2024.07.25

오전

도커설치하기
curl -sSL get.docker.com | sh

https가 안될경우 보안그룹들어가서 추가하기, https 443 port

**- 도커의GPG KEY 추가

- 리포지토리 시스템 추가
- 버전확인
- 설치**
- **내 계정에 docker 그룹 포함하기
- docker 명령어를 실행할때 sudo를 사용하지 않아도 됨**
- **sudo usermod -aG docker ubuntu**

kubectl 설치
curl -LO https://dl.k8s.io/release/v1.21.7/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kustomize 설치

sudo mkdir /install_dir && cd /install_dir
sudo wget https://github.com/kubernetes-sigs/kustomize/releases/download/kustomize%2Fv4.5.6/kustomize_v4.5.6_linux_amd64.tar.gz
압축 해제
sudo tar -zxvf kustomize_v4.5.6_linux_amd64.tar.gz
sudo mv kustomize /usr/local/bin/kustomize
sudo chmod 777 /usr/local/bin/kustomize
kustomize version

minikube 설치
**cd # 홈 디렉토리로 이동
sudo wget https://github.com/kubernetes/minikube/releases/download/v1.24.0/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
ls -l /usr/local/bin/minikube
minikube version

minikube용 k8s 시작
**minikube start --cpus 8 --memory 26240 --disk-size=80g --kubernetes-version=v1.21.12
kubectl get all -n kube-system

권한에러시
sudo usermod -aG docker $USER && newgrp docker

k9s
wget https://github.com/derailed/k9s/releases/download/v0.13.7/k9s_0.13.7_Linux_i386.tar.gz
tar xvzf k9s_0.13.7_Linux_i386.tar.gz
sudo mv k9s /usr/bin**

kubeflow 설치
**cd # 홈 디렉토리로 이동
git clone -b v1.6.0 https://github.com/kubeflow/manifests.git
cd manifests

while ! kustomize build example | kubectl apply -f -; do echo "Retrying to apply resources";
sleep 10; done**

error 난 구성 제거
kubectl get pods -n kubeflow | grep Error | awk '{ print $1 }' | xargs kubectl delete pod -n kubeflow

sudo sysctl -w fs.inotify.max_user_watches=524288
sudo sysctl -w fs.inotify.max_user_instances=512

kubectl edit service istio-ingressgateway -n istio-system

61 type: LoadBalancer
-> 변경
61 type: NodePort

kubectl get svc -n istio-system
minikube service list
kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80 --address '0.0.0.0'

minikube stop
sudo shutdown -h now
sudo vim /etc/sysctl.conf

fs.inotify.max_user_watches=524288
fs.inotify.max_user_instances=512**

nohup kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80 --address '0.0.0.0' &

**sudo vim /etc/sysctl.conf

fs.inotify.max_user_watches=524288
fs.inotify.max_user_instances=512**

nohup kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80 --address '0.0.0.0' &

kubectl get namespace | grep kubeflow

vim profile.yaml
**apiVersion: kubeflow.org/v1beta1
kind: Profile
metadata:
name: namespace1 # replace with the name of profile you want, this will be user's namespace name
spec:
owner:
kind: User
name: user1@email.com # replace with the email of the user

  resourceQuotaSpec: # resource quota can be set optionally
hard:
cpu: "2"
memory: "2Gi"
requests.nvidia.com/gpu: "1"
persistentvolumeclaims: "1"
requests.storage: "5Gi"**

kubectl apply -f profile.yaml

**kubectl get profile -A
kubectl get all -n namespace1

hash 암호만들기
https://bcrypt-generator.com**

cd /home/ubuntu/manifests/common/dex/base
vim config-map.yaml

email
hash
username
userID
변경

kubectl delete deployments.apps dex -n auth
kustomize build ~/manifests/common/dex/overlays/istio | kubectl apply -f -

