168.100.53]
ok: [192.168.100.55]
```

```
TASK [nas : Ensure log directory exists] ********************************************************************
ok: [192.168.100.53]
ok: [192.168.100.54]
ok: [192.168.100.55]

TASK [nas : Collect logs from pods] *************************************************************************
changed: [192.168.100.53]
changed: [192.168.100.54]
changed: [192.168.100.55]

TASK [nas : Ensure backup directory exists] *****************************************************************
ok: [192.168.100.53]
ok: [192.168.100.54]
ok: [192.168.100.55]

TASK [nas : Archive game data] ******************************************************************************
changed: [192.168.100.53]
changed: [192.168.100.54]
changed: [192.168.100.55]

PLAY [HAProxy 설정] *****************************************************************************************

TASK [Gathering Facts] **************************************************************************************
ok: [192.168.100.61]

TASK [haproxy : Install HAProxy] ****************************************************************************
ok: [192.168.100.61]

TASK [haproxy : Configure HAProxy] **************************************************************************
ok: [192.168.100.61]

TASK [haproxy : Open HAProxy port] **************************************************************************
changed: [192.168.100.61]

PLAY [WAF 설정] *********************************************************************************************

TASK [Gathering Facts] **************************************************************************************
ok: [192.168.100.60]

TASK [waf : Allow external port] ****************************************************************************
changed: [192.168.100.60]

TASK [waf : Add DNAT rule] **********************************************************************************
changed: [192.168.100.60]

TASK [waf : Add MASQUERADE rule] ****************************************************************************
changed: [192.168.100.60]
```

```
TASK [waf : Enable IP forwarding] ***************************************************************************
ok: [192.168.100.60]

PLAY RECAP **************************************************************************************************
192.168.100.50             : ok=9    changed=5    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.100.53             : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.100.54             : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.100.55             : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.100.60             : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.168.100.61             : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

위와 같이 앤서블을 활용하여 게임 자동화 배포를 모듈화 시킨 실행 화면입니다.

아래 화면이 게임 배포 및 LB 설정 waf에서 외부로 나가게 하는 설정을 마친뒤 MINIPUNK 게임이 실행됩니다.

http://172.16.6.60:5000

Master-Slave 복제를 활용한 Kubernetes Pod 로그 저장

본 작업은 Kubernetes 클러스터 내에서 생성되는
Pod 로그를 안정적으로 수집하고 보관하기 위한 목적으로, **MySQL의 Master-Slave 복제 구조**를 활용하여 로그 저장 시스템을 구축한 사례입니다.
로그 데이터는
`Master DB`를 통해 기록되며, 실시간 복제 구조를 통해 `Slave DB`에서도 동일한 데이터를 조회할 수 있어 **고가용성과 장애 복구 대응력**을 동시에 확보할 수 있습니다.

✅ 작업 목적

- Kubernetes Pod 로그를 주기적으로 수집하여 **MySQL DB에 저장**
- Master-Slave 복제를 통해 **로그 데이터를 실시간 이중화**
- Master 장애 시에도 **Slave를 통한 로그 열람 가능**
- 향후 **모니터링, 분석, 감사 등의 기반 데이터**로 활용 가능

🗄️ 시스템 구성

역할	호스트명	IP 주소	설명
Master DB	`masterdb`	`192.168.100.70`	로그 저장 (쓰기 전용)
Slave DB	`slavedb`	`192.168.100.71`	로그 복제본 (읽기 전용)
Kubernetes Master	`master0`	`192.168.100.50`	로그 수집 스크립트 실행

🛠️ 작업 절차

1. **Master DB (192.168.100.70) 설정**

```
sudo dnf install -y mysql-server  # MySQL 서버 설치
sudo systemctl enable --now mysqld  # MySQL 자동 실행 설정 및 시작
```

2. MySQL 설정 변경

`/etc/my.cnf`

```
[mysqld]
server-id=1  # Master 서버 ID
log_bin=/var/log/mysql/mysql-bin.log  # 바이너리 로그 (Slave와 동기화)
binlog_do_db=logs_db  # 복제할 DB
bind-address = 0.0.0.0
```

mysql 재시작

```
sudo systemctl restart mysqld
```

3. Master에서 복제 계정 생성

```
mysql -u root -p
Enter 누르면 로그인 됨.
```

📌 Slave에서 접근할 계정 생성

```
CREATE USER 'replica'@'192.168.100.71' IDENTIFIED WITH mysql_native_password BY 'Soldesk1.';
GRANT REPLICATION SLAVE ON *.* TO 'replica'@'192.168.100.71';
FLUSH PRIVILEGES;
```

📌 Master 상태 확인

```
SHOW MASTER STATUS;
```

```
mysql> SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+-------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+---------------+----------+--------------+------------------+-------------------+
| binlog.000002 |    31772 |              |                  |                   |
+---------------+----------+--------------+------------------+-------------------+
1 row in set (0.00 sec)
```

✅ `File: mysql-bin.000002`, `Position: 31772`와 같은 결과를 기억 (Slave 설정 시 필요)

192.168.100.50(kubernetes master node)에서 접속 가능하게 설정

```
CREATE USER 'root'@'192.168.100.50' IDENTIFIED WITH mysql_native_password BY 'Soldesk1.';
GRANT ALL PRIVILEGES ON *.* TO 'root'@'192.168.100.50' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

```
SELECT host, user FROM mysql.user;
```

```
mysql> SELECT host, user FROM mysql.user;
+----------------+------------------+
| host           | user             |
+----------------+------------------+
| 192.168.100.50 | root             |
| 192.168.100.71 | replica          |
| localhost      | mysql.infoschema |
| localhost      | mysql.session    |
| localhost      | mysql.sys        |
| localhost      | root             |
+----------------+------------------+
6 rows in set (0.00 sec)
```

위 명령어 수행 했을 때 user에 `root`와 `replica`가 있어야 합니다.

3️⃣ Slave DB (192.168.100.71) 설정

1. MySQL 설치

```
sudo dnf install -y mysql-server
sudo systemctl enable --now mysqld
```

2. MySQL 설정 변경

`/etc/my.cnf`

```
[mysqld]
server-id=2  # Slave 서버 ID
relay_log=/var/log/mysql/mysql-relay-bin.log  # 복제 로그
read_only=1  # Slave는 읽기 전용
```

mysql 재시작

```
sudo systemctl restart mysqld
```

3. Slave에서 Master와 연결

```
mysql -u root -p
```

Master와 연결 설정

```
CHANGE MASTER TO
MASTER_HOST='192.168.100.70',
MASTER_USER='replica',
MASTER_PASSWORD='Soldesk1.',
MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS=31772;
GET_MASTER_PUBLIC_KEY=1;
```

Slave 시작 (복제 시작)

```
START SLAVE;
SHOW SLAVE STATUS\G;          #  << 복제 상태 확인
```

혹시나 에러 날시 SQL 에러 스킵하고 다시 복제 시작

```
STOP SLAVE;
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
SHOW SLAVE STATUS\G;
```

```
mysql> SHOW SLAVE STATUS\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for source to send event
                  Master_Host: 192.168.100.70
                  Master_User: replica
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000002
          Read_Master_Log_Pos: 29835
               Relay_Log_File: mysql-relay-bin.000005
                Relay_Log_Pos: 30051
        Relay_Master_Log_File: mysql-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

`Slave_IO_Running: Yes`, `Slave_SQL_Running: Yes`이면 정상적으로 동작 중!

MASTER VM 설정 작업 (192.168.100.50)

`192.168.100.50`에서 MySQL 접근 테스트

Kubernetes Master Node(`192.168.100.50`)에서 실행

```
mysql -h 192.168.100.70 -u root -p
```

password : Soldesk1.

정상 접속되면 이상 없이 정상 작업중 !

Kubernetes 로그 자동 저장 스크립트

```
#!/bin/bash

# 로그를 저장할 대상 네임스페이스
NAMESPACE="game"

# MySQL 접속 정보
DB_HOST="192.168.100.70"
DB_USER="root"
DB_PASS="Soldesk1."
DB_NAME="logs_db"

# 로그 저장 경로
LOG_FILE="/var/log/k8s_pod_logs.log"

echo "[$(date)] 로그 수집 시작" >> $LOG_FILE

# 네임스페이스 내 모든 게임 Deployment 목록 가져오기
DEPLOYMENTS=$(kubectl get deploy -n $NAMESPACE -o jsonpath='{.items[*].metadata.name}')

for DEPLOYMENT in $DEPLOYMENTS; do
    APP_LABEL=$(kubectl get deploy $DEPLOYMENT -n $NAMESPACE -o jsonpath='{.spec.selector.match
Labels.app}')
```

```
    echo "[$(date)] Deployment: $DEPLOYMENT (app=$APP_LABEL)" >> $LOG_FILE

    # Deployment에 속한 모든 Pod 목록 가져오기
    POD_NAMES=$(kubectl get pods -n $NAMESPACE -l app=$APP_LABEL -o jsonpath='{.items[*].metadat
a.name}')

    for POD in $POD_NAMES; do
        echo "[$(date)] 처리 중인 Pod: $POD" >> $LOG_FILE

        # Pod 로그 가져오기 (300자까지만 저장하여 MySQL 보기 편하게 함)
        RAW_LOG_TEXT=$(kubectl logs $POD -n $NAMESPACE 2>> $LOG_FILE | head -c 300)

        # MySQL에서 사용할 수 있도록 로그 데이터 JSON 변환 (줄바꿈 등 제거)
        LOG_TEXT=$(echo "$RAW_LOG_TEXT" | sed "s/'/''/g" | jq -R -s '.')

        if [[ ! -z "$LOG_TEXT" ]]; then
            mysql --host="$DB_HOST" --user="$DB_USER" --password="$DB_PASS" --database="$DB_NAM
E" --execute="
            INSERT INTO pod_logs (pod_name, namespace, log_text) VALUES
            ('$POD', '$NAMESPACE', $LOG_TEXT);
            " 2>> $LOG_FILE
        fi
    done
done

echo "[$(date)] 로그 수집 완료" >> $LOG_FILE
```

스크립트 실행 권한 부여

```
chmod +x ~/log_to_mysql.sh
```

주기적으로 실행하도록 cron에 추가

```
crontab -e
```

다음 줄 추가 (30분마다 실행)

```
*/30 * * * * /bin/bash ~/log_to_mysql.sh
```

Slave에서 로그 조회

✔ Slave에서 실행하면 Master에서 저장된 로그를 확인 가능

```
show databases;
use logs_db;

SELECT * FROM pod_logs ORDER BY created_at DESC LIMIT 5\G; 실행 했을 때
```

```
mysql> SELECT * FROM pod_logs ORDER BY created_at DESC LIMIT 5\G;
*************************** 1. row ***************************
        id: 10
  pod_name: minipunk-deploy-7cfbf47857-75wwp
 namespace: game
  log_text:

created_at: 2025-03-14 00:42:18
*************************** 2. row ***************************
        id: 8
  pod_name: findboomgame-deploy-6b4b64b994-dwxcq
 namespace: game
  log_text:
> minesweeper@0.1.0 start
> react-scripts start

Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
(node:26) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning

created_at: 2025-03-14 00:42:18
```

현재 배포중인 pod logs 기록을 볼 수 있음. Pod Running중인 logs 기록을 30분에 한번씩 확인 가능합니다.
✔️ Master가 다운되더라도
Slave에서 로그 조회 가능 🚀

Kubernetes 기반의 환경에서는 각 Pod의 수명이 짧고 로그가 휘발성인 특성이 있습니다. 이를 해결하기 위해 MySQL의 Master-Slave 복제 기능을 이용해 로그를 안전하게 저장하고, **복구 가능한 구조로** 설계했습니다.

