**PyTorch Training (PyTorchJob)**

1. **PyTorch Operator 설치**

Pytorchjob의 실행을 위하여는 kubernetes custom resource인 pytorchjobs.kubeflow.org가 install 되어야 하고, 또한 training-operator pod가 실행되어야 한다.

제공하는 설치 매뉴얼(Install\_Manual\_MicroK8S\_Kubeflow\_Docker-V1.0.docx)을 참조하여 MicroK8s, Kubeflow 및 Docker를 설치한다.

1. **PyTorchJob 지원 확인**
2. PyTorch custom resource가 설치된 것을 확인한다.

**kubectl get crd**

|  |
| --- |
| NAME CREATED AT  ...  pytorchjobs.kubeflow.org 2022-04-26T05:20:12Z  ... |

1. Training operator POD가 실행 상태인 것을 확인한다.

**kubectl get pods -n kubeflow**

|  |
| --- |
| NAME READY STATUS RESTARTS AGE  …  training-operator-0 2/2 Running 2 48d  … |

1. **PV(persistent volume) 생성**
2. Pytorchjob 실행 결과로 생성된 pytorch model 파일(\*.pt)을 저장하기 위한 물리적 저장소(hostpath)를 생성한다.

## ****hostpath 생성****

**sudo mkdir ~/traindb/models**

1. 생성된 hostpath를 kubernetes에서 사용할 수 있도록 PV(persistent volume)을 배포한다. PV는 전역 자원이므로 namespace를 지정하지 않아도 된다.

**kubectl apply -f traindb-models-volume.yaml**

## ****traindb-models-volume.yaml****

|  |
| --- |
| apiVersion: v1  kind: PersistentVolume  metadata:  name: traindb-models-volume  labels:  type: traindb  name: traindb-models-volume  spec:  storageClassName: manual  capacity:  storage: 10Gi  accessModes:  - ReadWriteMany  hostPath:  path: "/home/traindbjst/traindb/models" |

1. PV 배포 상태를 확인한다.

**kubectl get pv traindb-models-volume**

|  |
| --- |
| CAPACITY ACCESSMODES RECLAIMPOLICY STATUS CLAIM STORAGECLASS REASON AGE  traindb-models-volume 10Gi **RWX** Retain **Available** manual 4s |

1. **Creating PVC**
2. 생성된 PV를 사용자 POD가 사용할 수 있도록 PVC(persistent volume claim)을 namespace kubeflow 배포한다.

**kubectl apply -f traindb-models-claim.yaml -n kubeflow**

## ****traindb-models-claim.yaml****

|  |
| --- |
| apiVersion: v1  kind: PersistentVolumeClaim  metadata:  name: traindb-models-claim  spec:  storageClassName: manual  accessModes:  - ReadWriteMany  resources:  requests:  storage: 3Gi  selector:  matchLabels:  name: traindb-models-volume  type: traindb |

1. PV 배포 상태를 확인한다.

**kubectl get pv traindb-models-volume -n kubeflow**

|  |
| --- |
| NAME CAPACITY ACCESSMODES RECLAIMPOLICY STATUS CLAIM STORAGECLASS REASON AGE  traindb-models-volume 10Gi RWX Retain **Bound** kubeflow/traindb-models-claim manual 2m |

1. PVC 배포 상태를 확인한다.

**kubectl get pvc pvc-traindb-models-claim -n kubeflow**

|  |
| --- |
| NAME STATUS VOLUME CAPACITY ACCESSMODES STORAGECLASS AGE  traindb-models-claim Bound traindb-models-volume 10Gi RWX manual 30s |

1. **Build Image**

Pytorchjob POD Example에서 실행될 Docker Image는 손글씨 인식 모델인 MNIST 모델을 사용한다.

제작된 Docker Image는 Docker Hub에 Push되어야 한다.

Docker image 제작을 위한 명령어는 다음과 같다.

