CCCR 4기 컨테이너 오케스트레이션 미니 프로젝트 - 최휘녕

Table of Contents >

1. 프로젝트 서론

이번 프로젝트는 CCCR 4기 DevOps 과정 중 컨테이너 오케스트레이션 파트의 실습 과제로서, 강의에서 배운 이론적 내용을 실제 환경에 적용해보는 경험이었습니다. 강의에서는 컨테이너 기술의 기초부터 시작하여, 쿠버네티스(Kubernetes)의 핵심 개념, 주요 오브젝트들(예: Pod, Deployment, Service, ConfigMap, Secret, Ingress 등)에 대한 이론을 다루었고, 이러한 요소들을 활용하여 하나의 완전한 인프라를 구축해보는 것을 최종 목표로 삼았습니다.

쿠버네티스는 현재 DevOps와 클라우드 네이티브 환경에서 핵심적인 기술로 자리잡고 있으며, 이를 실제로 구성하고 배포하는 능력은 실무에서 매우 중요합니다. 따라서 이번 프로젝트는 단순한 실습을 넘어, 실제 업무에서 발생할 수 있는 다양한 상황을 가정하고 이를 해결하는 능력을 기르는 데에 초점이 맞춰졌습니다.

본 프로젝트에서는 웹 서버와 데이터베이스 서버를 쿠버네티스 환경에서 각각 구성하고, 외부에서 접근할 수 있도록 인그레스(Ingress)를 활용해 배포하는 전 과정을 수행하였습니다. 이 과정에서 도커 이미지를 기반으로 한 애플리케이션 컨테이너화, 쿠버네티스 오브젝트의 정의 및 적용, 네트워크 연결 및 보안 설정까지 실질적인 운영 환경 수준의 작업을 경험해볼 수 있었습니다.

1.1 프로젝트 개요

프로젝트의 주요 목적은 강의에서 학습한 쿠버네티스의 핵심 개념을 직접 실습을 통해 체득하고, 이를 바탕으로 실제 서비스 환경에서 인프라를 어떻게 구성하고 운영할 수 있는지를 이해하는 데에 있습니다. 단순히 쿠버네티스 명령어를 익히는 것에 그치지 않고, 서비스 간의 연결 관계, 볼륨을 통한 데이터의 지속성 유지, 서비스를 외부에 노출하는 다양한 방식, 그리고 보안 설정(TLS 인증서 적용 등)까지 아우르는 전반적인 인프라 설계를 경험했습니다.

이 프로젝트는 개인 단위로 수행되었으며, 총 5일에 걸쳐 진행되었습니다. 클러스터 구성부터 각 컴포넌트의 배포, 테스트 및 문제 해결, 문서화까지 모두 스스로 진행함으로써 DevOps 엔지니어로서의 실무 감각을 익히는 데 큰 도움이 되었습니다.

1.2. 쿠버네티스 클러스터

본 프로젝트에서 사용한 쿠버네티스 클러스터는 VirtualBox 가상 머신(VM)을 기반으로 구성하였으며, 총 4개의 노드로 구성되어 있습니다.

- **컨트롤 플레인(Control Plane)** 1개: kube-control1
- **워커 노드(Worker Node)** 3개: kube-node1, kube-node2, kube-node3

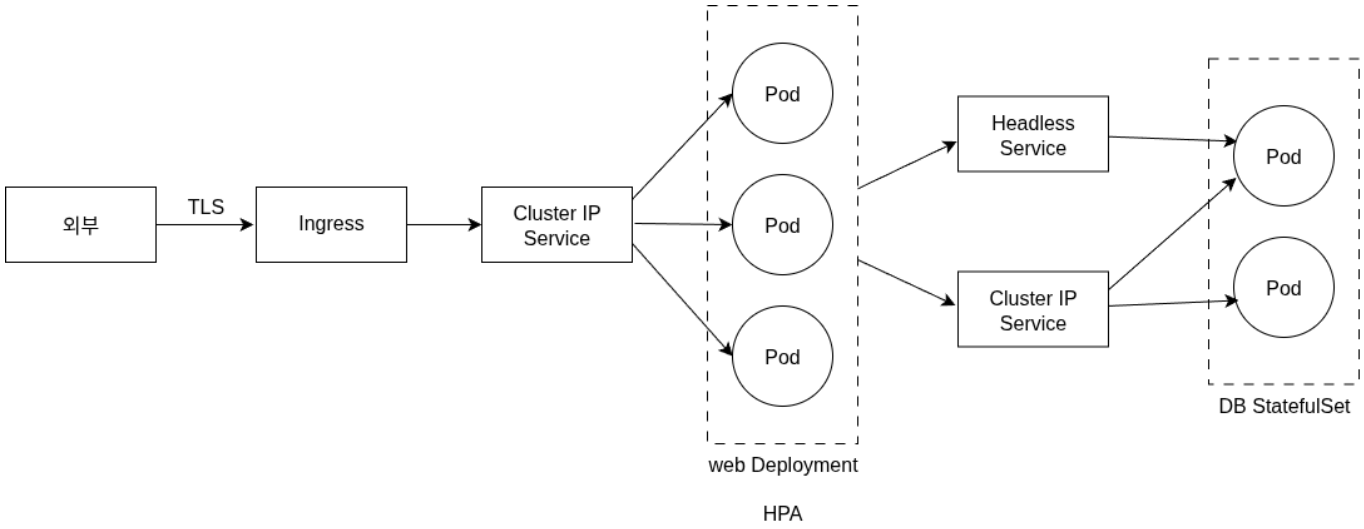
이러한 구성은 실무에서 자주 사용되는 멀티 노드 클러스터 환경을을 간접적으로 체험할 수 있게 해주며, 컨트롤 플레인과 워커 노드 간의 역할 분담, 통신 방식 등을 보다 명확히 이해하는 데 도움이 되었습니다.

사용된 소프트웨어 버전은 다음과 같습니다:

- **VirtualBox**: 7.1
- **Kubernetes**: 1.31

각 노드는 가상 환경이지만, 실제 쿠버네티스 클러스터의 작동 방식과 동일하게 작동하며, 네트워크 설정, 포트 포워딩, 리소스 제한 등도 현실적인 조건에 맞춰 구성되었습니다.

1.3. 아키텍처



이번 프로젝트에서는 웹 서버와 DB 서버로 구성된 단순하지만 실제 서비스에도 충분히 활용 가능한 기본 웹 애플리케이션 아키텍처를 설계하고 구현하였습니다.

- **웹 서버 구성**
  웹 서버는 WordPress 공식 이미지를 활용하여 구성하였으며, Deployment 오브젝트를 통해 총 3개의 파드를 배포하였습니다. 이 3개의 파드는 하나의 Cluster IP 서비스로 연결됩니다. DB와의 연결을 통해 동적으로 데이터를 주고받을 수 있도록 설정하였습니다.
  WordPress는 가장 널리 사용되는 CMS로, DB 연결을 확인할 수 있고 설정 파일을 ConfigMap으로 관리할 수 있어 쿠버네티스 실습을 목표로 하는 본 프로젝트에 적합하다고 생각해 선택했습니다.
- **데이터베이스(DB) 구성**
  데이터베이스는 MySQL 이미지를 활용하여 구성하였으며, StatefulSet 오브젝트를 통해 총 2개의 파드를 배포하였습니다. StatefulSet은 Pod의 고유성과 순서를 보장하는 오브젝트로, 데이터베이스처럼 상태(state)를 유지해야 하는 애플리케이션에 적합합니다.
  이중화 구조를 위해 하나의 파드는 읽기/쓰기(RW) 전용으로, 다른 하나는 읽기(Read-only) 전용으로 설정하였습니다. 이러한 구성은 단일 장애 지점을 줄이고, 고가용성(High Availability)을 실현하는 데에 중요한 역할을 합니다.
  또한, 쓰기 가능한 파드를 식별하기 위해 헤드리스 서비스(Headless Service)를 활용하였고, 읽기 전용 쿼리를 처리하기 위한 ClusterIP 기반의 서비스를 따로 구성하였습니다. 이처럼 다양한 서비스 유형을 활용함으로써, DB에 대한 접근 경로를 유연하게 제어할 수 있도록 하였습니다.
- **서비스 노출(Ingress 및 TLS 구성)**
  웹 서버는 외부와 통신할 수 있도록 Ingress 오브젝트를 통해 노출하였으며, 이를 통해 도메인 기반

접근과 경로 기반 라우팅을 구현하였습니다.
현재는 웹 서버가 하나의 서비스로 구성되어 있지만 향후 서비스가 추가될 경우 Ingress의 라우팅 기능을 활용해 여러 서비스를 효율적으로 관리할 수 있어 확장성을 염두에 두고 Ingress로 구성하였습니다.
또한, 보안 강화를 위해 TLS 설정을 적용하여 HTTPS 기반의 통신을 구성했습니다. 이를 위해 자체 서명 인증서(Self-signed TLS Certificate)를 생성하고 Secret 오브젝트로 등록한 뒤, Ingress 리소스에서 이를 참조하도록 설정했습니다.

2. 프로젝트 내용

본 프로젝트는 다음과 같은 순서로 진행되었습니다.

1. 쿠버네티스 클러스터 구성
2. 데이터베이스 이중화 서버 구성
3. 웹 서비스 구성
4. 모니터링

2.1. 쿠버네티스 클러스터 구성 (Kubespray)

이 절에서는 Vagrant를 활용하여 4개의 가상 머신(kube-control1, kube-node1, kube-node2, kube-node3)을 생성합니다. 이 가상 머신들은 쿠버네티스 클러스터의 노드로 사용됩니다. Vagrant를 사용하면 VM 환경을 코드로 정의할 수 있어 동일한 개발 및 테스트 환경을 반복적으로 손쉽게 구축할 수 있다는 장점이 있습니다.

이후, 생성된 가상 머신 위에서 Kubespray를 활용해 쿠버네티스 클러스터를 구성합니다.
Kubespray는 Ansible 기반의 설치 자동화 도구로, 복잡한 쿠버네티스 설치 과정을 단순화시켜 제공하며 다양한 설정을 코드로 관리할 수 있습니다. 이 도구를 통해 인증 방식, 애드온 설치를 세부적으로 설정하여 프로젝트에서 필요한 Helm, ingress nginx 등을 설치합니다.