- **장애 대비**: Master 다운 시에도 로그 열람 가능
- **신뢰성 향상**: 실시간 이중화로 데이터 유실 최소화
- **운영 효율성**: 자동 수집 + 주기적 저장 + SQL 조회 지원

FreeNAS 기반 NAS 스토리지 구축 및 Kubernetes 연동

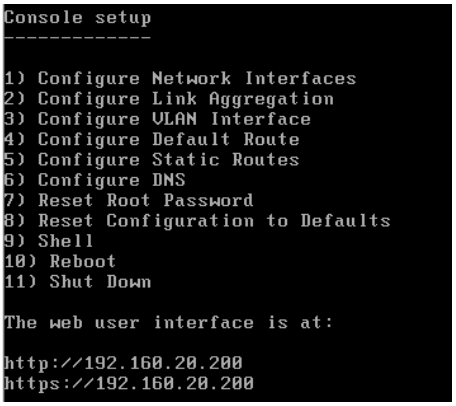
본 작업은 Kubernetes 기반 게임 서비스에서 생성되는 **데이터 및 로그를 안정적으로 저장**하고 **백업 체계를 구성**하기 위한 목적으로 FreeNAS를 활용한 NAS 스토리지를 구축하고, Kubernetes 클러스터와 NFS 프로토콜로 연동하는 내용을 포함합니다.

FreeNAS는 ZFS 기반의 오픈소스 NAS 운영체제로, 손쉬운 웹 기반 설정과 안정적인 NFS 공유를 제공하여 온프레미스 환경에 적합합니다.

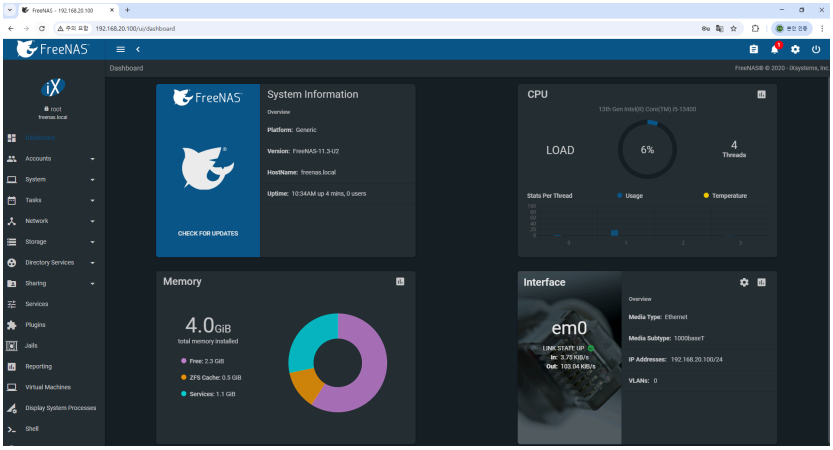
✅ 1. FreeNAS 가상머신 생성 및 네트워크 설정

- 강의자료에서 제공된 **FreeNAS 이미지 파일**을 VMware에 등록하여 새로운 가상머신 생성
- NAS 네트워크 설정:
  - IP 주소: `192.168.20.100`
  - 서브넷: `255.255.255.0`
  - DHCP 비활성화, IPv4 수동 설정
- 웹 브라우저에서 `http://192.168.20.100` 접속 시 NAS 관리 페이지에 접근 가능

```
■ Configure Network Interface
Enter an option from 1-14: 1
Select an interface (q to quit): 1
Reset network configuration? (y/n) y
Enter an option from 1-14: 1
Select an interface (q to quit): 1
Reset network configuration? (y/n) n
Configure interface for DHCP? (y/n) n
Configure IPv4? (y/n) y
Interface name: em0
IPv4 Address: 192.168.20.100
IPv4 Netmask: 255.255.255.0
Configure IPv6? (y/n) n
```



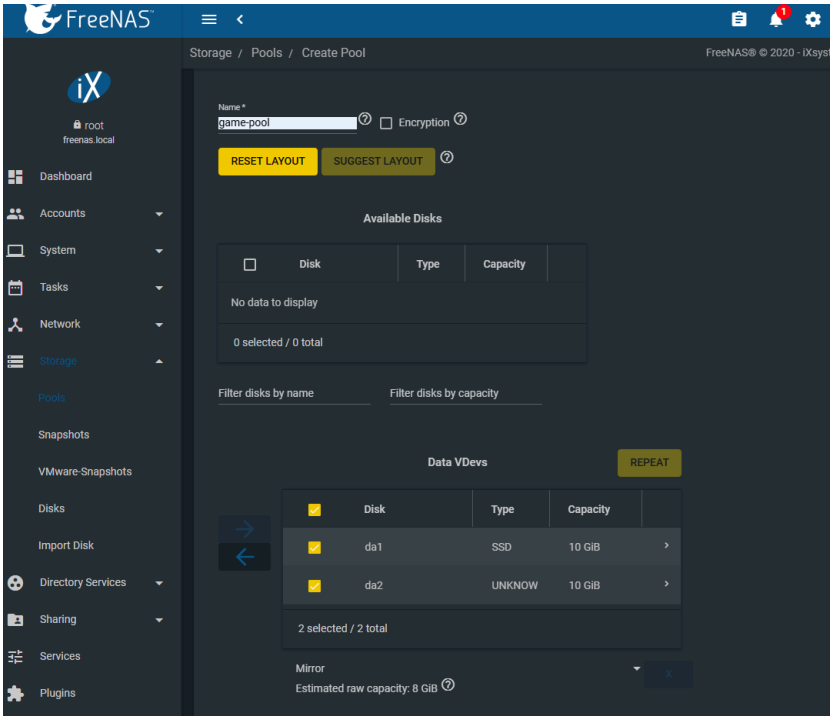
후에 `http://192.168.20.100` 접속하면 nas 서버 접속 완료



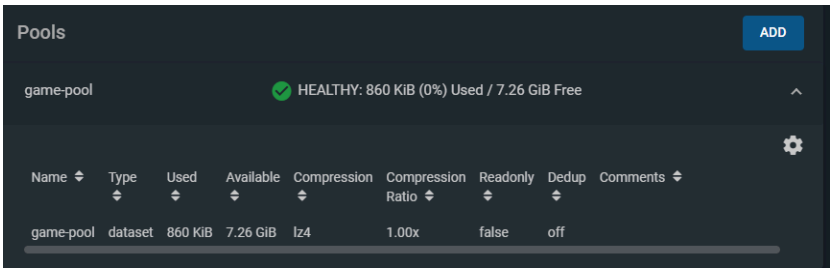
Pool: 물리적인 디스크(또는 가상 디스크)를 묶어서 스토리지를 구성하는 저장소.

1단계: 새로운 Pool 만들기

- `Storage → Pools → ADD` 를 눌러서 현재 화면으로 이동하기
- `Create new pool` 을 선택하고 `CREATE POOL` 을 누릅니다.



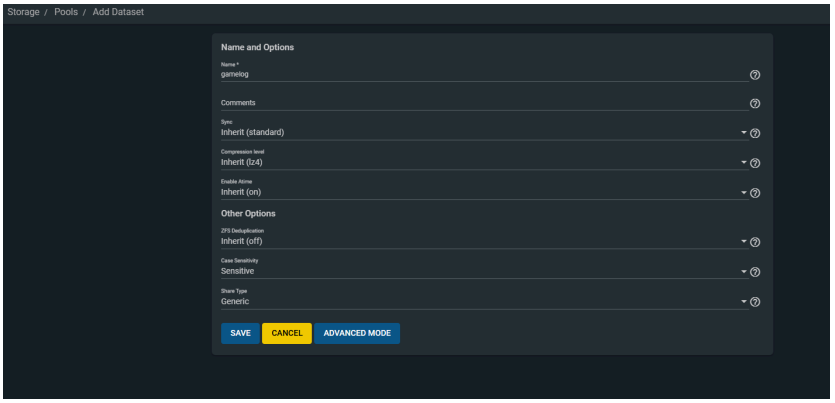
<< 설정후 create



설정하게 되면 game-pool이 생성된걸 확인 할 수 있음.

Dataset 생성하기

1. 생성한 `game-pool` 옆의 ⋮ 아이콘 클릭 후 **Add Dataset** 선택
2. Dataset 이름을 입력합니다. (예: `gamedata`)
3. 기본 옵션으로 설정하고 `Submit` 버튼을 누릅니다.



생성된 Dataset에 권한 설정하기

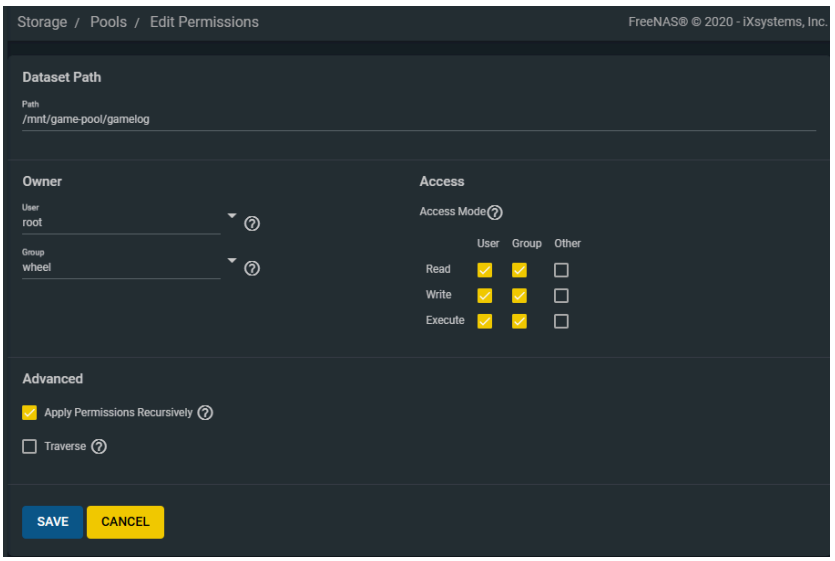
- Dataset 목록에서 생성한 `gamedata` 옆의 `⋮` 클릭 후 **Edit Permissions**를 선택합니다.