1. Docker Image를 Build한다.

**sudo docker build -f Dockerfile\_training -t traindb-training-mnist .**

1. Docker Image를 Tagging한다.

**sudo docker tag traindb-training-mnist joleedocker/traindb-training-mnist:1.0**

1. Docker Image를 Docker Hub에 Push한다.

**sudo docker push joleedocker/traindb-training-mnist:1.0**

1. 생성된 Docker image를 확인한다.

**sudo docker images**

Docker image 제작을 위한 Dockerfile은 다음과 같다.

## ****Dockerfile\_training****

|  |
| --- |
| FROM pytorch/pytorch:1.0-cuda10.0-cudnn7-runtime  RUN apt update  RUN apt install software-properties-common -y  RUN add-apt-repository ppa:deadsnakes/ppa -y  RUN apt install python3.7 --version -y  RUN python -m pip install --upgrade pip  RUN pip install tensorboardX==1.6.0  RUN mkdir -p /opt/mnist  RUN mkdir -p /opt/mnist/models  WORKDIR /opt/mnist/src  ADD mnist\_training.py /opt/mnist/src/mnist\_training.py  RUN chgrp -R 0 /opt/mnist \  && chmod -R g+rwX /opt/mnist  ENTRYPOINT ["python3", "/opt/mnist/src/mnist\_training.py"] |

Docker image 제작을 위한 실행 프로그램은 다음과 같다.