2.1.1. Vagrant를 이용한 가상 머신 환경 구성

- **Vagrantfile 파일 작성**

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VM_IMAGE = "nobreak-labs/ubuntu-noble"
VM_NAME_PREFIX = "kube"
VM_SUBNET = "192.168.56."
VM_CONFIG_CONTROL_PLANE = {
  count: 1, cpus: 2, memory: 3072, extra_disk_enabled: false, extra_disk_size:
10 # GB
}
```

```
VM_CONFIG_WORKER_NODE = {
  count: 3, cpus: 2, memory: 2560, extra_disk_enabled: false, extra_disk_size:
10 # GB
}

CHANGE_APT_REPO = <<-SCRIPT
  ARCH=$(uname -m)
  RELEASE=$(lsb_release -cs)
  if [ "$RELEASE" = "jammy" ]; then
    # Ubuntu 22.04
    if [ "$ARCH" = "x86_64" ]; then
      sed -i
's/archive.ubuntu.com\\|kr.archive.ubuntu.com\\|security.ubuntu.com/ftp.kaist.
ac.kr/g' /etc/apt/sources.list
    elif [ "$ARCH" = "aarch64" ]; then
      sed -i 's/ports.ubuntu.com/ftp.kaist.ac.kr/g' /etc/apt/sources.list
    fi
  elif [ "$RELEASE" = "noble" ]; then
    # Ubuntu 24.04
    if [ "$ARCH" = "x86_64" ]; then
      sed -i 's|^URIs:.*|URIs: http://ftp.kaist.ac.kr/ubuntu/|g'
/etc/apt/sources.list.d/ubuntu.sources
    elif [ "$ARCH" = "aarch64" ]; then
      sed -i 's|^URIs:.*|URIs: http://ftp.kaist.ac.kr/ubuntu-ports/|g'
/etc/apt/sources.list.d/ubuntu.sources
    fi
  fi
SCRIPT

VMS = (1..VM_CONFIG_CONTROL_PLANE[:count]).map do |i|
  {
    name: "#{VM_NAME_PREFIX}-control#{i}",
    image: VM_IMAGE,
    cpus: VM_CONFIG_CONTROL_PLANE[:cpus],
    memory: VM_CONFIG_CONTROL_PLANE[:memory],
    extra_disk_enabled: VM_CONFIG_CONTROL_PLANE[:extra_disk_enabled],
    extra_disk_size: VM_CONFIG_CONTROL_PLANE[:extra_disk_size] * 1024,
    ip: "#{VM_SUBNET}1#{i}"
  }
end

VMS += (1..VM_CONFIG_WORKER_NODE[:count]).map do |i|
  {
    name: "#{VM_NAME_PREFIX}-node#{i}",
    image: VM_IMAGE,
    cpus: VM_CONFIG_WORKER_NODE[:cpus],
```

```
    memory: VM_CONFIG_WORKER_NODE[:memory],
    extra_disk_enabled: VM_CONFIG_WORKER_NODE[:extra_disk_enabled],
    extra_disk_size: VM_CONFIG_WORKER_NODE[:extra_disk_size] * 1024,
    ip: "#{VM_SUBNET}2#{i}"
  }
end

Vagrant.configure("2") do |config|
  VMS.each do |vm|
    config.vm.define vm[:name] do |node|
      node.vm.box = vm[:image]
      node.vm.provider "virtualbox" do |vb|
        vb.linked_clone = false
        vb.name = vm[:name]
        vb.cpus = vm[:cpus]
        vb.memory = vm[:memory]
        if vm[:extra_disk_enabled] == true
          disk_path = "disks/#{vm[:name]}.vmdk"
          unless File.exist?(disk_path)
            vb.customize ['createmedium', 'disk', '--format', 'VMDK', '--
filename', disk_path, '--size', vm[:extra_disk_size]]
          end
          vb.customize ['storageattach', :id, '--storagectl', 'VirtIO
Controller', '--port', 1, '--device', 0, '--type', 'hdd', '--medium',
disk_path]
        end
      end
      node.vm.hostname = vm[:name]
      node.vm.network "private_network", ip: vm[:ip], nic_type: "virtio"
      node.vm.synced_folder ".", "/vagrant", disabled: true

      node.vm.provision "shell", inline: CHANGE_APT_REPO
    end
  end
end
```

- **컨트롤 플레인 접속**

```
vagrant up
vagrant ssh kube-control1
```

2.1.2. kubspray를 활용한 쿠버네티스 설치

이전 단계에서 접속한 컨트롤 플레인(kube-control1)에서 진행합니다.

- **python 패키지 설치**

```
sudo apt update
sudo apt install -y python3 python3-pip python3-venv git
```

- **4개의 가상 머신과 SSH 통신을 위한 키 교환**

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -N ''
ssh-copy-id -i ~/.ssh/id_ed25519.pub vagrant@192.168.56.11
ssh-copy-id -i ~/.ssh/id_ed25519.pub vagrant@192.168.56.21
ssh-copy-id -i ~/.ssh/id_ed25519.pub vagrant@192.168.56.22
ssh-copy-id -i ~/.ssh/id_ed25519.pub vagrant@192.168.56.23
```

- **홈 디렉토리에서 kubespray git clone**

```
cd ~
git clone \
--branch release-2.27 https://github.com/kubernetes-sigs/kubespray.git
```

- **kubespray 가상 환경 접속**

```
python3 -m venv ~/kubespray
source ~/kubespray/bin/activate
cd ~/kubespray
```

- **필요한 패키지 설치**

```
pip install -U -r requirements.txt
```

- **Ansible 인벤토리 수정**

```
cp -rfp inventory/sample inventory/mycluster
vi inventory/mycluster/inventory.ini
```

- **inventory/mycluster/inventory.ini 파일**

```
[kube_control_plane]
kube-control1 ansible_host=192.168.56.11 ip=192.168.56.11
ansible_connection=local
```

```
[etcd:children]
kube_control_plane

[kube_node]
kube-node1 ansible_host=192.168.56.21 ip=192.168.56.21
kube-node2 ansible_host=192.168.56.22 ip=192.168.56.22
kube-node3 ansible_host=192.168.56.23 ip=192.168.56.23
```

- **애드온 추가 설정**

```
vi inventory/mycluster/group_vars/k8s_cluster/addons.yml
```

```
helm_enabled: true
metrics_server_enabled: true
ingress_nginx_enabled: true
metallb_enabled: true
metallb_config:
    address_pools:
        primary:
            ip_range:
                - 192.168.56.200-192.168.56.209
            auto_assign: true
    layer2:
        - primary
```

- 프로젝트에서 사용할 애드온(helm, metric server, ingress nginx, metallb)를 추가하는 설정을 합니다.
  - `helm_enabled` : 쿠버네티스 패키지 매니저 Helm을 사용할 수 있게 합니다. 본 프로젝트에서는 Prometheus, Grafana의 설치를 위해 사용됩니다.
  - `metrics_server_enabled` : 메트릭을 수집하는 metrics-server를 설치합니다. HPA가 동작하기 위해서 필수적인 요소입니다.
  - `ingress_nginx_enabled` : Ingress-NGINX Controller를 설치합니다. 클러스터 외에서 HTTP/HTTPS 요청을 받아서 내부 서비스로 전달하는 Ingress gateway 역할을 합니다.
  - `metallb_enabled` : MetalLB를 설치합니다. Bare Metal 환경에서는 클라우드 환경과는 다르게 LoadBalancer 서비스 타입이 작동하지 않는데, MetalLB를 사용하여 가상 IP를 제공해 LoadBalancer 서비스를 사용할 수 있게 합니다. ingress-nginx가가 LoadBalancer 타입으로 배포되기 위해 사용됩니다.
- **쿠버네티스 설정**

```
vi inventory/mycluster/group_vars/k8s_cluster/k8s-cluster.yml
```

```
kube_proxy_strict_arp: true
kube_encrypt_secret_data: true
 kube_encryption_algorithm: "aesgcm"
 kube_encryption_resources: [ secrets ]
```

- **Ansible 통신 확인**

```
ansible all -i inventory/mycluster/inventory.ini -m ping
```

- **플레이북 실행: 쿠버네티스 설치 자동화**

```
ansible-playbook -i inventory/mycluster/inventory.ini --become cluster.yml #
```

- 모든 설정이 완료되려면 15분에서 60분 정도 걸립니다.
- **자격 증명**

```
mkdir ~/.kube
sudo cp /etc/kubernetes/admin.conf ~/.kube/config
sudo chown $USER:$USER ~/.kube/config
```

- **자동 완성 설정**

```
source /etc/bash_completion.d/kubectl.sh
```

- **kubespray 가상 환경 빠져나오기**

```
deactivate
```

2.2. 데이터베이스 이중화 서버 구성

이 절에서는 고가용성을 위한 이중화된 데이터베이스 서버 환경을 구성합니다. 데이터베이스 이중화는 단일 장애 지점(SPOF, Single Point of Failure)을 제거함으로써 서비스의 안정성을 확보할 수 있는 방법입니다.

데이터베이스는 StatefulSet을 통해 배포되며, 이 중 첫 번째 파드는 읽기 및 쓰기가 모두 가능한 Primary 서버로 구성됩니다. 나머지 파드들은 읽기 전용 Replica 서버로 설정되어 읽기 요청을 분산 처리할 수 있도록 합니다.
쓰기 작업을 오직 Primary 서버에서만 수행하도록 제한하는 이유는, 다수의 파드에서 동시에 쓰기를 허용할 경우 데이터의 정합성과 일관성이 깨질 수 있기 때문입니다.

쓰기가 가능한 Primary 서버와 읽기 전용 Replica 서버의 역할을 구분 짓기 위한 서버 설정 구성 파일을 포함하는 ConfigMap을 생성합니다.
그리고 이 두 서버를 구별할 수 있도록 헤드리스 서비스를 생성하고 읽기 요청을 위한 별도의 ClusterIP 서비스를 만들어 읽기 작업을 효율적으로 분산시킬 수 있도록 합니다.