권장 권한 설정:

- User: root
- Group: wheel 또는 root
- Apply Permissions Recursively(하위 권한적용) 체크

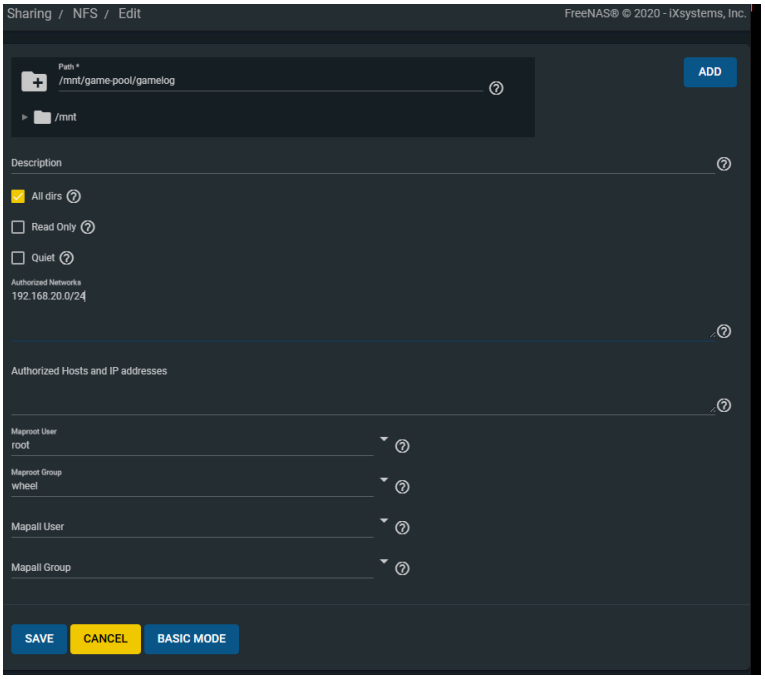
Permission:

- 770



NFS 공유 설정

- 좌측 메뉴에서 `Sharing → NFS` 이동
- `ADD`를 클릭하여 새 NFS 공유 설정
- Path: `/mnt/game-pool/gamedata`
- All dirs 체크
- Allowed Networks: `192.168.20.0/24`
- Mapall User/Group: `root/wheel`
- `Submit`을 눌러 NFS 공유 완료



이후 Kubernetes에서 NFS를 통해 NAS로 접근하면 정상적으로 읽고 쓸 수 있습니다.

Kubernetes에 NFS 연동

연동 작업은 **Master VM**에서 수행하였습니다.

📍 PersistentVolume(PV) 및 PersistentVolumeClaim(PVC) 설정

- `pv.yaml` 파일 구성:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: game-pv-gamedata
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /mnt/game-pool/gamedata
    server: 192.168.20.100
  persistentVolumeReclaimPolicy: Retain
```

- `pvc.yaml` 파일 구성:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: game-data-pvc
  namespace: game
spec:
```

```
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  volumeName: game-pv-gamedata
```

📍 **Kubernetes 적용 및 확인**

```
kubectl apply -f pv.yaml
kubectl apply -f pvc.yaml
kubectl get pv
kubectl get pvc -n game
```

```
[root@master0 ~]# k get pv
NAME              CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                       STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
game-pv-gamedata  10Gi       RWX            Retain           Bound    game-server/game-data-pvc                  <unset>                          11d
```

```
[root@master0 ~]# kubectl get pvc -n game-server
NAME            STATUS   VOLUME             CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
game-data-pvc   Bound    game-pv-gamedata   10Gi       RWX                           <unset>                 11d
```

Ansible 연동 및 자동화 백업

- Ansible에서 설정된 자동화 코드 실행 시,
  - 게임 컨테이너에서 생성되는 **데이터 및 로그 파일이 자동으로 NFS 공유 디렉토리에 백업**
- 백업은 NAS의 Dataset으로 저장되며, UI 또는 CLI로 직접 확인 가능

freenas CLI로 확인

```
root@freenas[...ame-pool/gamelogs/findboomgame/backup]# ls
game-backup-20250314-050229.tar.gz      game-backup-20250314-080423.tar.gz
game-backup-20250314-050543.tar.gz      game-backup-20250314-126576.tar.gz
root@freenas[...ame-pool/gamelogs/findboomgame/backup]# ls -l
total 18
-rw-r--r--  1 root  wheel  168 Mar 13 13:02 game-backup-20250314-050229.tar.gz
-rw-r--r--  1 root  wheel  168 Mar 13 13:05 game-backup-20250314-050543.tar.gz
-rw-r--r--  1 root  wheel  168 Mar 25 03:52 game-backup-20250314-080423.tar.gz
-rw-r--r--  1 root  wheel  168 Mar 25 03:52 game-backup-20250314-126576.tar.gz
```

이번 작업을 통해 Kubernetes 클러스터와 안정적으로 연동 가능한 NAS 기반 저장소를 구축하였습니다.

FreeNAS의 웹 기반 설정과 ZFS 지원 덕분에 운영 효율성이 높아졌으며, Ansible 자동화를 결합함으로써

지속적인 백업 및 로그 보존 체계까지 완성

되었습니다.

Helm 차트를 활용한 Prometheus & Grafana 기반 Kubernetes 모니터링 설정

본 작업은 Kubernetes 클러스터의 자원 사용 현황과 서비스 상태를 실시간으로 모니터링하기 위해,

Helm 차트를 이용한 Prometheus & Grafana 기반의 모니터링 환경을 구성한 작업입니다.특히 Helm 차트를 활용함으로써**복잡한 Kubernetes 리소스들을 손쉽게 설치하고 관리** 할 수 있었으며, 커스터마이징 및 확장성 또한 높였습니다.

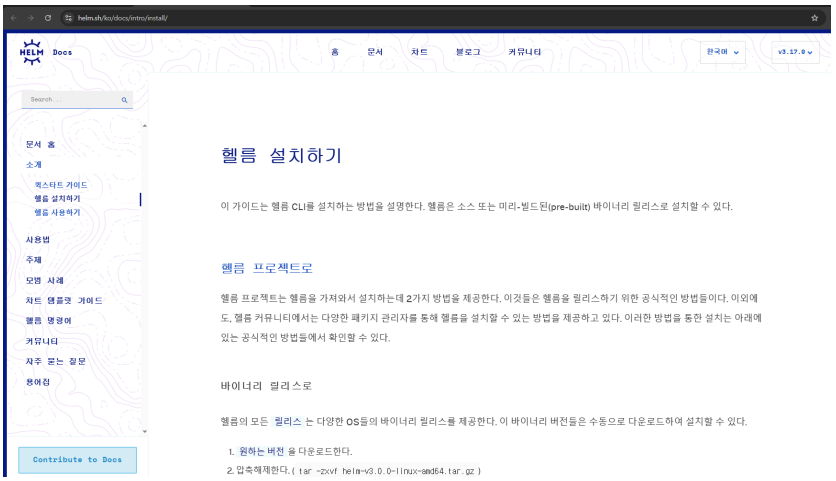
✅ Helm이란?

Helm은 Kubernetes 애플리케이션 배포를 간편하게 도와주는 **패키지 매니저**입니다.

- Helm은 **차트(chart)**라는 패키지 형식을 사용하며, 이 차트는 Kubernetes 리소스를 묶어 템플릿 형태로 제공합니다.
- 복잡한 구성 요소를 명령 한 줄로 배포 가능하며, 다양한 오픈소스 애플리케이션을 공식 Helm 저장소에서 간단히 설치할 수 있습니다.
- 설치 이후에도 값 변경만으로 업그레이드나 설정 변경이 간편해져 **운영 자동화에 매우 적합**합니다.

🛠️ 1. Helm 설치 및 저장소 등록

Helm 설치는 https://helm.sh/ko/docs/intro/install/ 이 링크를 참고하였습니다.



Helm 설치

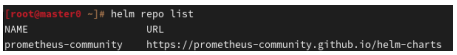
```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

📍 Helm 저장소 등록 및 업데이트

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

📍 Helm Chart 설치 확인

```
helm repo list
```



helm repo 안에 prometheus가 들어가 있는거 확인합니다.

📡 2. Prometheus & Grafana 설치

```
helm install prometheus prometheus-community/kube-prometheus-stack -n monitoring --create-namespa
ce
```

설치된 Helm 차트 확인:

```
helm list -n monitoring
```

```
[root@master0 ~]# helm list -n monitoring
NAME            NAMESPACE       REVISION        UPDATED                                   STATUS          CHART                           APP VERSION
prometheus      monitoring      1               2025-03-22 01:08:13.168084336 +0900 KST deployed        kube-prometheus-stack-70.1.1    v0.81.0
```

Prometheus 및 Grafana 포트 설정

🔷 1. Prometheus 서비스 확인

```
kubectl get svc -n monitoring | grep prometheus
```

```
[root@master0 ~/monitor]# kubectl get svc -n monitoring | grep prometheus
prometheus-grafana                        NodePort    10.98.75.91      <none>        80:30900/TCP                   21m
prometheus-kube-prometheus-alertmanager   ClusterIP   10.111.209.202   <none>        9093/TCP,8080/TCP              67m
prometheus-kube-prometheus-operator       ClusterIP   10.111.46.10     <none>        443/TCP                        67m
prometheus-kube-prometheus-prometheus     NodePort    10.107.183.3     <none>        9090:30090/TCP,8080:31028/TCP  67m
prometheus-kube-state-metrics             ClusterIP   10.110.99.93     <none>        8080/TCP                       67m
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                       67m
prometheus-prometheus-node-exporter       ClusterIP   10.105.47.70     <none>        9100/TCP                       67m
```

Prometheus가 Kubernetes의 `Deployment`, `Pod`, `Service` 등을 자동 감지하도록 설정해야한다.

PodMonitor 생성 수동 작업

`vi pod-monitor.yaml`

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: my-deployment-pod-monitor
  namespace: monitoring
  labels:
    release: prometheus
spec:
  selector:
    matchLabels:
      app: my-deployment
  podMetricsEndpoints:
    - port: metrics
      interval: 30s
```

ServiceMonitor 생성 수동 작업

`vi service-monitor.yaml`

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: my-deployment-monitor
  namespace: monitoring
```

```
  labels:
    release: prometheus
spec:
  selector:
    matchLabels:
      app: my-deployment
  endpoints:
    - port: metrics
      interval: 30s
```

실행

```
kubectl apply -f pod-monitor.yaml
kubectl apply -f service-monitor.yaml
```

실행 후

🔷 3. Grafana NodePort 변경 (외부 접근 허용)

1. 기존 서비스 수정

```
kubectl edit prometheus-grafana svc  -n monitoring
```

```
ports:
- name: http-web
  nodePort: 30900
  port: 80
  protocol: TCP
  targetPort: 3000
selector:
  app.kubernetes.io/instance: prometheus
  app.kubernetes.io/name: grafana
sessionAffinity: None
type: NodePort
```

type과 nodeport를 수정한다.

2. NodPort 확인

```
kubectl get svc -n monitoring prometheus-kube-prometheus-prometheus
```

Grafana 포트 확인

🔷 1. Grafana 서비스 확인

```
kubectl get svc -n monitoring prometheus-grafana
```

출력 예시 :