## ****mnist\_****training****.py****

|  |
| --- |
| from \_\_future\_\_ import print\_function  import argparse  import os  from tensorboardX import SummaryWriter  from torchvision import datasets, transforms  import torch  import torch.distributed as dist  import torch.nn as nn  import torch.nn.functional as F  import torch.optim as optim  WORLD\_SIZE = int(os.environ.get('WORLD\_SIZE', 1))  class Net(nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  self.conv1 = nn.Conv2d(1, 20, 5, 1)  self.conv2 = nn.Conv2d(20, 50, 5, 1)  self.fc1 = nn.Linear(4\*4\*50, 500)  self.fc2 = nn.Linear(500, 10)  def forward(self, x):  x = F.relu(self.conv1(x))  x = F.max\_pool2d(x, 2, 2)  x = F.relu(self.conv2(x))  x = F.max\_pool2d(x, 2, 2)  x = x.view(-1, 4\*4\*50)  x = F.relu(self.fc1(x))  x = self.fc2(x)  return F.log\_softmax(x, dim=1)    def train(args, model, device, train\_loader, optimizer, epoch, writer):  model.train()  for batch\_idx, (data, target) in enumerate(train\_loader):  data, target = data.to(device), target.to(device)  optimizer.zero\_grad()  output = model(data)  loss = F.nll\_loss(output, target)  loss.backward()  optimizer.step()  if batch\_idx % args.log\_interval == 0:  print('Train Epoch: {} [{}/{} ({:.0f}%)]\tloss={:.4f}'.format(  epoch, batch\_idx \* len(data), len(train\_loader.dataset),  100. \* batch\_idx / len(train\_loader), loss.item()))  niter = epoch \* len(train\_loader) + batch\_idx  writer.add\_scalar('loss', loss.item(), niter)  def test(args, model, device, test\_loader, writer, epoch):  model.eval()  test\_loss = 0  correct = 0  with torch.no\_grad():  for data, target in test\_loader:  data, target = data.to(device), target.to(device)  output = model(data)  test\_loss += F.nll\_loss(output, target, reduction='sum').item() # sum up batch loss  pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability  correct += pred.eq(target.view\_as(pred)).sum().item()  test\_loss /= len(test\_loader.dataset)  print('\naccuracy={:.4f}\n'.format(float(correct) / len(test\_loader.dataset)))  writer.add\_scalar('accuracy', float(correct) / len(test\_loader.dataset), epoch)  def should\_distribute():  return dist.is\_available() and WORLD\_SIZE > 1  def is\_distributed():  return dist.is\_available() and dist.is\_initialized()  def main():  # Training settings  parser = argparse.ArgumentParser(description='PyTorch MNIST Example')  parser.add\_argument('--batch-size', type=int, default=64, metavar='N',  help='input batch size for training (default: 64)')  parser.add\_argument('--test-batch-size', type=int, default=1000, metavar='N',  help='input batch size for testing (default: 1000)')  parser.add\_argument('--epochs', type=int, default=1, metavar='N',  help='number of epochs to train (default: 10)')  parser.add\_argument('--lr', type=float, default=0.01, metavar='LR',  help='learning rate (default: 0.01)')  parser.add\_argument('--momentum', type=float, default=0.5, metavar='M',  help='SGD momentum (default: 0.5)')  parser.add\_argument('--no-cuda', action='store\_true', default=False,  help='disables CUDA training')  parser.add\_argument('--seed', type=int, default=1, metavar='S',  help='random seed (default: 1)')  parser.add\_argument('--log-interval', type=int, default=10, metavar='N',  help='how many batches to wait before logging training status')  parser.add\_argument('--save-model', action='store\_true', default=False,  help='For Saving the current Model')  parser.add\_argument('--dir', default='logs', metavar='L',  help='directory where summary logs are stored')  if dist.is\_available():  parser.add\_argument('--backend', type=str, help='Distributed backend',  choices=[dist.Backend.GLOO, dist.Backend.NCCL, dist.Backend.MPI],  default=dist.Backend.GLOO)  args = parser.parse\_args()  use\_cuda = not args.no\_cuda and torch.cuda.is\_available()  if use\_cuda:  print('Using CUDA')  model\_export\_dir = "/opt/mnist/models/”  writer = SummaryWriter(args.dir)  torch.manual\_seed(args.seed)  device = torch.device("cuda" if use\_cuda else "cpu")  if should\_distribute():  print('Using distributed PyTorch with {} backend'.format(args.backend))  dist.init\_process\_group(backend=args.backend)  kwargs = {'num\_workers': 1, 'pin\_memory': True} if use\_cuda else {}  train\_loader = torch.utils.data.DataLoader(  datasets.FashionMNIST('../data', train=True, download=True,  transform=transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,))  ])),  batch\_size=args.batch\_size, shuffle=True, \*\*kwargs)  test\_loader = torch.utils.data.DataLoader(  datasets.FashionMNIST('../data', train=False, transform=transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,))  ])),  batch\_size=args.test\_batch\_size, shuffle=False, \*\*kwargs)  model = Net().to(device)  if is\_distributed():  Distributor = nn.parallel.DistributedDataParallel if use\_cuda \  else nn.parallel.DistributedDataParallelCPU  model = Distributor(model)  optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)  for epoch in range(1, args.epochs + 1):  train(args, model, device, train\_loader, optimizer, epoch, writer)  test(args, model, device, test\_loader, writer, epoch)  torch.save(model.state\_dict(),model\_export\_dir + "mnist\_cnn.pt")  if (args.save\_model):  torch.save(model.state\_dict(),"mnist\_cnn.pt")    if \_\_name\_\_ == '\_\_main\_\_':  main() |

1. **Creating a PyTorch training job**
2. Pytorchjob training POD를 Kubernetes에 namespace kubeflow에배포한다.

**kubectl create -f traindb-training-mnist.yaml -n kubeflow**

YAML파일은 다음과 같다.