데이터베이스의 안정적인 저장 공간 확보를 위해 동적 볼륨 프로비저닝이 가능한 NFS 서버를 컨트롤 플레인에 구축하고 NFS 프로비저너를 클러스터에 배포합니다.

그리고 StatefulSet 오브젝트로 데이터베이스 서버를 배포합니다. 마지막으로, 웹 서버에서 사용할 wordpress 데이터베이스를 생성하는 Job 리소스를 배포합니다.

2.2.1. 데이터베이스 서버 설정 구성 파일 ConfigMap 생성

- **mysql-conf-cm.yml 파일**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-conf-cm
  labels:
    app.kubernetes.io/name: db
data:
  primary.cnf:
    [mysqld]
    log-bin
  replica.cnf:
    [mysqld]
    super-read-only
```

- primary.cnf
  - `log-bin` : 바이너리 로그 활성화 설정. 데이터 변경 사항을 기록하여 변경 내역을 복제 서버에 전달하기 위해서 필요한 설정.
- replica.cnf
  - `super-read-only` : 데이터 변경이 절대 불가능한 읽기 전용 설정.
- **ConfigMap 배포**

```
kubectl create -f mysql-conf-cm.yml
```

2.2.2. 헤드리스 서비스 및 ClusterIP 서비스 생성

- **mysql-svc.yml 파일**

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc
  labels:
    app.kubernetes.io/name: db
spec:
  ports:
    - name: mysql
      port: 3306
  clusterIP: None
  selector:
    app.kubernetes.io/name: db
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc-read
  labels:
    app.kubernetes.io/name: db
spec:
  ports:
    - name: mysql
      port: 3306
  selector:
    app.kubernetes.io/name: db
```

- **서비스 배포**

```
kubectl create -f mysql-svc.yml
```

2.2.3. 동적 볼륨 프로비저닝: NFS 서버 구성

- **NFS 서버 구성: 패키지 설치 및 공유 디렉토리 생성**

```
sudo apt update
sudo apt install -y nfs-kernel-server &> /dev/null
sudo mkdir /srv/nfs-volume
```

- **NFS 서버 구성: /etc/exports 파일**

```
/srv/nfs-volume *(rw,sync,no_root_squash,no_subtree_check)
```

- 설정 내용
  - `/srv/nfs-volume` 디렉토리를 대상으로 설정
  - `*` : 모든 호스트에 대해 접근 허용
  - `rw` : 읽기 및 쓰기 권한 허용
  - `sync` : 클라이언트 요청 시 데이터가 디스크에 동기화되어 기록됨 (데이터 안정성 보장)
  - `no_root_squash` : 클라이언트의 root 권한을 서버에서도 유지 (root 권한 무시하지 않음)
  - `no_subtree_check` : 서브 디렉토리까지 전체 공유
- **NFS 서버 구성: 공유 시작 및 확인**

```
sudo exportfs -r
sudo exportfs -v
```

- **NFS 클라이언트 구성: kubespray 환경 접속**

```
source ~/kubespray/bin/activate
```

- **NFS 클라이언트 구성: Ansible 통신 확인**

```
ansible -i ~/kubespray/inventory/mycluster/inventory.ini kube_node -m ping
```

- **NFS 클라이언트 구성: 모든 노드 nfs-common(클라이언트) 패키지 설치**

```
ansible -i ~/kubespray/inventory/mycluster/inventory.ini kube_node -m apt -a
'name=nfs-common state=latest' -b
```

- **NFS 클라이언트 구성: 가상 환경 빠져나오기**

```
deactivate
```

2.2.4. 동적 볼륨 프로비저닝: NFS 프로비저너 배포

- **NFS 프로비저너 GitHub 저장소 복제**

```
git clone https://github.com/kubernetes-sigs/nfs-subdir-external-
provisioner.git
```

- nfs-subdir-external-provisioner/deploy/ 디렉토리

    - `class.yaml` : StorageClass 리소스를 정의하는 파일입니다. NFS 프로비저너가 동적 볼륨 프로비저닝을 수행할 때 사용하는 StorageClass입니다. PVC가 이 StorageClass를 참조하면 NFS 프로비저너가 자동으로 PV를 생성합니다.
    - `deployment.yaml` : NFS 프로비저너를 배포하는 Deployment 리소스를 정의합니다. NFS 프로비저너는 PVC 요청이 들어오면 NFS 서버에 서브 디렉토리를 생성해 볼륨으로 할당합니다.
    - `rbac.yaml` : Role-Based Access Control 권한 설정 파일입니다. NFS 프로비저너가 필요한 권한을 가질 수 있도록 설정합니다.
    - `kustomization.yaml` : class.yaml, deployment.yaml, rbac.yaml 파일을 Kustomize 도구를 사용하여 한번에 배포할 수 있는 파일입니다.
    - `나머지 파일` : 배포에 필요한 추가 리소스 디렉토리와 테스트용 파일들입니다. 본 프로젝트에서는 사용하지 않습니다.
    - 이 디렉토리의 deployment.yaml, class.yaml 파일을 수정합니다.
- **deployment.yaml 파일 수정**

```
env:
    - name: NFS_SERVER
      value: 192.168.56.11
    - name: NFS_PATH
      value: /srv/nfs-volume

volumes:
  - name: nfs-client-root
    nfs:
      server: 192.168.56.11
      path: /srv/nfs-volume
```

- `spec.env` 항목에서 환경 변수 값을 아래와 같이 알맞게 수정합니다.
    - `NFS_SERVER=192.168.56.11`
      → NFS 서버가 실행 중인 컨트롤 플레인 노드의 IP 주소
    - `NFS_PATH=/srv/nfs-volume`
      → NFS 서버에서 공유된 디렉토리 경로
- **class.yaml 파일 수정**

```
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
```

- 배포될 nfs-client StorageClass를 기본 StorageClass로 설정하기 위해 annotation을 추가합니다.
- **NFS 프로비저너 배포**

```
kubectl create -k nfs-subdir-external-provisioner/deploy/
```

- **배포된 리소스 확인**

```
kubectl get deploy,sc,roles.rbac.authorization.k8s.io
```

2.2.5. 데이터베이스 서버 구성

- **mysql.yml 파일**

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  labels:
    app.kubernetes.io/name: db
spec:
  replicas: 2
  selector:
    matchLabels:
      app.kubernetes.io/name: db
  serviceName: mysql-svc
  template:
    metadata:
      labels:
        app.kubernetes.io/name: db
    spec:
      initContainers:
        - name: init-mysql
          image: mysql:5.7
          command:
            - bash
            - "-c"
            - |
              set -ex
              [[ $HOSTNAME =~ -([0-9]+)$ ]] || exit 1
              ordinal=${BASH_REMATCH[1]}
              echo [mysqld] > /mnt/conf.d/server-id.cnf
              echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/server-id.cnf
              if [[ $ordinal -eq 0 ]]; then
                cp /mnt/config-map/primary.cnf /mnt/conf.d/
              else
                cp /mnt/config-map/replica.cnf /mnt/conf.d/
              fi
```

```
          volumeMounts:
            - name: conf
              mountPath: /mnt/conf.d
            - name: config-map
              mountPath: /mnt/config-map
        - name: clone-mysql
          image: gcr.io/google-samples/xtrabackup:1.0
          command:
            - bash
            - "-c"
            - |
              set -ex
              [[ -d /var/lib/mysql/mysql ]] && exit 0
              [[ $HOSTNAME =~ -([0-9]+)$ ]] || exit 1
              ordinal=${BASH_REMATCH[1]}
              [[ $ordinal -eq 0 ]] && exit 0
              ncat --recv-only mysql-$(($ordinal-1)).mysql-svc 3307 | xbstream
-x -C /var/lib/mysql
              xtrabackup --prepare --target-dir=/var/lib/mysql
          volumeMounts:
            - name: data
              mountPath: /var/lib/mysql
              subPath: mysql
            - name: conf
              mountPath: /etc/mysql/conf.d
      containers:
        - name: mysql
          image: mysql:5.7
          env:
            - name: MYSQL_ALLOW_EMPTY_PASSWORD
              value: "1"
          ports:
            - name: mysql
              containerPort: 3306
          volumeMounts:
            - name: data
              mountPath: /var/lib/mysql
              subPath: mysql
            - name: conf
              mountPath: /etc/mysql/conf.d
          livenessProbe:
            exec:
              command: ["mysqladmin", "ping"]
            initialDelaySeconds: 30
            periodSeconds: 10
            timeoutSeconds: 5
```

```
          readinessProbe:
            exec:
              command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]
            initialDelaySeconds: 15
            periodSeconds: 2
            timeoutSeconds: 1
        - name: xtrabackup
          image: gcr.io/google-samples/xtrabackup:1.0
          ports:
            - name: xtrabackup
              containerPort: 3307
          command:
            - bash
            - "-c"
            - |
              set -ex
              cd /var/lib/mysql

              if [[ -f xtrabackup_slave_info && "x$(<xtrabackup_slave_info)"
!= "x" ]]; then
                cat xtrabackup_slave_info | sed -E 's/;$//g' >
change_master_to.sql.in
                rm -f xtrabackup_slave_info xtrabackup_binlog_info
              elif [[ -f xtrabackup_binlog_info ]]; then
                [[ `cat xtrabackup_binlog_info` =~ ^(.*?)[[:space:]]+(.*?)$ ]]
|| exit 1
                rm -f xtrabackup_binlog_info xtrabackup_slave_info
                echo "CHANGE MASTER TO MASTER_LOG_FILE='${BASH_REMATCH[1]}', \
                MASTER_LOG_POS=${BASH_REMATCH[2]}" > change_master_to.sql.in
              fi

              if [[ -f change_master_to.sql.in ]]; then
                echo "Waiting for mysqld to e ready (accepting connections)"
                until mysql -h 127.0.0.1 -e "SELECT 1"; do sleep 1; done

                echo "Initializing replication from clone position"
                mysql -h 127.0.0.1 -e "$(<change_master_to.sql.in), \
                    MASTER_HOST= 'mysql-0.mysql-svc', \
                    MASTER_USER='root', \
                    MASTER_PASSWORD='', \
                    MASTER_CONNECT_RETRY=10; \
                    START SLAVE;" || exit 1
                mv change_master_to.sql.in change_master_to.sql.orig
              fi