```
NAME                TYPE      CLUSTER-IP     EXTERNAL-IP   PORT(S)       AGE
prometheus-grafana   NodePort   10.111.180.22   <none>       80:31000/TCP   3d16h
```

✅ 접속 확인

- **Grafana 접속 주소**: http://192.168.100.50:30900
- Grafana 기본 ID/PW: `admin / prom-operator` (기본 설정 시)

📊 5. Grafana 대시보드 확인

- 설치된 Grafana에는 기본적으로 Kubernetes 클러스터 및 노드 상태를 확인할 수 있는 다양한 대시보드가 탑재되어 있음
- 커스텀 대시보드 구성도 가능

✅ 확인 가능한 항목

- 게임 서비스별 Pod 상태 및 리소스 사용률
- 노드 CPU, 메모리, 네트워크 트래픽 현황
- Prometheus 쿼리 기반의 세부 지표 분석

✅ 1번 대시보드 – 노드 리소스 모니터링 (Node Exporter)

> 📍 데이터 소스: Prometheus
>
> 📍 대상: `Node (서버)` 리소스 사용량
>
> 📍 인스턴스 예시: `192.168.100.50:9100`

🔍 주요 모니터링 항목:

- **CPU Usage / Load Average**

  → 각 노드의 CPU 사용률 및 평균 부하
- **Memory Usage**

  → 사용 중인 메모리 양과 총 메모리 대비 사용률 (%)
- **Disk IO 및 Disk Space Usage**

  → 디스크 입출력 추이 및 파티션별 용량 사용률
- **Network I/O**

  → 노드 기준 수신/전송 트래픽과 패킷량 모니터링

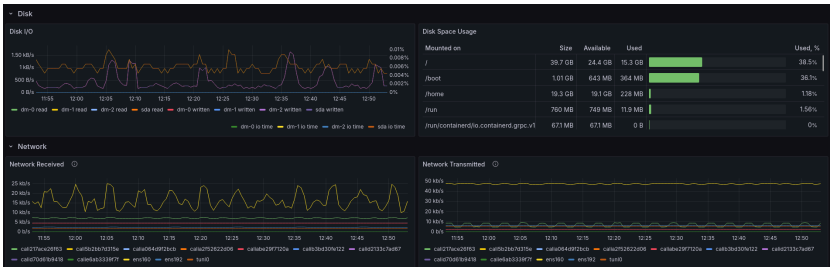
📌 이 대시보드는 **물리 서버 또는 VM 노드 자체의 성능**을 모니터링하는 데 사용됩니다.
클러스터의 상태와 관계없이 개별 노드의 리소스를 분석할 수 있습니다.

✅ 2번 대시보드 – 노드 리소스 모니터링 (Node Exporter)

> 📍 데이터 소스: Prometheus
>
> 📍 대상: `Node (서버)` 리소스 사용량
>
> 📍 인스턴스 예시: `192.168.100.53:9100`

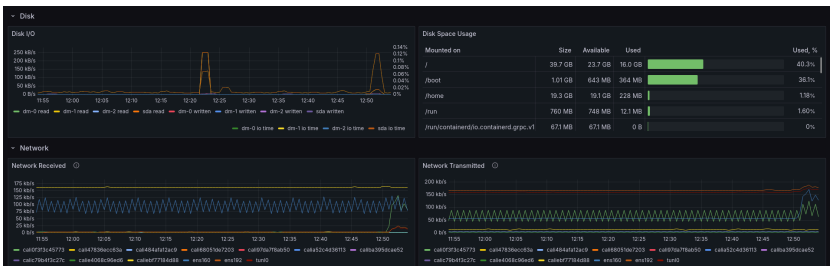
✅ 3번 대시보드 – 노드 리소스 모니터링 (Node Exporter)

> 📍 데이터 소스: Prometheus
>
> 📍 대상: `Node (서버)` 리소스 사용량
>
> 📍 인스턴스 예시: `192.168.100.54:9100`



✅ 4번 대시보드 – 노드 리소스 모니터링 (Node Exporter)

> 📍 데이터 소스: Prometheus
>
> 📍 대상: `Node (서버)` 리소스 사용량
>
> 📍 인스턴스 예시: `192.168.100.55:9100`



✅ 5번 대시보드 – 게임 서비스(Pod) 리소스 모니터링

> 📍 데이터 소스: Prometheus
>
> 📍 Namespace 기준 필터링: `game-server`, `game`, `ingress-nginx`, `kube-system`, `monitoring` 등
>
> 📍 대상: **Deployment 및 Pod 단위**

🔍 주요 모니터링 항목:

- **CPU Quota / Usage**

  → 각 게임 Deployment(Pod)에서 사용 중인 CPU 자원

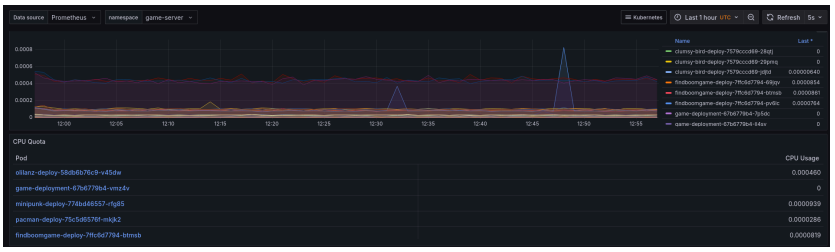
- **Memory Usage (RSS, Cache, Total)**

  → Pod 단위의 실시간 메모리 사용량

- **네트워크 트래픽**

  → 현재 트래픽 대역폭 및 패킷 수 기준 전송/수신 상태

📌 이 대시보드는 Kubernetes 클러스터 내에서 실행 중인 **게임 서비스 컨테이너(Pod) 단위 리소스 사용량**을 모니터링합니다. 리소스 오버 사용, 비정상 Pod 탐지, 성능 튜닝 분석 등에 유용합니다.



✅ 6번 대시보드 – 클러스터 전체 상태 모니터링 (Kubernetes Cluster Overview)

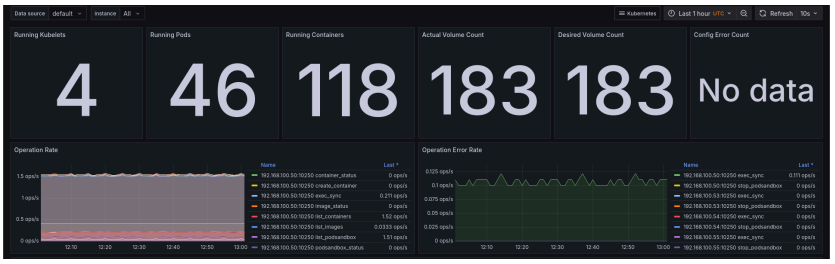
> 📍 데이터 소스: Prometheus
>
> 📍 대상: 클러스터 레벨, Workload, PVC, Pod Lifecycle 등

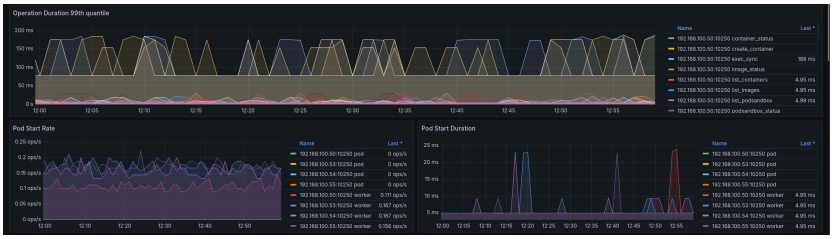
🔍 주요 모니터링 항목:

- **클러스터 리소스 요약**
  - 실행 중인 노드, Pod, 컨테이너 수, PVC 수 등
- **Operation Error Rate / State**
  - 클러스터 내부 구성요소의 상태 이상 또는 오류 발생률
- **Pod Start Rate / Duration**

  - Pod 시작 속도 및 지연 시간 → 성능 문제 탐지 가능
- **Volume 관련 지표**
  - PVC 바인딩 상태, 사용량 등 스토리지 상태 정보 포함

📌 이 대시보드는 클러스터 전반의 상태를 한눈에 파악할 수 있는 **종합 상태 대시보드**로, 운영자나 관리자 입장에서 **장애 감지 및 자원 상태 점검**에 매우 유용합니다.





CI/CD 파이프라인 구축 작업

본 프로젝트에서는 게임 사이트의 지속적 통합 및 배포 자동화를 위해, **Gitea → Jenkins → Docker → Kubernetes → HAProxy/WAF**로 이어지는 **CI/CD 파이프라인**을 구성하였습니다.

이를 통해 소스코드 푸시부터 Docker 이미지 빌드, Kubernetes 배포, 보안 및 로드밸런서 적용까지 모든 단계를 자동화하여 **운영 효율성과 배포 안정성**을 확보하였습니다.

✅ 사용 도구 및 도입 이유

도구	사용 목적	선택 이유
Gitea	Git 저장소	경량의 자체 호스팅 Git 서버로, 내부 네트워크 환경에 적합
Jenkins	CI/CD 자동화	플러그인 생태계가 풍부하고, Ansible 및 Docker 등과의 연동이 용이
Docker Private Registry	이미지 저장소	온프레미스에서 자체 이미지 관리 및 보안 유지를 위해 사용
Ansible	Kubernetes 배포 자동화	선언적 설정 관리로 반복 가능한 배포 수행 가능
HAProxy / WAF	트래픽 분산 및 보안 강화	외부 요청 필터링 및 NodePort 기반 부하 분산 처리

구성 요소 및 주요 IP

구성 요소	주소
Gitea	http://192.168.100.75:3000
Jenkins	http://192.168.100.75:8080
Docker Registry	http://192.168.100.75:5000
Kubernetes Master Node	192.168.100.50
HAProxy (LB VM)	192.168.100.61
WAF VM	172.16.6.60

🛠️ 파이프라인 구축 절차

1. Gitea 설치 및 프로젝트 레포지토리 생성

- Docker로 Gitea 컨테이너 실행
- `game-project` 라는 Git 저장소 생성
- 홈페이지 코드와 Jenkinsfile, Kubernetes 배포 YAML 등을 푸시

2. Jenkins 설치 및 Gitea 연동

- Jenkins를 Docker로 실행
- 초기 설정 및 관리자 계정 생성
- Gitea 인증 정보 등록 (Credentials)
- Git, Docker, Kubernetes, Pipeline 등 플러그인 설치

3. Docker Private Registry 설정

- 로컬 Docker Registry 실행 (`192.168.100.75:5000`)
- Jenkins 내 Docker가 해당 Registry로 이미지를 푸시할 수 있도록 `insecure-registries` 설정
- Registry 접근 확인 (`curl -I http://192.168.100.75:5000/v2/`)

4. Pipeline 구성 (Jenkinsfile)

주요 단계:

1. **Checkout Code**

   → Gitea에서 소스 코드 가져오기

2. **Build Docker Image**

   → `Dockerfile` 기반으로 게임 사이트 이미지 빌드

3. **Push to Registry**

   → 개인 Registry로 Docker 이미지 업로드

4. **Deploy with Ansible**

   → Ansible을 이용해 Kubernetes에 배포하고 HAProxy/WAF 설정 자동화

5. Ansible을 통한 배포 작업

- `deploy_game_site.yml` 플레이북을 통해 다음 작업 자동화:
  - Namespace 생성

- ConfigMap, Deployment, Service 설정
- HAProxy 부하 분산 설정 삽입
- WAF NAT 및 방화벽 포트 열기

6. SSH Key 기반 인증 설정

- Jenkins 컨테이너에서 SSH 키 생성
- 각 Kubernetes 노드 및 LB/WAF VM에 공개키 등록
- Jenkins → Ansible → 각 VM으로 원격 배포 가능 상태 구축

구현과정

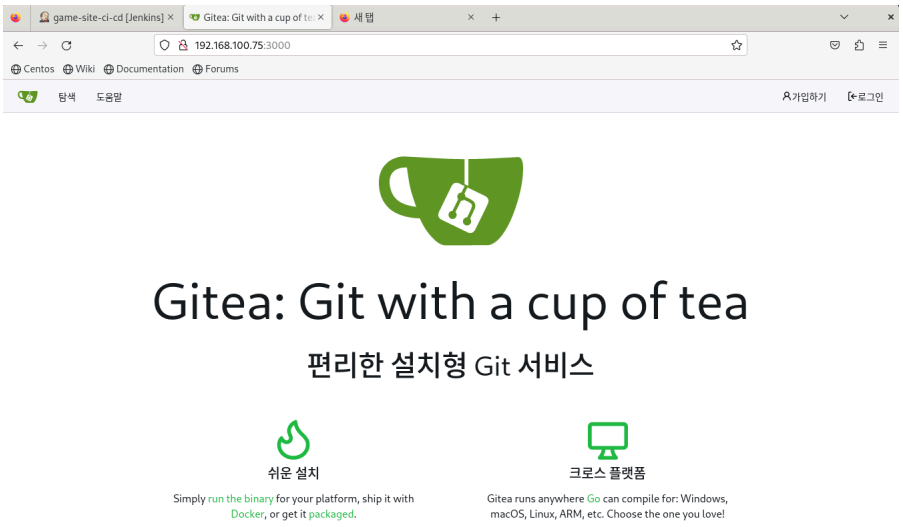
CI-CD VM 환경에서

gitea 컨테이너 실행한다.