kubectl get deployments -n auth

계정삭제

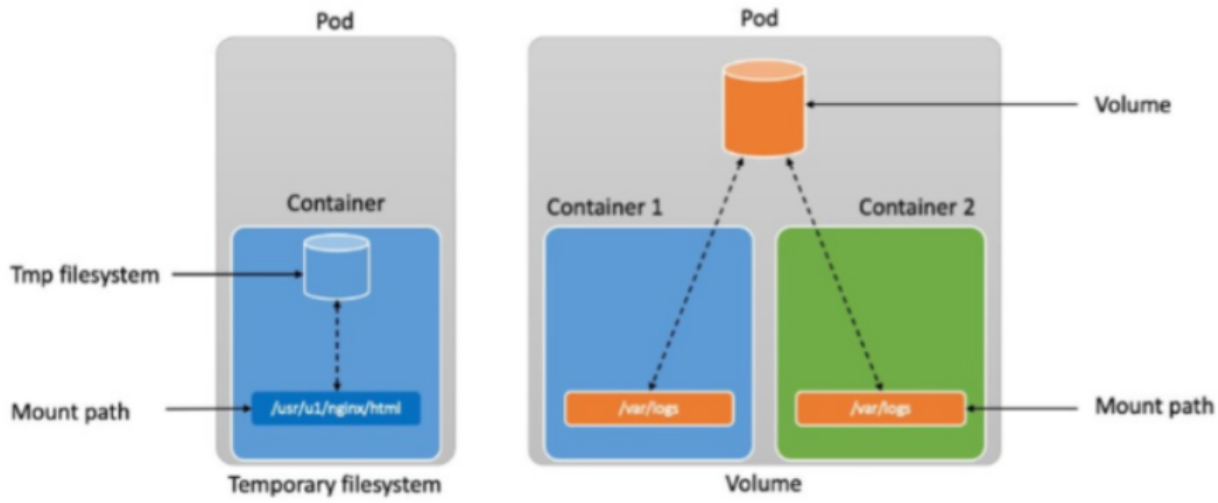
config-map.yaml
vim ~/manifests/common/dex/base/config-map.yaml

kustomize build ~/manifests/common/dex/overlays/istio | kubectl apply -f -

kubectl delete profile namespace1
kubectl get profile -A

kubectl get all -n namespace1
kubectl get namespace | grep namespace1

쿠버네티스 볼륨



임시 파일 시스템은 컨테이너의 수명 주기와 연결되어 있습니다. 임시 볼륨은 포드의 수명 주기에 연결됩니다.

임시 스토리지(Ephemeral storage)

컨테이너는 temporary filesystem(tmpfs)를 사용하여 파일을 읽고 쓸 수 있습니다. 그러나 임시 스토리지는 세 가지의 저장소 요구 사항을 충족하지 않습니다. 컨테이너가 충돌하는 경우 temporary filesystem은 손실되고, 컨테이너는 깨끗한 상태로 다시 시작됩니다. 또한 여러 컨테이너가 temporary filesystem을 공유할 수 없습니다.

임시 볼륨(Ephemeral volumes)

임시 Kubernetes Volume은 임시 스토리지가 직면한 두 가지 문제를 모두 해결합니다. 임시 Volume의 수명은 Pod와 연결됩니다. 이를 통해 컨테이너를 안전하게 재시작하고 Pod내의 컨테이너간 데이터를 공유할 수 있습니다. 그러나 Pod가 삭제되는 즉시 Volume도 삭제가 되므로, 이는 여전히 세 가지 요구 사항을 충족하지 못합니다.

스토리지와 포드 분리: 영구 볼륨(Persistent Volumes)

- Kubernetes는 Persistent Volumes도 지원합니다. Persistent Volumes을 사용하면 애플리케이션, 컨테이너, 포드, 노드 또는 클러스터 자체의 수명 주기와 관계없이 데이터가 지속됩니다.

영구 볼륨 클레임(Persistent volume claims)

- Persistent Volume(PV)은 실제 스토리지 볼륨을 나타냅니다. Kubernetes는 PV를 포드에 연결하는데 필요한 추가 추상화 계층인 PersistentVolumeClaim(PVC)을 가지고 있습니다.
- 간단히 말해서, Kubernetes는 PV객체는 클러스터 관리자 범위에 속하고, 반면에 PVC 객체는 애플리케이션 개발자 범위에 속해야 한다는 개념으로 구축되었습니다.

정적 프로비저닝(Static provisioning)

- "영구 볼륨 클레임" 섹션에서 설명한 바와 같이, 먼저 관리자가 하나 이상의 PV를 생성하고 애플리케이션 개발자는 PVC를 생성합니다. 이를 정적 프로비저닝이라고 합니다.
- Kubernetes에서 PV 및 PVC를 수동으로 만들어야 하므로 정적 입니다.
- 대규모 환경에서는 관리하기가 점점 더 어려워질 수 있으며, 특히 수백 개의 PV와 PVC를 관리하는 경우에는 더욱 그렇습니다.

동적 프로비저닝(Dynamic provisioning)

- 동적 프로비저닝을 사용하면 PV객체를 생성할 필요가 없습니다. 대신에, PVC를 생성할 때 내부적으로 자동으로 생성됩니다. Kubernetes는 Storage Class라는 다른 객체를 사용하여 이를 수행합니다.
- Storage Class는 컨테이너 애플리케이션에 사용되는 백엔드 영구 스토리지(예: Amazon EFS 파일 스토리지, Amazon EBS 블록 스토리지 등)의 클래스를 정의하는 추상화입니다.
- Pod 관점에서 볼 때, EFS 볼륨, EBS 볼륨, NFS 드라이브 또는 기타 어떤 것이든, 그 Pod는 PVC 객체만 보게 됩니다.
- 실제 스토리지 기술을 다루는 모든 내부적인 논리는 Storage Class 객체가 사용하는 프로비저너에 의해 구현됩니다.