## ****traindb-training-mnist.yaml****

|  |
| --- |
| apiVersion: "kubeflow.org/v1"  kind: PyTorchJob  metadata:  name: traindb-training-mnist  spec:  pytorchReplicaSpecs:  Master:  replicas: 1  restartPolicy: OnFailure  template:  spec:  volumes:  - name: task-pv-storage-master  persistentVolumeClaim:  claimName: traindb-models-claim  containers:  - name: pytorch  image: joleedocker/traindb-training-mnist:1.0  imagePullPolicy: Always  volumeMounts:  - mountPath: "/opt/mnist/models"  name: task-pv-storage-master  command:  - "python3"  - "/opt/mnist/src/mnist\_training.py"  - "--epochs=1"  Worker:  replicas: 1  restartPolicy: OnFailure  template:  spec:  volumes:  - name: task-pv-storage-worker  persistentVolumeClaim:  claimName: traindb-models-claim  containers:  - name: pytorch  image: joleedocker/traindb-training-mnist:1.0  imagePullPolicy: Always  volumeMounts:  - mountPath: "/opt/mnist/models"  name: task-pv-storage-worker  command:  - "python3"  - "/opt/mnist/src/mnist\_training.py"  - "--epochs=1" |

1. 배포된 Pytorchjob training POD를 확인한다.

**kubectl get pods -l job-name= traindb-training-mnist.yaml -n kubeflow**

1. 배포된 Pytorchjob training POD의 실행 로그를 확인한다.

**PODNAME=$(kubectl get pods -l job-name= traindb-training-mnist.yaml, replica-type=master,replica-index=0 -o name -n kubeflow)**

**kubectl logs -f ${PODNAME} -n kubeflow**

|  |
| --- |
| Using distributed PyTorch with gloo backend  Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz  Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz  Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz  Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz  Processing...  Done!  Train Epoch: 1 [0/60000 (0%)] loss=2.2908  Train Epoch: 1 [640/60000 (1%)] loss=2.2315  Train Epoch: 1 [1280/60000 (2%)] loss=2.1507  Train Epoch: 1 [1920/60000 (3%)] loss=1.9067  …  …  …  Train Epoch: 1 [58240/60000 (97%)] loss=0.5401  Train Epoch: 1 [58880/60000 (98%)] loss=0.7318  Train Epoch: 1 [59520/60000 (99%)] loss=0.3656  accuracy=0.7928 |

1. 배포된 Pytorchjob training POD를 확인한다. 실행이 완료되었다면 상태가 Completed 이다.

**kubectl get pods -l job-name=traindb-training-mnist -n kubeflow**

|  |
| --- |
| NAME READY STATUS RESTARTS AGE  pytorchjob-mnist-pvc-worker-0 0/1 Completed 0 19m  pytorchjob-mnist-pvc-master-0 0/1 Completed 0 19m |

1. Pytorchjob training POD가 실행 완료되면 hostpath에 pytorch model이 생성된 것을 확인한다.

**ls -l ../traindb/models**

|  |
| --- |
| total 1700  -rw-r--r-- 1 root root 1725733 Jun 13 19:23 mnist\_cnn.pt |

1. Pytorchjob training POD를 삭제할 경우 다음 명령을 실행한다.

**kubectl delete pods -l job-name=traindb-training-mnist -n kubeflow**

|  |
| --- |
| pod "pytorchjob-mnist-pvc-worker-0" deleted  pod "pytorchjob-mnist-pvc-master-0" deleted |

1. Pytorchjob training POD가 삭제된 것을 확인한다.

**kubectl get pods -l job-name=traindb-training-mnist -n kubeflow**

|  |
| --- |
| No resources found in kubeflow namespace. |

1. **Monitoring a PyTorchJob**

Pytorchjob training POD의 실행 및 완료까지의 상태를 확인하고 싶은 경우 아래의 명령어를 실행한다.