```
docker run -d --name gitea \
  -p 3000:3000 -p 222:22 \
  -v /var/lib/gitea:/data \
  gitea/gitea:latest
```

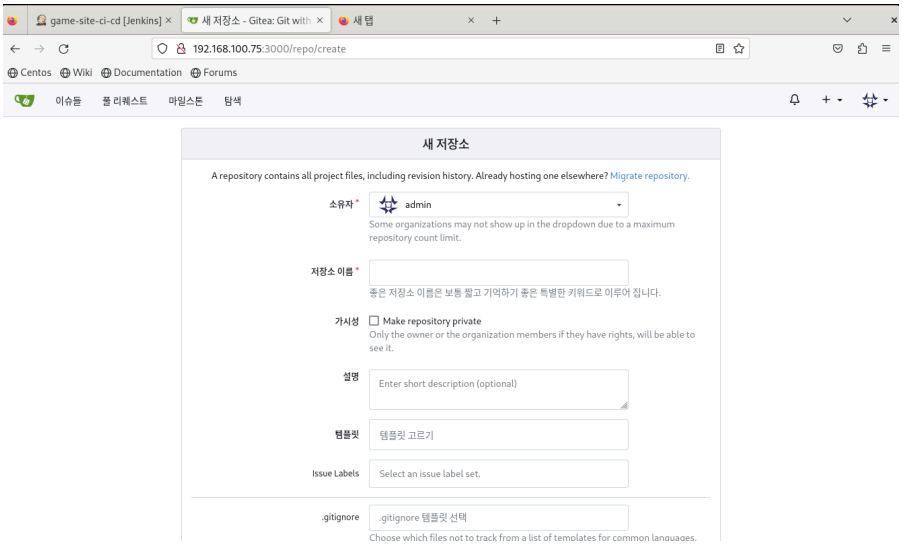
📌Gitea 웹 UI 접속:

`http://cicd.game.local:3000`



가입하기 클릭 → 사용자명 `admin` → 이메일주소 아무거나 → 비밀번호 `admin1234` 비밀번호 `admin1234` 계정 등록후 로그인

로그인후 새 저장소 생성하기

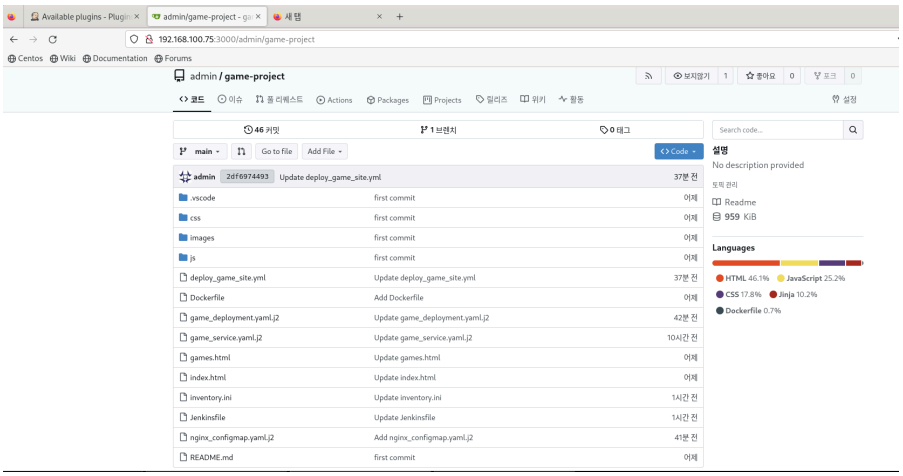


저장소 이름 : `game-project`

나머지 기본 설정 후 저장소 만들기 클릭

홈페이지 게임 사이트 코드를 푸시 할거다.

```
touch README.md
git init
git checkout -b main
git add .
git commit -m "first commit"
git remote add origin http://192.168.100.75:3000/admin/game-project.git
git push -u origin main
```



필요한 파일들을 넣어 놓은뒤 Jenkins 컨테이너 실행 및 설정을 수행합니다.

(2) Jenkins 컨테이너 실행 및 설정

1. Jenkins 컨테이너 실행

```
docker run -d --name jenkins \
  -p 8080:8080 -p 50000:50000 \
```

```
-v /var/run/docker.sock:/var/run/docker.sock \
-v jenkins_home:/var/jenkins_home \
jenkins/jenkins:lts
```

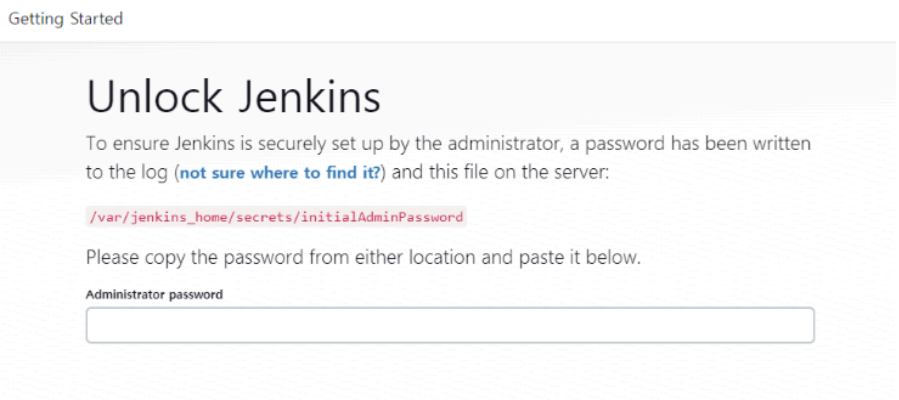
`docker ps -a` 로 Up 되었나 확인.

```
[root@main ~]# docker ps -a
CONTAINER ID   IMAGE                 COMMAND                  CREATED       STATUS         PORTS
                                NAMES
75f0f34c39f3   jenkins/jenkins:lts   "/usr/bin/tini -- /u…"   12 days ago   Up 4 seconds   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp, 0.0.0.0:50000->50000
```

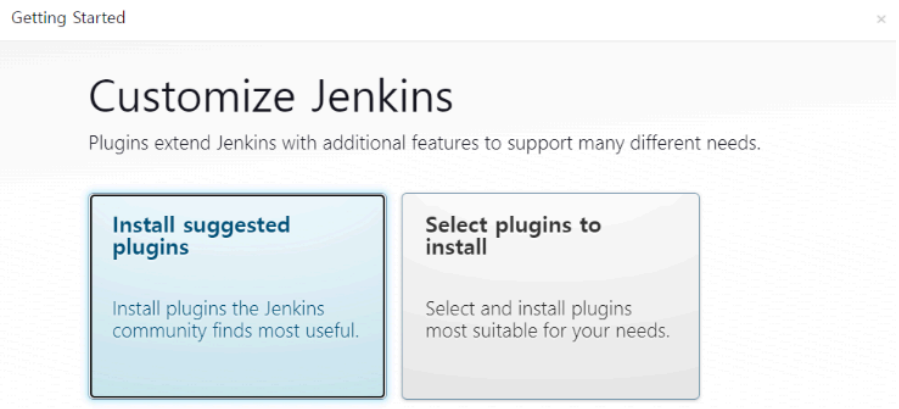
📌Jenkins **웹 UI 접속:**

http://cicd.game.local:8080 접속 후 관리자 비밀번호 입력