**

sagemaker

notebook 생성

```
**

import torch
from torch import nn # nn contains all of PyTorch's building blocks for neural networks
import matplotlib.pyplot as plt

plt.ion()
# Check PyTorch version
torch.__version__



# Create *known* parameters

weight = 0.7

bias = 0.3



# Create data

start = 0
```

```
end = 1

step = 0.02

X = torch.arange(start, end, step).unsqueeze(dim=1)

y = weight * X + bias



X[:5], y[:5]



# Create train/test split

train_split = int(0.8 * len(X)) # 80% of data used for training set, 20% for
testing

X_train, y_train = X[:train_split], y[:train_split]

X_test, y_test = X[train_split:], y[train_split:]



len(X_train), len(y_train), len(X_test), len(y_test)



def plot_predictions(

    train_data=X_train,

    train_labels=y_train,

    test_data=X_test,

    test_labels=y_test,

    predictions=None):
```

```
    """
    Plots training data, test data and compares predictions.
    """
    plt.figure(figsize=(10, 7))

    # Plot training data in blue
    plt.scatter(train_data, train_labels, c="b", s=4, label="Training data")

    # Plot test data in green
    plt.scatter(test_data, test_labels, c="g", s=4, label="Testing data")

    if predictions is not None:
        # Plot the predictions in red (predictions were made on the test data)
        plt.scatter(test_data, predictions, c="r", s=4, label="Predictions")

    # Show the legend
    plt.legend(prop={"size": 14});

plot_prediction()
```

**

```
**

# Create a Linear Regression model class

class LinearRegressionModel(nn.Module): # <- almost everything in PyTorch is a
nn.Module (think of this as neural network lego blocks)

    def __init__(self):

        super().__init__()

        self.weights = nn.Parameter(torch.randn(1, # <- start with random
weights (this will get adjusted as the model learns)

                                                dtype=torch.float), # <-
PyTorch loves float32 by default

                                       requires_grad=True) # <- can we update
this value with gradient descent?)



        self.bias = nn.Parameter(torch.randn(1, # <- start with random bias
(this will get adjusted as the model learns)

                                               dtype=torch.float), # <- PyTorch
loves float32 by default

                                    requires_grad=True) # <- can we update this
value with gradient descent?))



    # Forward defines the computation in the model

    def forward(self, x: torch.Tensor) -> torch.Tensor: # <- "x" is the input
data (e.g. training/testing features)

        return self.weights * x + self.bias # <- this is the linear regression
formula (y = m*x + b)
```

```
**

# Set manual seed since nn.Parameter are randomly initialzied

torch.manual_seed(42)



# Create an instance of the model (this is a subclass of nn.Module that
contains nn.Parameter(s))

model_0 = LinearRegressionModel()

model_0




# Make predictions with model

with torch.no_grad():

    y_preds = model_0(X_test)
**

# Check the predictions

print(f"Number of testing samples: {len(X_test)}")

print(f"Number of predictions made: {len(y_preds)}")



plot_predictions(predictions=y_preds)
**

# Create the loss function

loss_fn = nn.L1Loss() # MAE loss is same as L1Loss >> L2Loss(Mean Squared
Error): nn.MSELoss
```

```
# Create the optimizer

optimizer = torch.optim.SGD(params=model_0.parameters(), # parameters of
target model to optimize

                            lr=0.01) # learning rate (how much the optimizer
should change parameters at each step, higher=more (less stable), lower=less
(might take a long time))



torch.manual_seed(42)



# Set the number of epochs (how many times the model will pass over the
training data)

epochs = 100



# Create empty loss lists to track values

train_loss_values = []

test_loss_values = []

epoch_count = []



for epoch in range(epochs):

    ### Training
```

```
    # Put model in training mode (this is the default state of a model)
    model_0.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model_0(X_train)
    # print(y_pred)

    # 2. Calculate the loss (how different are our models predictions to the ground truth)
    loss = loss_fn(y_pred, y_train)

    # 3. Zero grad of the optimizer
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Progress the optimizer
    optimizer.step()
```

```
    ### Testing



    # Put the model in evaluation mode

    model_0.eval()



    with torch.no_grad():

      # 1. Forward pass on test data

      test_pred = model_0(X_test)



      # 2. Caculate loss on test data

      test_loss = loss_fn(test_pred, y_test.type(torch.float)) # predictions
come in torch.float datatype, so comparisons need to be done with tensors of
the same type



      # Print out what's happening

      if epoch % 10 == 0:

            epoch_count.append(epoch)

            train_loss_values.append(loss.detach().numpy())

            test_loss_values.append(test_loss.detach().numpy())

            print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss:
{test_loss} ")

**
```