              exec ncat --listen --keep-open --send-only --max-conns=1 3307 -c
```

```
\
                "xtrabackup --backup --slave-info --stream=xbstream --
host=127.0.0.1 --user=root"
          volumeMounts:
            - name: data
              mountPath: /var/lib/mysql
              subPath: mysql
            - name: conf
              mountPath: /etc/mysql/conf.d
      volumes:
        - name: conf
          emptyDir: {}
        - name: config-map
          configMap:
            name: mysql-conf-cm
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        storageClassName: nfs-client
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 1Gi
```

- **mysql StatefulSet 각 컨테이너의 기능**
  - `init 컨테이너` : 파드가 생성될 때 메인 컨테이너보다 먼저 실행되는 초기화 컨테이너입니다.
    - **`init-mysql`**
      StatefulSet의 파드는 StatefulSet 이름 뒤에 인덱스 순서대로 지정되는 원리를 이용해 파드의 이름에 포함된 숫자를 기준으로 Primary(0)과 Replica(1 이상상)을 구분하고 이에 맞는 적절한 설정 파일을 사용해 파드의 MySQL을 설정합니다.
      또한, server-id 값을 각 파드에 고유하게 부여하여 MySQL 복제 시 사용할 서버 식별 번호를 설정합니다.
    - **`clone-mysql`**
      이 컨테이너는 파드 이름에서 숫자를 추출하여 Primary(0)가 아닌 경우에만 실행되는 초기화 컨테이너입니다.
      먼저 해당 파드의 데이터 디렉토리에 MySQL 데이터가 존재하는지 확인하고 이미 존재할 경우에는 동작을 종료합니다.
      만약 데이터가 없다면, init-mysql에서 부여한 server-id를 기반으로 자신보다 앞 순번(n-1)의 파드(예: 현재 파드가 mysql-1이라면 mysql-0)에 TCP 연결을 시도합니다.
      ncat 명령어를 통해 3307 포트로 전달되는 xtrabackup 데이터 스트림을 수신하고 압축을 해제하여 /var/lib/mysql 경로에 복제합니다.

마지막으로 xtrabackup --prepare 명령어를 실행해 복제된 데이터를 복구 가능한 상태로 준비하도록 합니다.

- `컨테이너` : 파드의 메인 컨테이너입니다.
  - **`mysql`**
    이 컨테이너는 파드의 메인 컨테이너로, MySQL 서버를 실행합니다.
    실습 환경인 본 프로젝트에서는 빠른 실행을 위해 MYSQL_ALLOW_EMPTY_PASSWORD 환경 변수를 1로 설정하여 비밀번호 없이 MySQL에 접근 가능하도록 설정하였지만 실제 운영 환경에서는 이 설정 없이 비밀번호를 설정하여 보안을 강화해야 합니다.
    이전 단계에서 init 컨테이너가 복제해 둔 데이터 디렉토리(/var/lib/mysql)와 설정 파일(/etc/mysql/conf.d)을 볼륨으로 마운트하여 MySQL이 적절한 설정과 데이터를 가지도록 구성합니다.
    또한 livenessProbe와 readinessProbe가 설정됩니다.
    - `livenessProbe` 는 서버가 정상적으로 살아 있는지 주기적으로 확인하여 비정상인 경우 재시작 정책을 적용합니다.
    - `readinessProbe` 는 서버가 요청을 받을 준비가 되었는지 확인하여, 준비되지 않은 상태일 경우 서비스 엔드포인트에 등록되지 않도록 합니다.
  - **`xtrabackup`**
    이 컨테이너는 Replica 파드에서 MySQL 복제를 자동으로 구성하고 자신의 데이터를 다음 Replica에 전달하는 역할을 수행합니다.
    먼저 이전 단계에서 복제한 데이터 내부에 존재하는 `xtrabackup_slave_info` 또는 `xtrabackup_binlog_info` 파일을 분석해 복제 시작 지점 정보(CHANGE MASTER TO 쿼리)를 생성하고, MySQL이 준비될 때까지 대기한 뒤 복제를 시작합니다.
    그런 다음, 자신의 MySQL 데이터를 `xtrabackup` 명령어로 스트리밍하여 다음 파드가 받을 수 있도록 3307 포트로 송출합니다.
    이를 통해 후속 Replica 파드가 복제 데이터를 받을 수 있도록 백업 서버 역할도 함께 수행합니다.

- **MySQL StatefulSet 배포**

```
kubectl create -f mysql.yml
```

- **create-wordpress-db.yml 파일**

```
apiVersion: batch/v1
kind: Job
metadata:
  name: create-wordpress-db
spec:
  template:
    spec:
```

```
      containers:
        - name: create-db
          image: mysql:5.7
          command:
            - bash
            - "-c"
            - |
              echo "Waiting for MySQL to be ready..."
              until mysqladmin ping -h mysql-0.mysql-
svc.default.svc.cluster.local --silent; do
                sleep 2
              done
              echo "Creating wordpress database..."
              mysql -h mysql-0.mysql-svc.default.svc.cluster.local -uroot -e
"CREATE DATABASE IF NOT EXISTS wordpress;"
      restartPolicy: OnFailure
```

- 데이터베이스 Primary 서버에 접속하여 wordpress 데이터베이스를 생성하는 Job을 수행합니다.
- **Job 배포**

```
kubectl create -f create-wordpress-db.yml
```

2.3. 웹 서비스 구성

이 절에서는 WordPress를 활용하여 웹 서비스를 구성합니다. WordPress는 PHP 기반의 CMS로 웹 사이트를 쉽고 빠르게 구축할 수 있습니다. 데이터베이스 서버와 연결하는 설정을 가지고 있어 실습에 사용되었습니다.

WordPress가 데이터베이스 서버와 연동될 때 필요한 환경 변수들을 정의하기 위해 ConfigMap을 생성합니다. 데이터베이스명, 호스트명, 사용자명, 비밀번호의 정보를 포함하고 있어 WordPress 컨테이너에 환경 변수로 설정됩니다. 이처럼 ConfigMap을 사용하여 환경 변수를 설정하면 데이터베이스 정보가 변경되어도 이미지를 변경하지 않을 수 있어 유연성이 증가합니다.

파드의 CPU 리소스 사용량에 기반한 HPA(Horizontal Pod Autoscaler)를 설정합니다. 최소 2개에서 최대 5개까지 파드가 자동으로 스케일 아웃/인 되도록 지정하여 트래픽 변화에 따라 적절한 자원을 사용합니다. 이 설정을 위해 WordPress Deployment 리소스 내 파드에 CPU 요청(request)과 제한(limit)을 설정하여 HPA가 metric-server에서 수집한 메트릭을 토대로 파드 수를 적절히히 조정할 수 있게 합니다.

다음으로 WordPress를 실행할 파드를 관리하는 Deployment를 생성합니다. WordPress 공식 이미지를 사용하며 앞서 생성한 ConfigMap을 참조해 DB 정보를 입력받고 CPU 리소스 요청 및 제한을 설정합니다.

마지막으로 클러스터 내에서 WordPress 파드에 접근을 편리하게 하는 ClusterIP 서비스를 생성하고 외부 요청이 라우팅될 수 있도록 Ingress 리소스와 서비스를 연결합니다. Ingress는 도메인 기반 라우팅과 TLS 종료 등의 기능을 포함합니다.

2.3.1. 데이터베이스 정보 환경 변수 ConfigMap 생성

- **wordpress-env-cm.yml 파일**

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: wordpress-env-cm
data:
  WORDPRESS_DB_HOST: mysql-0.mysql-svc.default.svc.cluster.local
  WORDPRESS_DB_USER: root
  WORDPRESS_DB_PASSWORD: ''
  WORDPRESS_DB_NAME: wordpress
```

- **ConfigMap 배포**

```
kubectl create -f wordpress-env-cm.yml
```

2.3.2. 오토스케일링: HPA 생성

- **wordpress-hpa-cpu.yml 파일**

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: wordpress-hpa-cpu
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: wordpress
  minReplicas: 2
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
```

```
      type: Utilization
      averageUtilization: 70
```

- **HPA 배포**

```
kubectl create -f wordpress-hpa-cpu.yml
```

2.3.3. 웹 서비스 구성

- **wordpress.yml 파일**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
spec:
  replicas: 3
  selector:
    matchLabels:
      app.kubernetes.io/name: wordpress
  template:
    metadata:
      labels:
        app.kubernetes.io/name: wordpress
    spec:
      containers:
        - name: wordpress
          image: wordpress:6.8.1-php8.1-apache
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 500m
              memory: 256Mi
          ports:
            - containerPort: 80
              protocol: TCP
          envFrom:
            - configMapRef:
                name: wordpress-env-cm
```

- **Deployment 배포**

```
kubectl create -f wordpress.yml
```

2.3.4. ClusterIP 서비스 및 Ingress 생성

- **ClusterIP 서비스: wordpress-svc.yml 파일**

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-svc
  labels:
    app.kubernetes.io/name: wordpress
spec:
  ports:
    - name: wordpress
      port: 80
  selector:
    app.kubernetes.io/name: wordpress
```

- **서비스 배포**

```
kubectl create -f wordpress-svc.yml
```

- **TLS 인증서와 키를 저장할 tls 디렉토리 생성**

```
mkdir tls
```

- **TLS 인증서와 키 생성**

```
openssl genrsa -out tls/proj-tls.key 2048
openssl req -new -x509 -key tls/proj-tls.key -out tls/proj-tls.crt -days 3650
-subj /CN=cccr-miniproject.com
```

- **TLS 시크릿 생성**

```
kubectl create secret tls tls-secret --cert=tls/proj-tls.crt --key=tls/proj-
tls.key
```

- **Ingress: ing.yml 파일**

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ing
spec:
  tls:
    - hosts:
      - cccr-miniproject.com
      secretName: tls-secret
  ingressClassName: nginx
  rules:
    - host: cccr-miniproject.com
      http:
        paths:
          - backend:
              service:
                name: wordpress-svc
                port:
                  number: 80
            path: /
            pathType: Prefix
```