```
docker exec -it jenkins bash -- cat /var/jenkins_home/secrets/initialAdminPassword
```

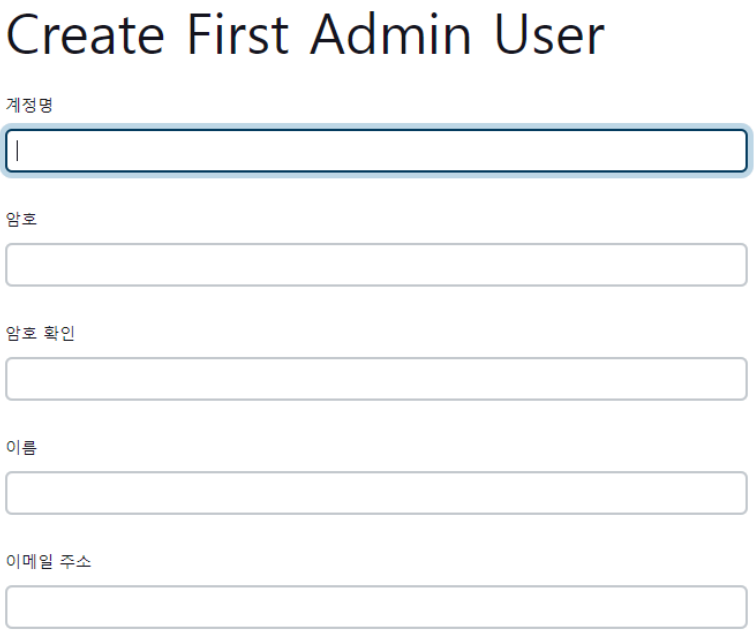


위에서 확인한 비밀번호 입력하기

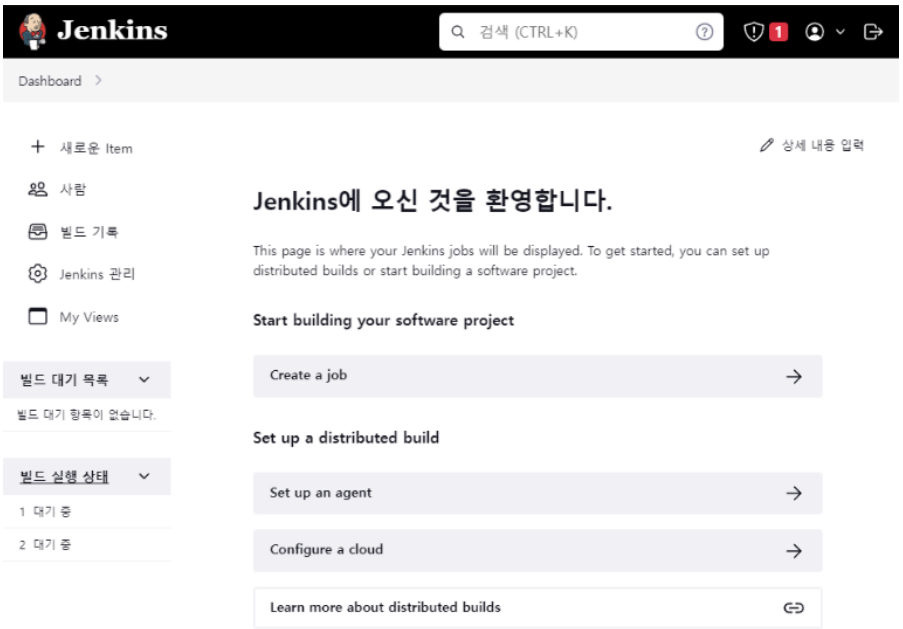


플러그인 설치에 대한 화면이 나오는데 왼쪽을 클릭해서 모든 플러그인을 설치해주자.

`Install suggested plugins`



플러그인 설치가 끝나면 초기 관리자 계정생성을 요청하는 페이지가 나온다. User를 생성한다.



초기 셋팅을 다 하게 되면 이러한 화면이 나온다.

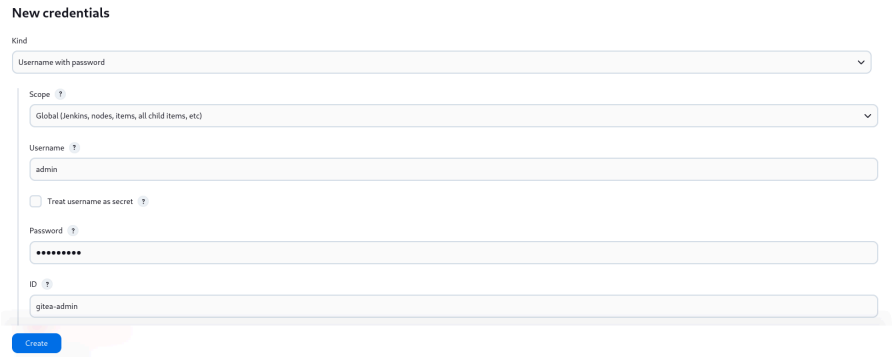
- **필요한 플러그인 설치**
  - **Git** 플러그인 (Gitea 연동)
  - **Docker** 플러그인 (Docker 이미지 빌드)
  - **Kubernetes** 플러그인 (Kubernetes 배포)
  - **Ansible** 플러그인 (Ansible 플레이북 실행)
  - **Pipeline** 플러그인 (Jenkinsfile 지원)

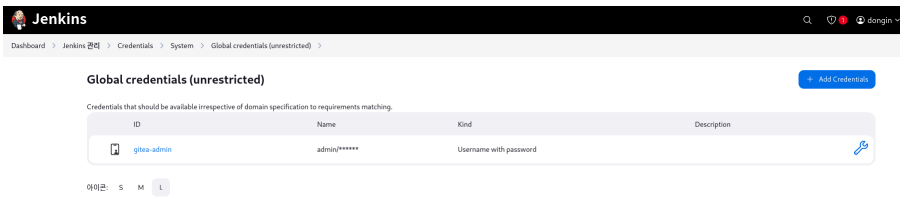
Install suggested plugins를 했어도 plugin이 포함 안되어 있는 경우

Jenkins 관리 → plugins → Available plugins 에서 위 플러그인을 설치 하자.

1. **Jenkins에 Gitea 연결**
   - **Jenkins → Jenkins 관리 → Credentials → global → Add Credentials 클릭**
   - Gitea의 사용자 인증 정보(ID/PW) 추가

   - Username : `admin`
   - Password : `gitea1234`
   - ID : `gitea-admin`
   - create





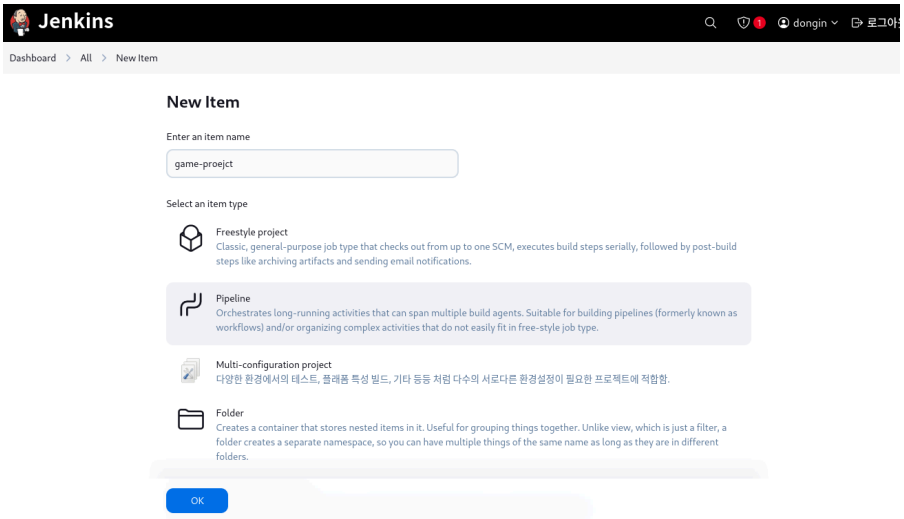
Jenkins와 gitea가 연결된 상태이다.

2. Jenkins Pipeline 구현 (Jenkinsfile)

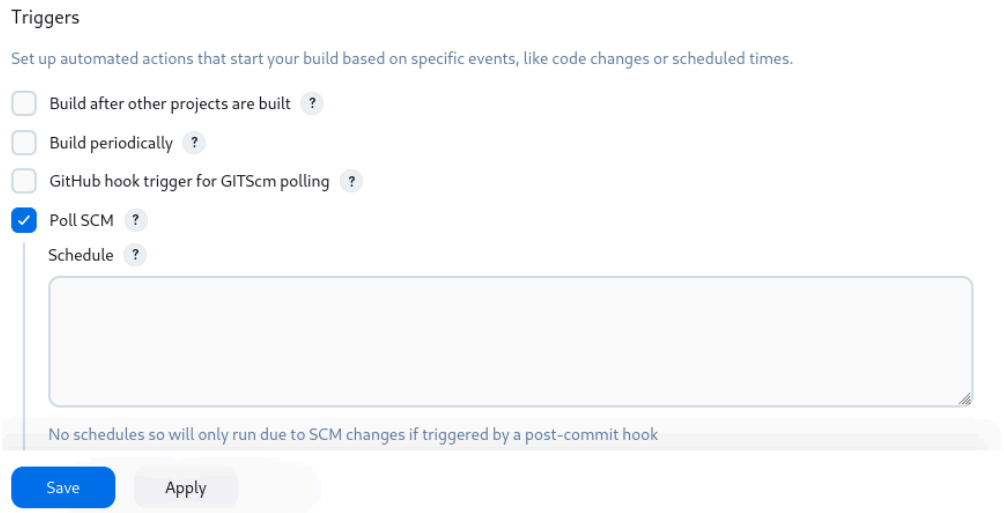
1. **Jenkins에서 새로운 Pipeline Job 생성**

- Jenkins 대시보드 → New Item → Pipeline 선택

`name : game-project`



Triggers Poll SCM 체크



아래 Pipline 스크립트 작성

```
pipeline {
    agent any  // 파이프라인이 실행될 에이전트를 정의 (여기서는 어떤 에이전트든 사용 가능)

    environment {
        // Gitea 저장소 URL과 인증 정보 설정
        GITEA_URL = 'http://192.168.100.75:3000/admin/game-project.git'  // Gitea 저장소 URL
        GITEA_CREDENTIALS_ID = 'gitea-admin'  // Jenkins에 등록된 Gitea 인증 정보 ID
        SSH_KEY_PATH = '/var/lib/jenkins/.ssh/id_rsa'  // Ansible 실행을 위한 SSH 키 경로
    }

    stages {
        // 첫 번째 단계: 코드 체크아웃 (Gitea에서 코드 가져오기)
        stage('Checkout Code') {
            steps {
                script {
```

```
                checkout([  // GitSCM을 이용한 Git 저장소에서 코드 가져오기
                    $class: 'GitSCM',
                    branches: [[name: '*/main']],  // 'main' 브랜치에서 코드 체크아웃
                    userRemoteConfigs: [[
                        url: env.GITEA_URL,  // Gitea 저장소 URL
                        credentialsId: env.GITEA_CREDENTIALS_ID  // Jenkins에 저장된 Gitea 계정 정보
                    ]]
                ])
            }
        }
    }

    // 두 번째 단계: Docker 이미지 빌드
    stage('Build Docker Image') {
        steps {
            script {
                // Docker 이미지를 빌드하고, 로컬 레지스트리(192.168.100.75:5000)에 태그를 붙여서 생성
                sh 'docker build -t 192.168.100.75:5000/game-site-image:latest .'
            }
        }
    }

    // 세 번째 단계: Docker 이미지를 레지스트리에 푸시
    stage('Push to Registry') {
        steps {
            script {
                // 빌드한 Docker 이미지를 레지스트리(192.168.100.75:5000)로 푸시
                sh 'docker push 192.168.100.75:5000/game-site-image:latest'
            }
        }
    }

    // 네 번째 단계: Ansible을 이용한 배포
    stage('Deploy with Ansible') {
        steps {
            script {
                // Ansible을 이용하여 배포를 수행 (inventory.ini와 playbook을 사용)
                sh '''
                    export PATH=$PATH:/usr/bin  // Ansible 명령어를 사용할 수 있도록 PATH 설정
                    ansible-playbook -i inventory.ini deploy_game_site.yml \  // Ansible Playbook 실행
                    --ssh-common-args="-o StrictHostKeyChecking=no -o IdentitiesOnly=yes -i ${SSH_KEY_PA
TH}"  // SSH 연결 옵션
                '''
            }
        }
    }
  }
}
```

`Save` 버튼 클릭

이후 gitea 레파지토리에 Dockerfile과 Kubernetes 배포 설정을 하기위한 파일 코드를 푸시해야한다.

- `Dockerfile`