**kubectl get -o yaml pytorchjobs traindb-training-mnist -n kubeflow**

## ****진행 메시지****

|  |
| --- |
| apiVersion: kubeflow.org/v1  kind: PyTorchJob  metadata:  creationTimestamp: "2022-06-14T02:44:27Z"  generation: 1  name: traindb-training-mnist  namespace: kubeflow  resourceVersion: "31210806"  selfLink: /apis/kubeflow.org/v1/namespaces/kubeflow/pytorchjobs/traindb-training-mnist  uid: ac61a72d-e5e1-4c58-87f3-2dad8a729be1  spec:  pytorchReplicaSpecs:  Master:  replicas: 1  restartPolicy: OnFailure  template:  spec:  containers:  - command:  - python3  - /opt/mnist/src/mnist\_training.py  - --epochs=1  image: joleedocker/traindb-training-mnist:1.0  imagePullPolicy: Always  name: pytorch  volumeMounts:  - mountPath: /opt/mnist/models  name: task-pv-storage-master  volumes:  - name: task-pv-storage-master  persistentVolumeClaim:  claimName: traindb-models-claim  Worker:  replicas: 1  restartPolicy: OnFailure  template:  spec:  containers:  - command:  - python3  - /opt/mnist/src/mnist\_training.py  - --epochs=1  image: joleedocker/traindb-training-mnist:1.0  imagePullPolicy: Always  name: pytorch  volumeMounts:  - mountPath: /opt/mnist/models  name: task-pv-storage-worker  volumes:  - name: task-pv-storage-worker  persistentVolumeClaim:  claimName: traindb-models-claim  status:  conditions:  - lastTransitionTime: "2022-06-14T02:44:27Z"  lastUpdateTime: "2022-06-14T02:44:27Z"  message: PyTorchJob traindb-training-mnist is created.  reason: PyTorchJobCreated  status: "True"  type: Created  - lastTransitionTime: "2022-06-14T02:44:28Z"  lastUpdateTime: "2022-06-14T02:44:28Z"  message: PyTorchJob traindb-training-mnist is running.  reason: JobRunning  status: "True"  type: Running  replicaStatuses:  Master:  active: 1  Worker:  active: 1 |

## ****완료 메시지****

|  |
| --- |
| apiVersion: kubeflow.org/v1  kind: PyTorchJob  metadata:  creationTimestamp: "2022-06-14T02:44:27Z"  generation: 1  name: traindb-training-mnist  namespace: kubeflow  resourceVersion: "31210806"  selfLink: /apis/kubeflow.org/v1/namespaces/kubeflow/pytorchjobs/traindb-training-mnist  uid: ac61a72d-e5e1-4c58-87f3-2dad8a729be1  spec:  pytorchReplicaSpecs:  Master:  replicas: 1  restartPolicy: OnFailure  template:  spec:  containers:  - command:  - python3  - /opt/mnist/src/mnist\_training.py  - --epochs=1  image: joleedocker/traindb-training-mnist:1.0  imagePullPolicy: Always  name: pytorch  volumeMounts:  - mountPath: /opt/mnist/models  name: task-pv-storage-master  volumes:  - name: task-pv-storage-master  persistentVolumeClaim:  claimName: traindb-models-claim  Worker:  replicas: 1  restartPolicy: OnFailure  template:  spec:  containers:  - command:  - python3  - /opt/mnist/src/mnist\_training.py  - --epochs=1  image: joleedocker/traindb-training-mnist:1.0  imagePullPolicy: Always  name: pytorch  volumeMounts:  - mountPath: /opt/mnist/models  name: task-pv-storage-worker  volumes:  - name: task-pv-storage-worker  persistentVolumeClaim:  claimName: traindb-models-claim  status:  completionTime: "2022-06-14T02:57:54Z"  conditions:  - lastTransitionTime: "2022-06-14T02:44:27Z"  lastUpdateTime: "2022-06-14T02:44:27Z"  message: PyTorchJob traindb-training-mnist is created.  reason: PyTorchJobCreated  status: "True"  type: Created  - lastTransitionTime: "2022-06-14T02:44:28Z"  lastUpdateTime: "2022-06-14T02:44:28Z"  message: PyTorchJob traindb-training-mnist is running.  reason: JobRunning  status: "False"  type: Running  - lastTransitionTime: "2022-06-14T02:57:54Z"  lastUpdateTime: "2022-06-14T02:57:54Z"  message: PyTorchJob traindb-training-mnist is successfully completed.  reason: JobSucceeded  status: "True"  type: Succeeded  replicaStatuses:  Master:  succeeded: 1  Worker:  succeeded: 1 |