- **Ingress 배포**

```
kubectl create -f ing.yml
```

- **Ingress Class 서비스 주소 확인**

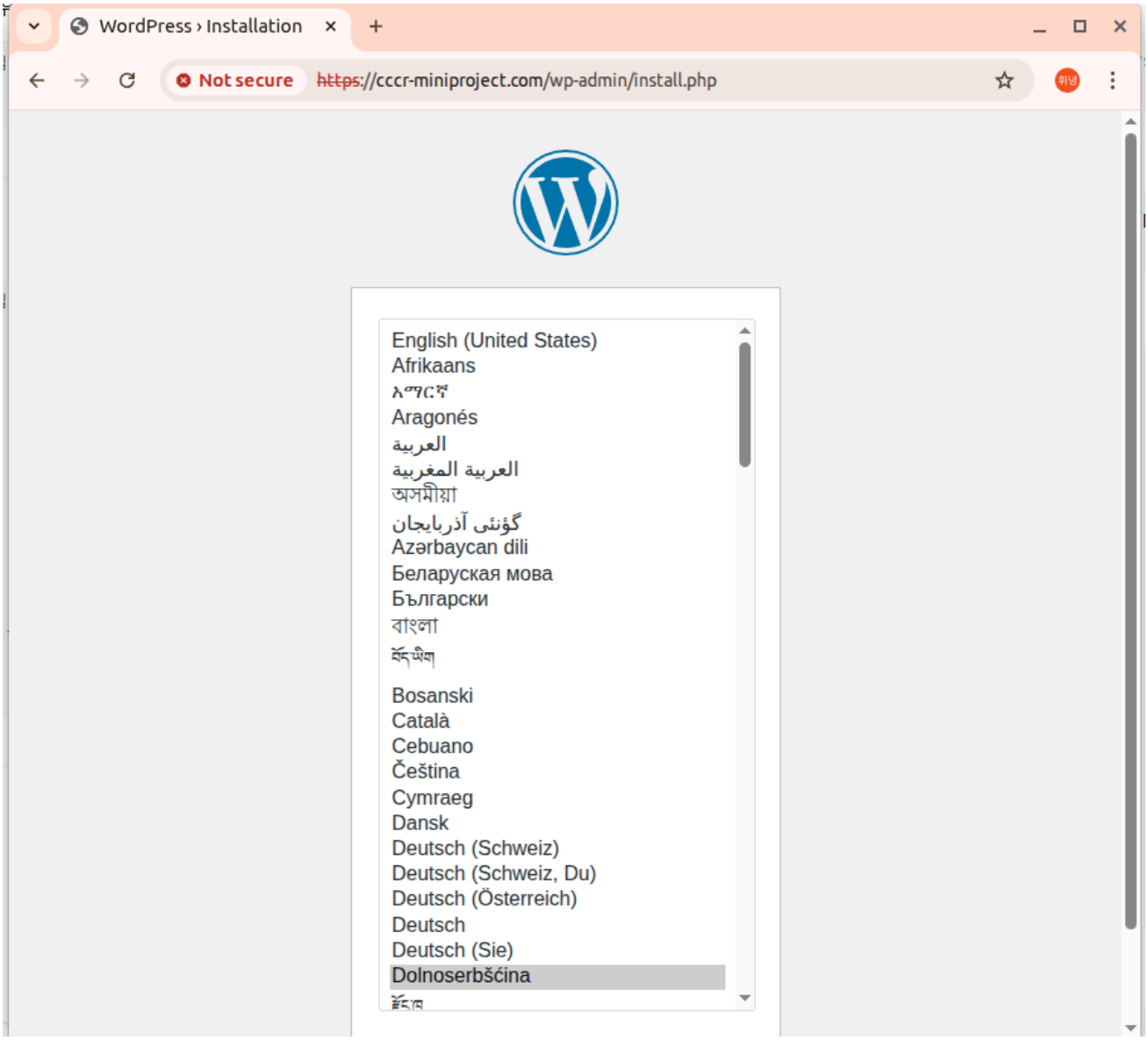
```
kubectl get svc -n ingress-nginx
```

- **호스트의 /etc/hosts 도메인 설정**

```
192.168.56.200 cccr-miniproject.com
```

- 확인한 Ingress Class 서비스의 External IP를 작성한다.

- **웹 브라우저 접속**



- 웹 브라우저에 https://cccr-miniproject.com 을 입력하면 WordPress에 접속이 가능합니다.

2.4. 모니터링

이 절에서는 Prometheus와 Grafana를 사용하여 웹 서비스의 사용량을 모니터링할 수 있는 환경을 구성합니다.

Prometheus는 쿠버네티스 클러스터의 메트릭 데이터를 수집하고 저장하는 역할을 하며 Grafana는 Prometheus가 수집한 데이터를 시각화하여 대시보드 형태로 제공하는 도구입니다.
이 두 도구를 함께 사용하여 클러스터 및 애플리케이션 리소스 사용량을 실시간으로 모니터링하고 파악할 수 있습니다.
Prometheus와 Grafana를 설치하고 구성하기 위해 Helm 패키지 매니저를 사용합니다.

2.4.1. 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
```

2.4.2. Prometheus 및 grafana 구성

- **모니터링 리소스를 위한 monitor 네임스페이스 생성**

```
kubectl create ns monitor
```

- **prometheus-community Helm 차트 저장소 추가**

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

- **grafana 서비스 리소스 파일 생성**

```
grafana:
  service:
    type: LoadBalancer
```

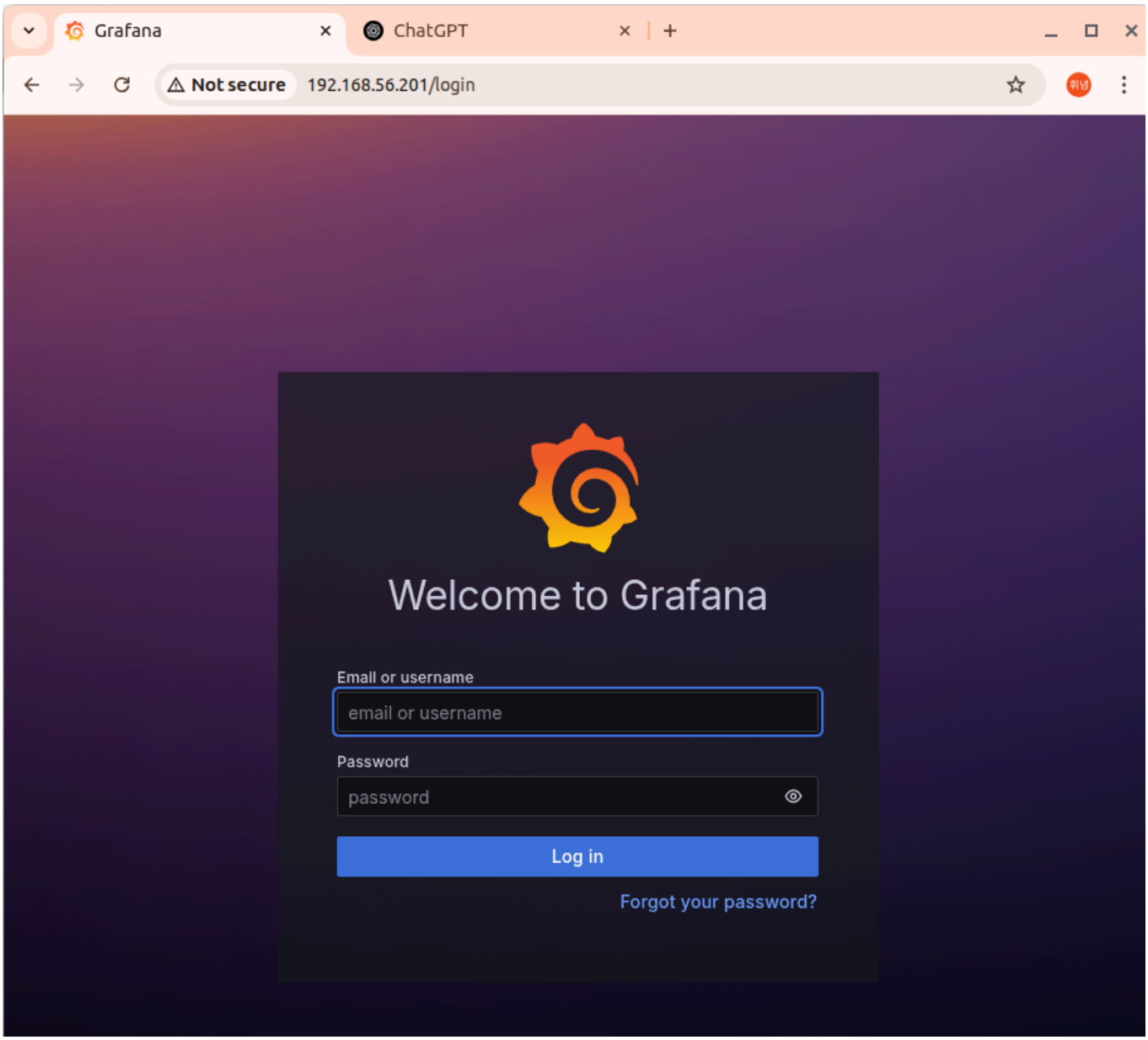
- **Helm 차트 릴리즈 생성**