```
FROM nginx:latest
COPY . /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- `deploy_game_site.yml`

```
- name: Deploy Game Site on Kubernetes
  hosts: master
  become: yes
  vars:
    game_name: "game-site"
    app_name: "game-app"
    deployment_name: "game-deployment"
    service_name: "game-service"
    configmap_name: "game-config"
    container_image: "192.168.100.75:5000/game-site-image:latest"
    container_port: 8080
    node_port: 30050
    kubernetes_namespace: "game"
  tasks:
    - name: Ensure Kubernetes namespace exists
      shell: kubectl create namespace {{ kubernetes_namespace }} --dry-run=client -o yaml | kubectl ap
ply -f -

    - name: Create Kubernetes ConfigMap YAML
      template:
        src: nginx_configmap.yaml.j2
        dest: "/tmp/{{ game_name }}-configmap.yaml"

    - name: Apply Kubernetes ConfigMap
      command: kubectl apply -f /tmp/{{ game_name }}-configmap.yaml

    - name: Create Kubernetes Deployment YAML
      template:
        src: game_deployment.yaml.j2
        dest: "/tmp/{{ game_name }}-deployment.yaml"

    - name: Create Kubernetes Service YAML
      template:
        src: game_service.yaml.j2
        dest: "/tmp/{{ game_name }}-service.yaml"

    - name: Apply Kubernetes Deployment
      command: kubectl apply -f /tmp/{{ game_name }}-deployment.yaml

    - name: Apply Kubernetes Service
      command: kubectl apply -f /tmp/{{ game_name }}-service.yaml

    - name: Wait for Pods to be in Running state
      shell: kubectl wait --for=condition=Ready pod -l app={{ app_name }} -n {{ kubernetes_namespace
}} --timeout=120s
```

```
- name: Configure HAProxy Load Balancer
  hosts: lb_vm
  become: yes
  vars:
    container_port: 8080
    node_port: 30050
  tasks:
    - name: Check if HAProxy is installed
      command: haproxy -v
      register: haproxy_installed
      ignore_errors: yes
      changed_when: false

    - name: Install HAProxy (only if not installed)
      raw: apt update && apt install -y haproxy
      when: haproxy_installed.rc != 0

    - name: Update HAProxy Configuration
      blockinfile:
        path: "/etc/haproxy/haproxy.cfg"
        block: |
          frontend game_frontend
              bind *:{{ container_port }}
              default_backend game_backend

          backend game_backend
              balance roundrobin
              server worker1 192.168.100.53:{{ node_port }} check
              server worker2 192.168.100.54:{{ node_port }} check
              server worker3 192.168.100.55:{{ node_port }} check
        marker: "# {mark} Ansible managed block - game"
      notify: Restart HAProxy

  handlers:
    - name: Restart HAProxy
      service:
        name: haproxy
        state: restarted
        enabled: yes

- name: Configure WAF VM
  hosts: waf_vm
  become: yes
  vars:
    container_port: 8080
    node_port: 30050
  tasks:
    - name: Configure NAT rules for WAF
      shell: |
        firewall-cmd --permanent --direct --add-rule ipv4 nat PREROUTING 0 -p tcp --dport {{ container_
port }} -j DNAT --to-destination 192.168.100.61:{{ container_port }}
```

```
    firewall-cmd --permanent --direct --add-rule ipv4 nat POSTROUTING 0 -o eth0 -j MASQUERADE
    firewall-cmd --reload
```

- `game_deployment.yaml.j2`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ deployment_name }}
  namespace: {{ kubernetes_namespace }}
  labels:
    app: {{ app_name }}
spec:
  replicas: 3
  selector:
    matchLabels:
      app: {{ app_name }}
  template:
    metadata:
      labels:
        app: {{ app_name }}
    spec:
      containers:
        - name: {{ app_name }}
          image: {{ container_image }}
          ports:
            - containerPort: {{ container_port }}
          volumeMounts:
            - name: nginx-config
              mountPath: /etc/nginx/conf.d
      volumes:
        - name: nginx-config
          configMap:
            name: {{ app_name }}-nginx-config
```

- `game_service.yaml.j2`

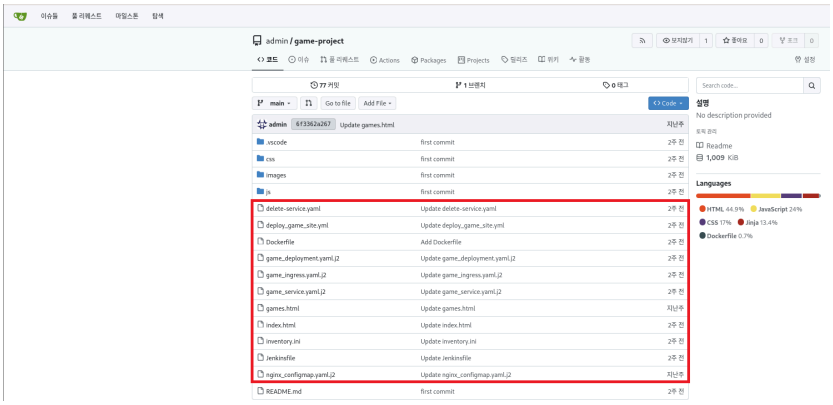
```
apiVersion: v1
kind: Service
metadata:
  name: game-service
  namespace: game
spec:
  selector:
    app: game-app
  type: NodePort
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 30050
```

- `nginx_configmap.yaml.j2`

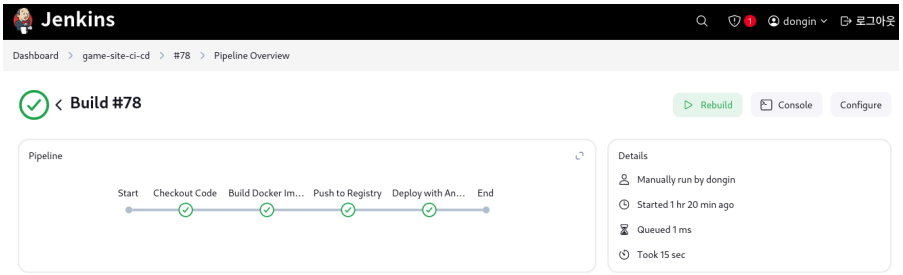
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ app_name }}-nginx-config
  namespace: {{ kubernetes_namespace }}
data:
  default.conf: |
    server {
        listen {{ container_port }};
        server_name localhost;
        location / {
            root /usr/share/nginx/html;
            index index.html;
        }
    }
```

- `inventory.ini`

```
[master]
192.168.100.50 ansible_ssh_user=root

[worker]
192.168.100.53 ansible_ssh_user=root
192.168.100.54 ansible_ssh_user=root
192.168.100.55 ansible_ssh_user=root
[lb_vm]
192.168.100.61 ansible_ssh_user=root
[waf_vm]
192.168.100.60 ansible_ssh_user=root
```



gitea 레파지토리에 저장하고 파이프라인 빌드 해하기



`kubectl get po,deploy,svc -n game`

🧪 테스트 및 결과 확인

1. Jenkins 빌드 성공 확인

- Pipeline Job 실행 → 모든 단계가 성공적으로 완료되었는지 확인

2. Kubernetes 상태 확인

```
kubectl get po,svc -n game
```

- `game-deployment` 및 `game-service(NodePort: 30050)` 가 실행 중이어야 함

```
[root@master0 ~/kube]# kubectl get po,deploy,svc -n game
NAME                                      READY   STATUS    RESTARTS        AGE
pod/findboomgame-deploy-7ffc6d7794-h9s6q  1/1     Running   1 (3h18m ago)   9h
pod/findboomgame-deploy-7ffc6d7794-hmvvj  1/1     Running   1 (3h18m ago)   9h
pod/findboomgame-deploy-7ffc6d7794-lmqg8  1/1     Running   0               7h36m
pod/game-deployment-67b6779b4-75g74       1/1     Running   0               83m
pod/game-deployment-67b6779b4-8njkb       1/1     Running   0               83m
pod/game-deployment-67b6779b4-nrp7k       1/1     Running   0               83m

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/findboomgame-deploy  3/3     3            3           9h
deployment.apps/game-deployment      3/3     3            3           104m

NAME                      TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/findboomgame-svc  NodePort   10.104.185.200   <none>        3000:30001/TCP   9h
service/game-service      NodePort   10.108.211.129   <none>        8080:30050/TCP   104m
```

`game-deployment` `game-service` `pod/game-deployment` 가 게임 사이트가 배포된 것을 확인 가능

3. HAProxy 설정 확인

```
cat /etc/haproxy/haproxy.cfg
```

- 해당 backend/server 라인에 Worker 노드와 포트가 등록되어 있어야 함

```
# BEGIN Ansible managed block - game
frontend game_frontend
    bind *:8080
    default_backend game_backend

backend game_backend
    balance roundrobin
    server worker1 192.168.100.53:30050 check
    server worker2 192.168.100.54:30050 check
    server worker3 192.168.100.55:30050 check
# END Ansible managed block - game
```

4. WAF 설정 확인

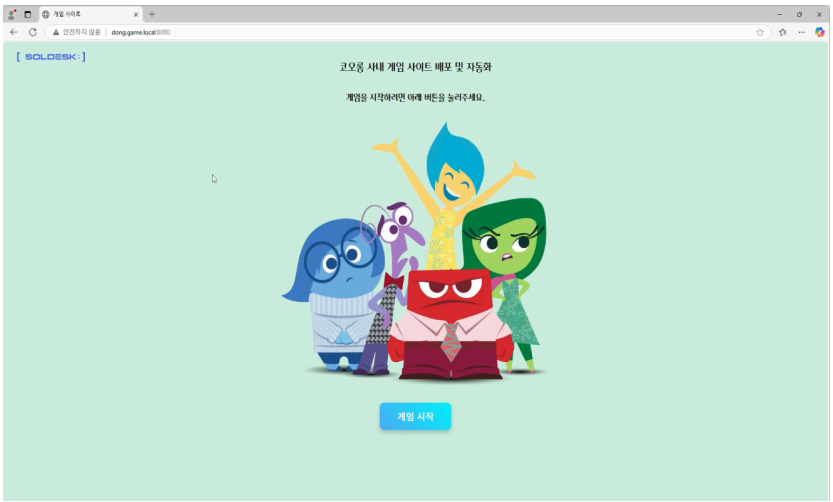
```
iptables -t nat -L -n -v
```

- NAT 및 포트 포워딩 설정이 반영되어 있어야 함

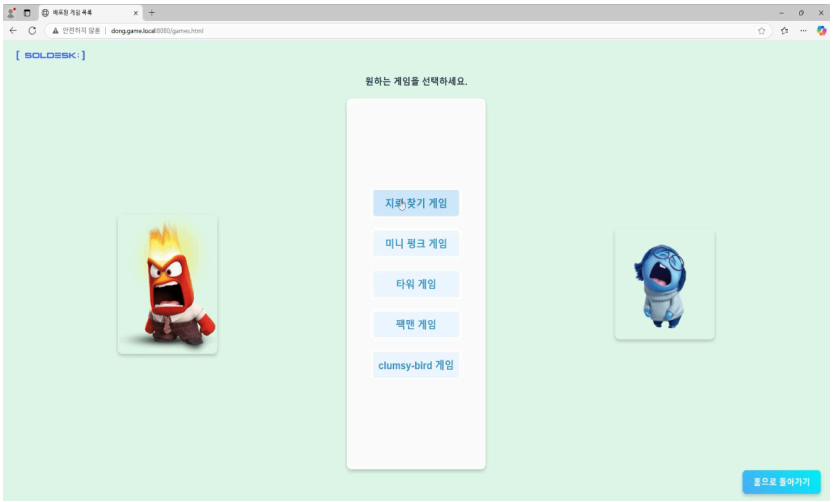
```
[root@WAF ~]# iptables -t nat -L -n -v
Chain PREROUTING (policy ACCEPT 54788 packets, 15M bytes)
 pkts bytes target     prot opt in     out     source               destination
   35  1820 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:8080 to:192.168.100.61:8080
```

5. 외부 접속 테스트

http://dong.game.local:8080



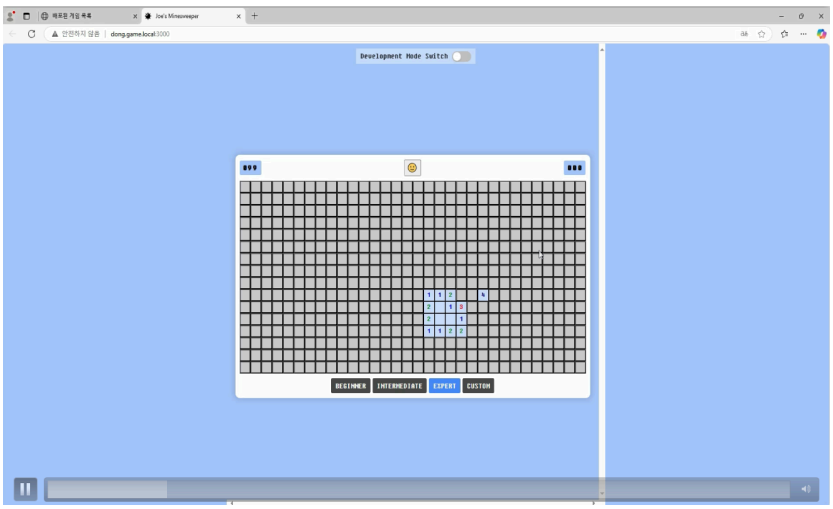
http://dong.game.local:8080/game.html



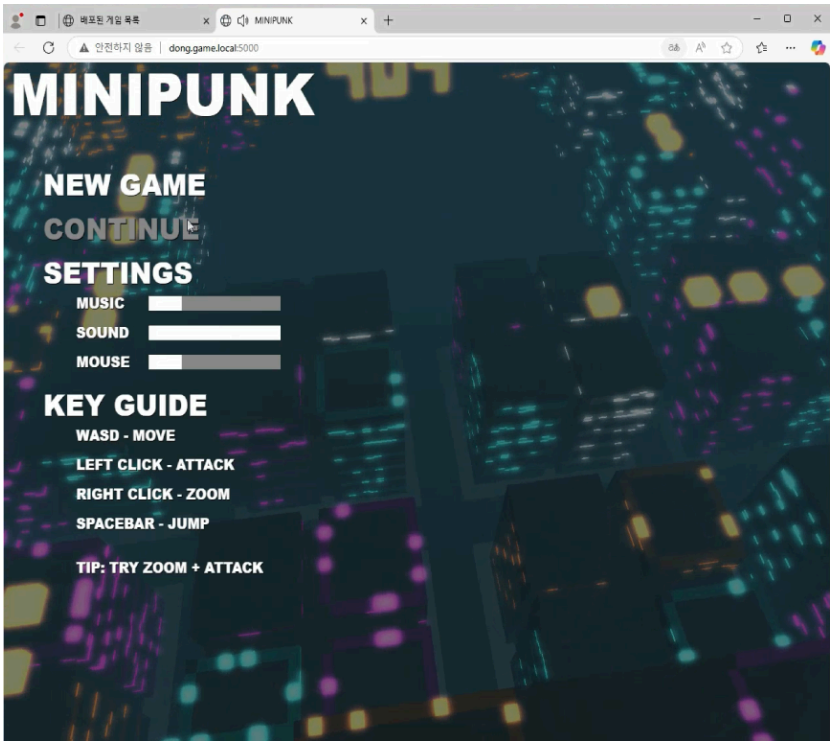
이후에 각 게임들을 누르게 되면 Ansible을 활용하여 쿠버네티스 게임 배포 및 nodeport로 외부 접속을 활성화 한뒤 LB 설정과 waf에서 nat 설정을 하여 외부로 빠져나가게 적용을 시켰습니다.

각 버튼들을 누르게 되면

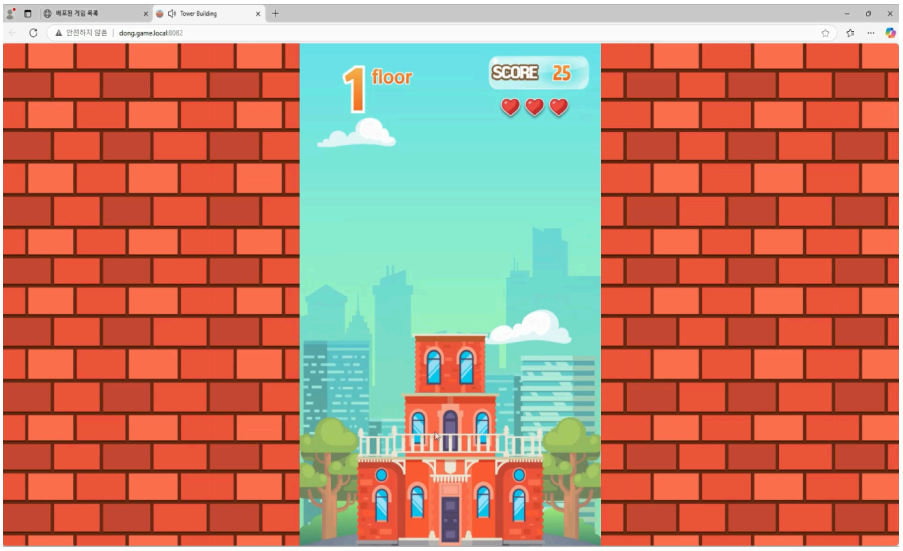
지뢰찾기 게임



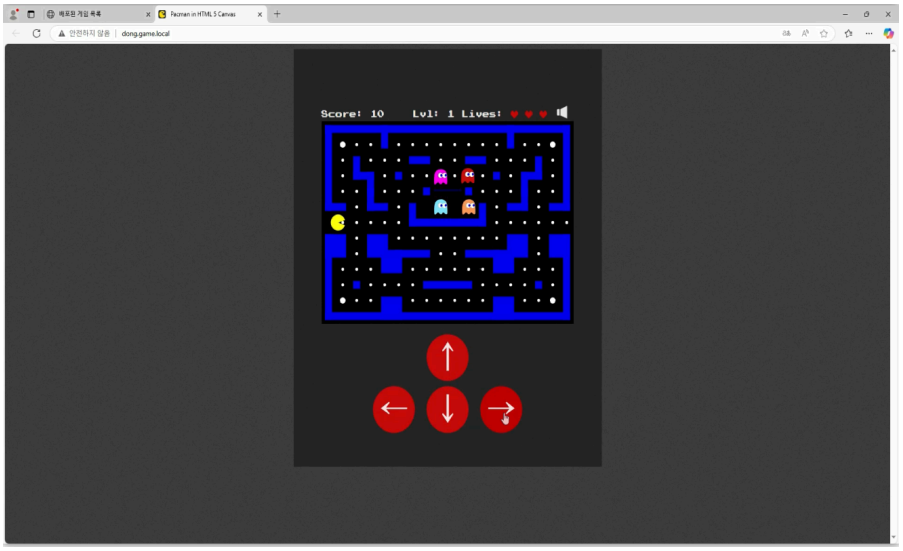
미니펑크 게임


MINIPUNK
NEW GAME
CONTINUE
SETTINGS
MUSIC
SOUND
MOUSE
KEY GUIDE
WASD - MOVE
LEFT CLICK - ATTACK
RIGHT CLICK - ZOOM
SPACEBAR - JUMP
TIP: TRY ZOOM + ATTACK

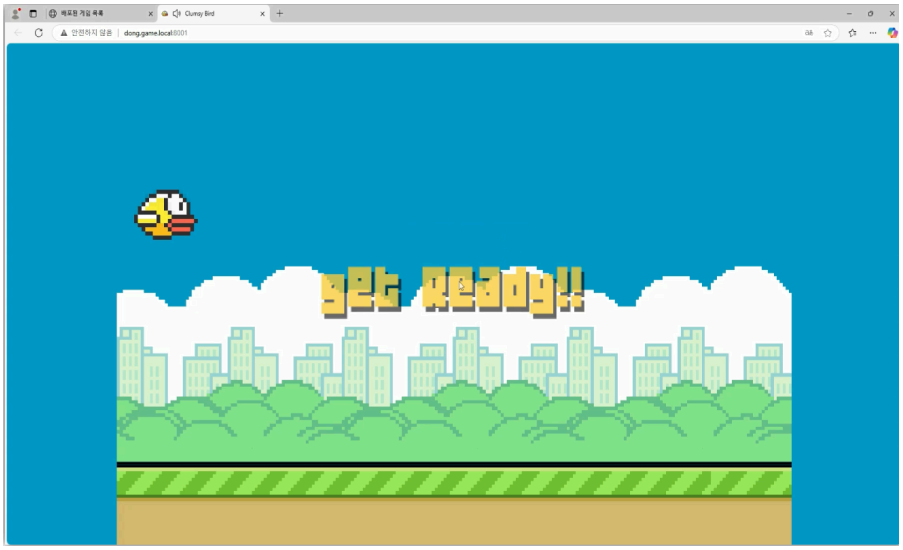
타워 쌓기 게임


1 floor
SCORE 25

팩맨 게임



Clumsy-bird 게임



마무리

이번 **온프레미스 기반 게임 서비스 자동화 프로젝트**는 단순한 기술 구현을 넘어, 하나의 완전한 서비스를 안정적으로 운영하기 위한 **전체 DevOps 생태계**를 직접 설계하고 실현해본 뜻깊은 경험이었습니다.

초기에는 단순히 "게임 서비스를 쿠버네티스에 띄워보자"는 목표에서 시작했지만, 실제 구현 과정에서는 **인프라 설계, 보안, 자동화, 모니터링, 장애 대응, 코드 관리** 등 다양한 영역이 유기적으로 연결되어야만 진정한 서비스 환경이 된다는 것을 체감하게 되었습니다.

특히 모든 작업을 온프레미스 환경에서 수행함으로써, 클라우드 환경과는 달리 우리가 **하드웨어와 네트워크, OS, 방화벽, 가상화 설정까지 직접 손을 대야 했다는 점에서 큰 도전과 학습이 있었습니다.**

Gitea를 통한 **자체 Git 레포지토리 관리**, Jenkins 기반 **CI/CD 파이프라인 구축**, Docker Private Registry 운영, Ansible을 통한 **Kubernetes 자동 배포**, Helm과 Grafana/Prometheus를 활용한 **시각화된 실시간 모니터링**, 그리고 HAProxy와 WAF를 통한 **보안 및 트래픽 분산**까지 모든 과정은 이론과는 전혀 다른 **실전 감각과 문제 해결 능력**을 요구했습니다.

예상치 못한 장애, 설정 오류, 퍼미션 문제, 네트워크 포워딩 등 실제 운영환경에서 마주할 수 있는 다양한 이슈들을 직접 경험하며, 기술이 단지 '작동'하는 것보다 '운영 가능한 상태'로 유지하는 것이 훨씬 더 중요하다는 것을 배웠습니다.

무엇보다 이번 프로젝트를 통해 느낀 가장 큰 성과는, **내 손으로 완성한 하나의 서비스가 외부에서 접속되고, 부하가 분산되고, 문제가 감지되고 자동으로 복구되는 전 과정을 구축해냈다는 성취감**이었습니다.

앞으로 이 기반 위에 기능을 확장하고, 모니터링을 고도화하고, 보안 정책을 강화하고, 더 나아가 클라우드나 하이브리드 환경으로 연결한다면 진정한 **엔터프라이즈 인프라 운영의 실무 역량**으로 이어질 수 있다고 생각합니다.