```
helm install prom prometheus-community/kube-prometheus-stack -f prometheus-
grafana.yml -n monitor
```

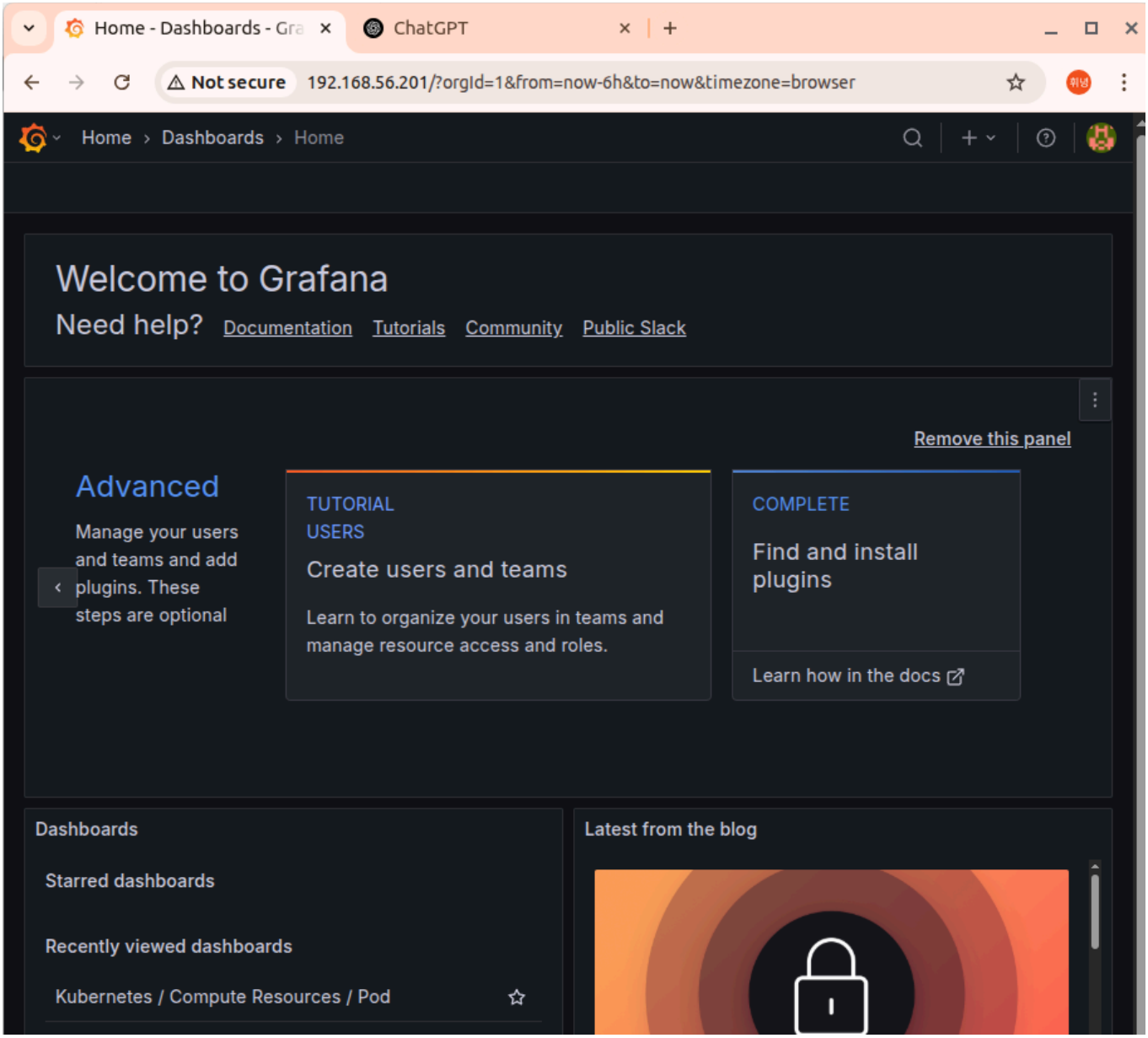
- **prom-grafana 서비스를 통해 접속**

```
$ kubectl get svc -n monitor
NAME                                    TYPE           CLUSTER-IP
EXTERNAL-IP      PORT(S)                      AGE
service/prom-grafana                    LoadBalancer   10.233.36.119
192.168.56.201   80:32713/TCP
```

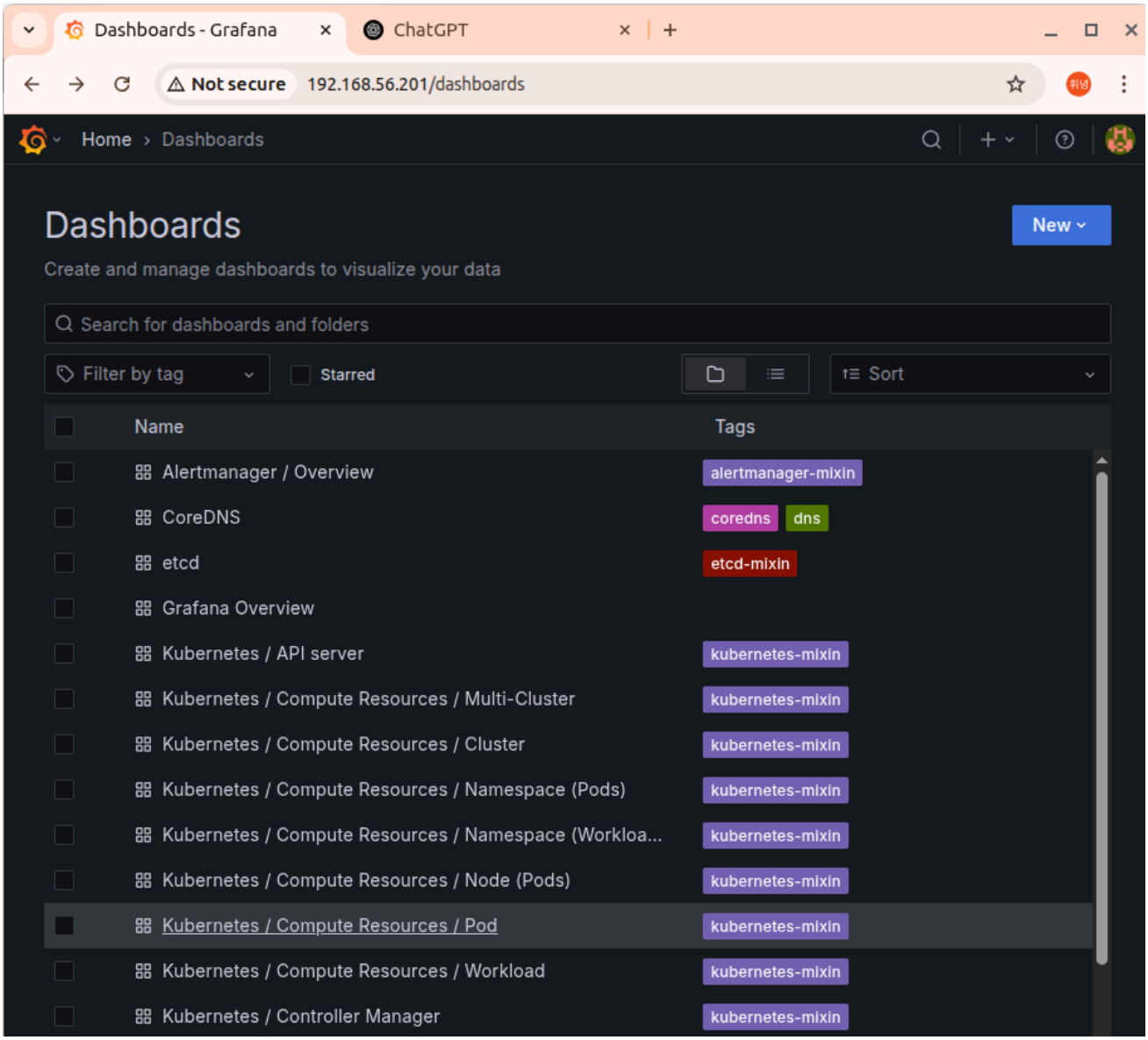
- prom-grafana 서비스의 External IP를 웹 브라우저에 입력하면 grafana 페이지에 접속할 수 있습니다.

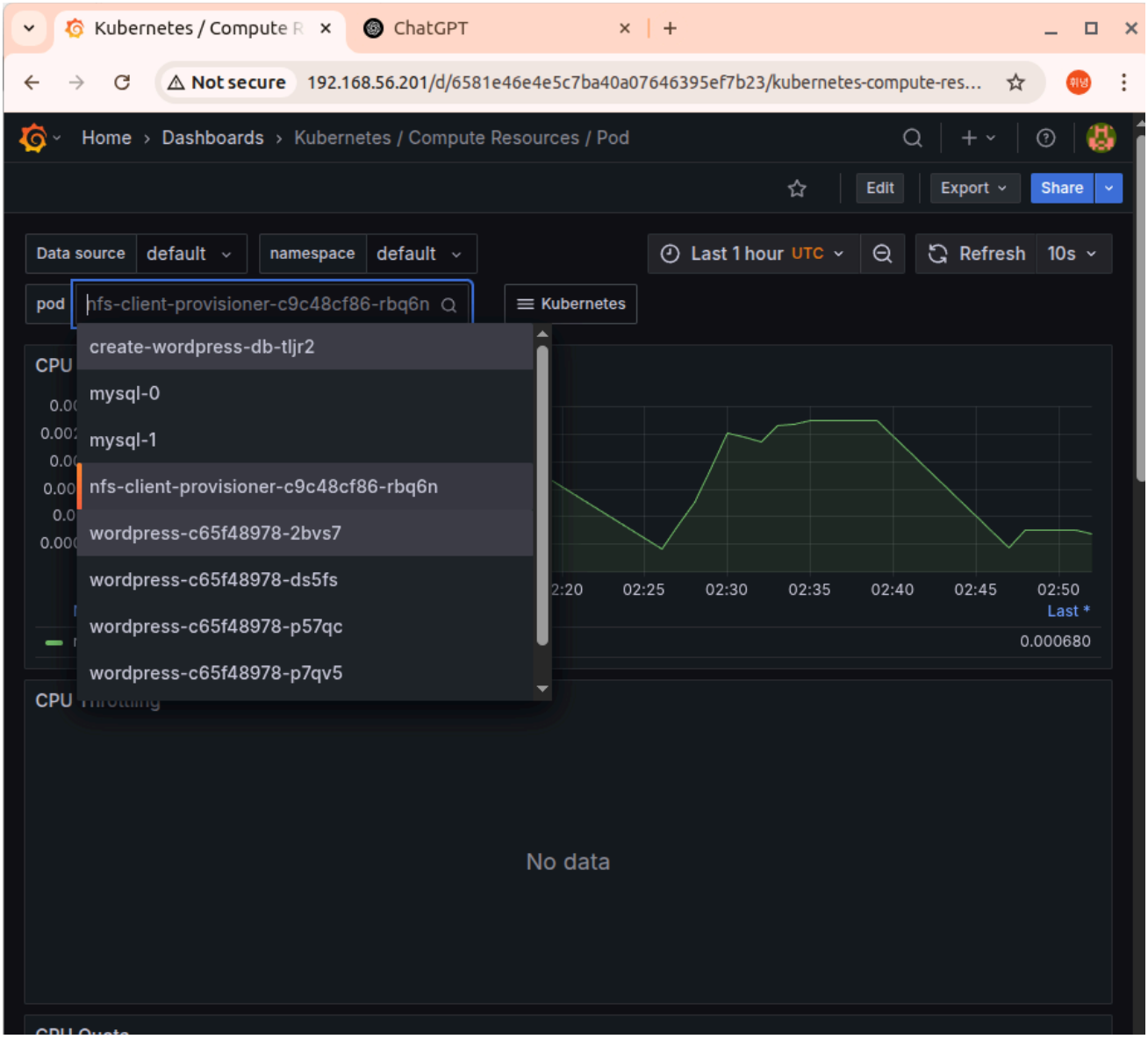


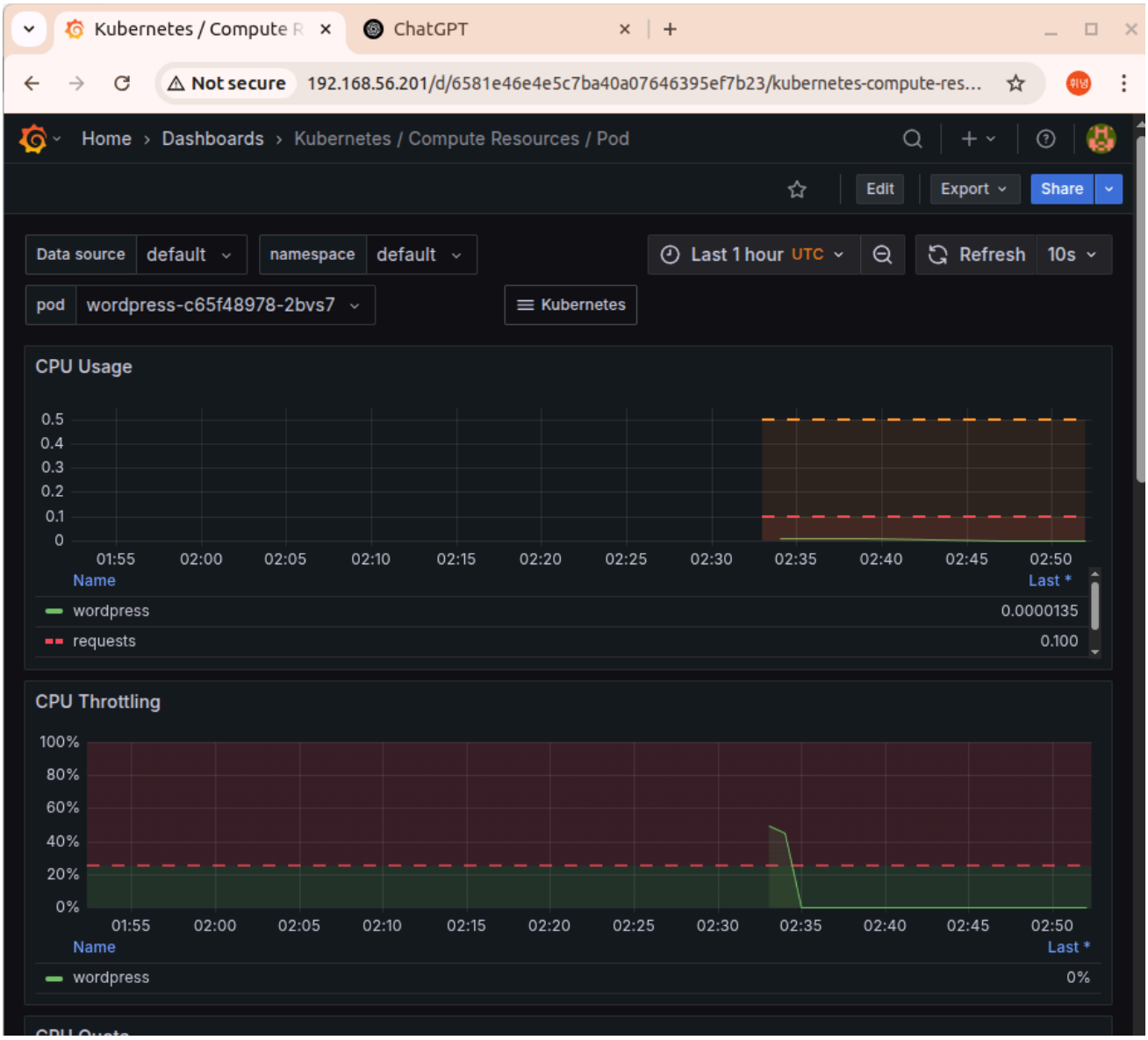
- 기본 관리자 계정 정보는 다음과 같습니다.
  - 아이디: admin
  - 비밀번호: prom-operator


Home - Dashboards - Gra
ChatGPT
Not secure
192.168.56.201/?orgId=1&from=now-6h&to=now&timezone=browser
Home › Dashboards › Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Advanced
Manage your users and teams and add plugins. These steps are optional
TUTORIAL
USERS
Create users and teams
Learn to organize your users in teams and manage resource access and roles.
COMPLETE
Find and install plugins
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Kubernetes / Compute Resources / Pod
Latest from the blog

- **wordpress 파드 모니터링**




Kubernetes / Compute R
ChatGPT
Not secure
192.168.56.201/d/6581e46e4e5c7ba40a07646395ef7b23/kubernetes-compute-res...
Home › Dashboards › Kubernetes / Compute Resources / Pod
Edit
Export
Share
Data source
default
namespace
default
Last 1 hour UTC
Refresh
10s
pod
nfs-client-provisioner-c9c48cf86-rbq6n
Kubernetes
create-wordpress-db-tljr2
mysql-0
mysql-1
nfs-client-provisioner-c9c48cf86-rbq6n
wordpress-c65f48978-2bvs7
wordpress-c65f48978-ds5fs
wordpress-c65f48978-p57qc
wordpress-c65f48978-p7qv5
CPU
02:20
02:25
02:30
02:35
02:40
02:45
02:50
Last *
0.000680
No data



3. 프로젝트 결과

3.1. 트러블슈팅

3.1.1. 동적 프로비저닝 바인딩 실패

문제 배경

데이터베이스 StatefulSet에서 PVC를 배포하였지만 PV가 생성되지 않아 계속 Pending 상태가 되는 문제가 발생했습니다.

```
$ kubectl describe pvc data-mysql-0
  Type    Reason                Age                   From
Message
  ----    ------                ----                  ----
-------
```

```
  Normal  ExternalProvisioning  2m53s (x42 over 13m)  persistentvolume-
controller  Waiting for a volume to be created either by the external
provisioner 'k8s-sigs.io/nfs-subdir-external-provisioner' or manually by the
system administrator. If volume creation is delayed, please verify that the
provisioner is running and correctly registered.
```

볼륨이 생성되지 않고 있으니 프로비저너가 제대로 동작하고 있는지 확인하라는 오류 메시지를 확인했습니다.

해결 과정
NFS 프로비저너에 문제가 있다고 판단하고 GitHub 저장소를 다시 불러와 kustomization.yaml 파일을 다시 배포했습니다. 하지만 PV는 여전히 생성되지 않았습니다.

```
$ kubectl get pod
NAME                                     READY   STATUS              RESTARTS
AGE
mysql-0                                  0/2     Pending             0
34m
nfs-client-provisioner-6f9c66997d-r4kgs   0/1     ContainerCreating   0
34m
```

NFS 프로비저너가 ContainerCreating 상태에 멈춰 있는 것을 확인했습니다.

```
$ kubectl describe pod nfs-client-provisioner-6f9c66997d-r4kgs

Events:
  Type     Reason       Age                   From               Message
  ----     ------       ----                  ----               -------
  Normal   Scheduled    35m                   default-scheduler  Successfully
assigned default/nfs-client-provisioner-6f9c66997d-r4kgs to kube-node1
  Warning  FailedMount  2m58s (x10 over 32m)  kubelet
MountVolume.SetUp failed for volume "nfs-client-root" : mount failed: exit
status 32
Mounting command: mount
Mounting arguments: -t nfs 10.3.243.101:/ifs/kubernetes
/var/lib/kubelet/pods/bbec2b38-d2e9-4cb1-bc8f-
8c19a88e491e/volumes/kubernetes.io~nfs/nfs-client-root
Output: mount.nfs: Connection refused for 10.3.243.101:/ifs/kubernetes on
/var/lib/kubelet/pods/bbec2b38-d2e9-4cb1-bc8f-
8c19a88e491e/volumes/kubernetes.io~nfs/nfs-client-root
```

NFS 프로비저너의 Events를 확인해 보니 Mounting arguments에 10.3.254.101:/ifs/kubernetes를 마운팅 지점으로 인식하고 있었습니다.

NFS 프로비저너를 배포할 때, 볼륨 설정에 오류가 있었는지 확인했습니다.

```
vi deployment.yaml
```

deployment.yaml 파일을 확인해 보니 env 필드의 NFS_SERVER 환경 변수 값이 기본 값 (10.3.243.101) 그대로 설정되어 있었고 volumes 필드의 nfs-client-root 볼륨과 연결되는 nfs 서버, 경로 또한 기본 값으로 설정되어 있었습니다.
이 설정을 바꾸지 않고 그대로 배포해서 로컬의 NFS 서버에 접속이 안 되어 볼륨 생성이 불가능했던 것이었습니다.

```
env:
  - name: NFS_SERVER
    value: 192.168.56.1

volumes:
  - name: nfs-client-root
    nfs:
      server: 192.168.56.11
      path: /srv/nfs-volume
```

현재 설정대로 해당 부분을 수정하고 NFS 프로비저너를 재배포하니 제대로 동작하는 것을 확인할 수 있었습니다.

문제 해결 후 원인을 분석해보니, 처음에는 기존에 사용하던 NFS 프로비저너를 그대로 활용하려고 했었습니다. 하지만 여러 실습을 진행하면서 배포 중이던 모든 파드를 삭제하는 과정에서 NFS 프로비저너도 함께 삭제된 것으로 보입니다. 이로 인해 문제가 발생한 것으로 결론지었습니다.
원래는 쿠버네티스 클러스터 구성 등의 과정을 프로젝트 범위에 포함하지 않으려 했으나, 처음 상태부터의 과정을 기록하는 것이 중요하다고 판단하여 쿠버네티스 클러스터를 다시 생성하는 것부터 프로젝트를 시작하였습니다.

3.1.2. Calico unauthorized error

문제 배경
NFS 프로비저너를 배포했지만 파드의 상태가 계속 ContainerCreating으로 멈추는 현상이 발생했습니다.

```
$ kubectl get po
NAME                                     READY   STATUS              RESTARTS
```

```
AGE
nfs-client-provisioner-c9c48cf86-4jt6n   0/1     ContainerCreating   0
17m
```

이후로 데이터베이스 StatefulSet으로 생성한 파드와 웹 서비스 Deployment로 생성한 파드들 또한 같은 문제를 겪었습니다.

해결 과정
가장 먼저 파드의 Events를 확인해보았습니다.

```
$ kubectl describe pod nfs-client-provisioner-c9c48cf86-4jt6n
  Warning  FailedCreatePodSandBox  8m41s (x4 over 8m45s)   kubelet
(combined from similar events): Failed to create pod sandbox: rpc error: code
= Unknown desc = failed to setup network for sandbox
"1e80aabad43698206e3d00e4764144c17e0290d0406119862323a188472ad490": plugin
type="calico" failed (add): error getting ClusterInformation: connection is
unauthorized: Unauthorized
```

ClusterInformation을 가져오는 권한이 없어서 에러가 발생했다는 것으로 보입니다. calico와 연관되어 있다는 것도 알 수 있었습니다.

calico가 제대로 실행되고 있는지를 확인했습니다.

```
$ kubectl get po -n kube-system
NAME                                       READY   STATUS    RESTARTS
AGE
calico-node-2wg7n                          1/1     Running   2 (28h ago)
2d20h
calico-node-b9gmj                          1/1     Running   3 (28h ago)
2d20h
calico-node-prm2b                          1/1     Running   3 (43m ago)
2d20h
calico-node-wklhd                          1/1     Running   2 (28h ago)
2d20h
```

모든 파드가 정상적으로 실행 중이었습니다. 그렇다면 권한이 제대로 설정되어 있는지를 확인해야 했습니다.

- **calico-node의 서비스 어카운트 확인**
  Calico는 모든 노드에 하나씩 배포하기 위해 DaemonSet으로 구성되어 있습니다. 이 DaemonSet의 설정을 확인했습니다.

```
$ kubectl get daemonset calico-node -n kube-system -o yaml | grep
serviceAccountName
serviceAccountName: calico-node
```

calico-node가 사용하고 있는 서비스 어카운트는 calico-node입니다.

- **calico-node 서비스 어카운트와 바인딩된 ClusterRole 확인**
  ClusterRoleBinding 설정을 보고 calico-node 서비스 어카운트가 어떤 권한이 부여된 ClusterRole과 바인딩 되었는지를 확인합니다.

```
$ kubectl get clusterrolebindings.rbac.authorization.k8s.io calico-node -o
yaml
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-node
subjects:
- kind: ServiceAccount
  name: calico-node
  namespace: kube-system
```

calico-node라는 이름의 ClusterRole과 바인딩 되어 있습니다.

- **calico-node ClusterRole 권한 확인**

```
$ kubectl get clusterrole calico-node -o yaml
  - clusterinformations
```

calico-node ClusterRole의 정보를 YAML 파일로 확인하였고 clusterinformations이라는 권한이 부여되어 있는 것을 볼 수 있었습니다.
그렇다면 권한도 부여가 되어 있는데 왜 해당 권한이 없다는 에러가 생겼던 건지 검색을 해보았습니다.

검색 결과, 토큰이 만료되어 더 이상 해당 권한을 사용할 수 없는 문제가 있다는 것을 알게 되었습니다.
토큰을 재발급받기 위해 calico-node를 재생성했습니다.

- **calico-node 삭제**

```
$ kubectl delete po -n kube-system calico-node-2wg7n calico-node-b9gmj calico-
node-prm2b calico-node-wklhd
```

calico-node는 DaemonSet으로 관리되고 있기 때문에 파드를 삭제하면 새로운 파드가 재생성될 것입니다.

```
$ kubectl get po
NAME                                     READY   STATUS     RESTARTS   AGE

nfs-client-provisioner-c9c48cf86-4jt6n   1/1     Running    0          32m
```

calico-node를 재생성하고 나서 파드가 잘 생성되는 것을 확인할 수 있습니다.

3.2. 추후 발전 방향

3.2.1. 데이터베이스 이중화 활용 개선

현재 본 프로젝트에서는 이중화된 데이터베이스 서버를 실질적으로 활용하고 있지 않습니다. 데이터베이스 이중화의 이유와 원리를 이해하고 실습해 보기 위해 이중화 구성을 시도하긴 했으나, 본 프로젝트는 개발보다는 인프라 구축에 중점을 두고 있기 때문에, 이를 활용하기 위한 이미지 변경은 이루어지지 않았습니다.
데이터 베이스 연결을 가장 단순하게 확인할 수 있도록 WordPress 이미지를 활용하였으며, 이로 인해 데이터베이스에 대한 읽기와 쓰기 요청 간의 차이를 고려한 처리는 진행되지 않고 모든 요청을 Primary 데이터베이스로 전달하고 있습니다.

향후 개발까지 진행하게 된다면, WordPress 이미지를 대신해 웹 페이지를 구성하여 다음과 같은 방식으로 데이터베이스 이중화 구성의 실효성을 높일 수 있을 것입니다.

- 데이터 조회(Read)에 해당하는 읽기 요청은 ClusterIP 서비스를 통하여 Primary와 Replica 데이터베이스에 분산되어 전달되도록 하고
- 데이터 생성(Create), 수정(Update), 삭제(Delete)에 해당하는 쓰기 요청은 쓰기가 가능한 Primary 데이터베이스로 전달하도록 설정합니다.

이러한 구조를 구현하여 데이터베이스 부하 분산 효과와 시스템의 확장성 및 안정성을 향상시킬 수 있으며 단일 장애점을 제거해 장애 대응력 또한 강화될 수 있을 것입니다.

3.2.2. 로깅 추가 구성

현재 본 프로젝트에서는 모니터링은 구성되어 있으나 로깅은 구현되지 않았습니다. 이는 프로젝트 진행에 사용된 데스크탑 환경에서 메모리 리소스가 부족하여 로깅 시스템을 구성하는 데 필요한 파드까지 동시에 실행하기에는 무리가 있었기 때문입니다.

향후 로깅 구성을 추가적으로 진행한다면 웹 서비스 파드 및 데이터베이스 서버에서 발생하는 로그를 수집, 저장, 조회할 수 있는 중앙집중형 로깅 시스템을 다음과 같이 EFG 스택으로 구성할 것입니다.

- Fluent Bit를 각 노드에 DaemonSet 형태로 배포하여 각 노드의 /var/log/containers/ 경로에 있는 컨테이너 로그를 수집합니다.
- 수집된 로그는 검색 가능한 저장소인 Elasticsearch에 저장됩니다.
- Grafana에서 Elasticsearch를 로그 데이터 소스로 연결하여 웹 기반의 시각화된 로그 조회 환경을 제공합니다.

이러한 구성을 통해 웹 서비스 장애 분석, 성능 진단 등을 실시간 로그 분석을 통해 쉽게 진행할 수 있을 것입니다.

3.2.3. 전체 구현 Helm 차트화

현재 본 프로젝트에서는 모든 오브젝트를 직접 배포하고, 특히 시크릿 같은 경우에는 각 설정 값을 직접 작성해야 하는 번거로움이 있습니다. 이런 과정을 거치지 않고, 프로젝트 전체 구현을 Helm 차트로 실행한다면 훨씬 효율적으로 인프라를 배포할 수 있을 것입니다.

향후 Helm 차트화를 진행한다면 작성한 모든 쿠버네티스 오브젝트를 정리하고 helm create 명령어를 통해 기본 구조를 생성하고 변수들은 values.yaml에 정리해 helm install 명령어를 사용하여 배포할 것입니다.

이러한 구성을 통해 프로젝트 인프라 배포 자동화 및 일관성을 확보할 수 있고 수동 설정에서 생길 수 있는 실수를 줄이며 수정 또한 유연하게 이루어질 수 있을 것입니다.

3.3. 회고

이번 프로젝트에서는 계획을 최대한 작은 단위로 세분화하고, 매일 시작 전에 당일 목표를 명확하게 작성하고 이를 달성하고자 노력하였으며 하루를 마무리할 때는 다음 날의 구현 계획을 미리 세웠습니다.
이러한 방식으로 진행하니 의미 없이 흘러가는 시간일 줄어들고, 효율이 높아졌으며, 프로젝트도 여유 있게 마무리할 수 있었습니다.
앞으로 진행할 다른 프로젝트에서도 이 방법을 적용해 체계적으로 목표를 세워 진행해야겠다고 느꼈습니다.

또한, 처음에는 강의 실습에서 사용했던 쿠버네티스 클러스터를 그대로 활용했는데 3.1. 트러블슈팅에서 작성한 NFS 프로비저너 부재뿐만 아니라, LimitRange와 ResourceQuota 실습 후 이를 제대로 삭제하지 않아 발생한 OOMKilled 문제도 있었습니다.
이처럼 의도하지 않은 이전 설정이 남아 있을 경우, 그 원인을 정확히 인지하지 못하면 문제를 파악하기 어려울 수 있다는 점을 깨달았습니다. 실습 및 테스트 환경에서 설정한 내용은 작업 후 반드시 원래 상태로 되돌려야 하며, 쿠버네티스 내 오브젝트에 대한 명확한 이해가 중요하다는 것을 다시 한 번 느꼈습니다.