🧪 Lab: Deploy Multi-GPU Training Job on Kubernetes

Goal:
Run a distributed deep learning job on Kubernetes using multiple GPUs. You'll do a **single-node / multi-GPU** run first (simpler), then an **advanced multi-node** variant.

What you'll need:

- A Kubernetes cluster (v1.24+) with at least **1 GPU node** (for single-node) or **2 GPU nodes** (for multi-node), each with ≥2 GPUs
- NVIDIA GPU drivers + **nvidia-container-toolkit** on GPU nodes
- **NVIDIA K8s device plugin** deployed cluster-wide
- `kubectl`, `helm` installed locally
- Outbound internet (or a PVC with your dataset)

---

0) Prep & Verification (once per cluster)

1. **Create a namespace** for this lab:

```
kubectl create namespace ai-lab
```

2. **Install NVIDIA Device Plugin** (if not already):

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo update
helm upgrade --install nvidia-device-plugin nvdp/nvidia-device-plugin \
  --namespace kube-system
```

3. **Confirm GPUs are allocatable:**

```
kubectl get nodes \
  -o=custom-columns=NAME:.metadata.name,GPUS:.status.allocatable.nvidia\.com/g
```

You should see non-zero GPU counts on your GPU nodes.

---

1) Provide the training script via ConfigMap

This PyTorch **DDP** script trains ResNet18 on CIFAR-10 and works for both single- and multi-node. It uses `torchrun` and saves checkpoints only on rank 0.

```
cat > train.py <<'PY'
import os, argparse, torch, torch.nn as nn, torch.optim as optim
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, DistributedSampler

def parse_args():
    p = argparse.ArgumentParser()
    p.add_argument("--epochs", type=int, default=2)
    p.add_argument("--batch-size", type=int, default=256)
    p.add_argument("--data-dir", type=str, default="/workspace/data")
    p.add_argument("--out-dir", type=str, default="/outputs")
    return p.parse_args()

def setup_distributed():
    # torchrun sets LOCAL_RANK, RANK, WORLD_SIZE
    local_rank = int(os.environ.get("LOCAL_RANK", 0))
    torch.cuda.set_device(local_rank)
    dist.init_process_group(backend="nccl")
    return local_rank

def main():
    args = parse_args()
    os.makedirs(args.out_dir, exist_ok=True)
```

```
    local_rank = setup_distributed()
    rank = dist.get_rank()
    world_size = dist.get_world_size()

    transform_train = transforms.Compose([
        transforms.Resize(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor()
    ])
    dataset = datasets.CIFAR10(root=args.data_dir, train=True, download=True,
    sampler = DistributedSampler(dataset, num_replicas=world_size, rank=rank,
    loader = DataLoader(dataset, batch_size=args.batch_size, sampler=sampler,

    model = models.resnet18(num_classes=10).cuda()
    model = DDP(model, device_ids=[int(os.environ.get("LOCAL_RANK", 0))])
    criterion = nn.CrossEntropyLoss().cuda()
    optimizer = optim.Adam(model.parameters(), lr=1e-3)

    for epoch in range(args.epochs):
        sampler.set_epoch(epoch)
        model.train()
        running = 0.0
        for i, (x, y) in enumerate(loader):
            x, y = x.cuda(non_blocking=True), y.cuda(non_blocking=True)
            optimizer.zero_grad(set_to_none=True)
            out = model(x)
            loss = criterion(out, y)
            loss.backward()
            optimizer.step()
            running += loss.item()
            if i % 50 == 0 and rank == 0:
                print(f"[epoch {epoch}] step {i} avg_loss={running/(i+1):.4f}"

    if rank == 0:
        torch.save(model.module.state_dict(), os.path.join(args.out_dir, "resr
        print("Saved checkpoint.")

    dist.destroy_process_group()

if __name__ == "__main__":
    main()
PY
```

```
kubectl -n ai-lab create configmap ddp-train --from-file=train.py
```

---

2) Single-node, Multi-GPU Job (easiest path)

This runs 1 pod requesting **4 GPUs** on one node (adjust to your node's GPU count).

Apply the Job:

```
cat > pytorch-ddp-1node.yaml <<'YAML'
apiVersion: batch/v1
kind: Job
metadata:
  name: ddp-1node
  namespace: ai-lab
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
      - name: trainer
        image: nvcr.io/nvidia/pytorch:24.03-py3
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            nvidia.com/gpu: 4   # <-- set to number of GPUs on the node
        env:
        - name: NCCL_ASYNC_ERROR_HANDLING
          value: "1"
        - name: NCCL_IB_DISABLE    # set "0" if you have InfiniBand
          value: "1"
        - name: OMP_NUM_THREADS
          value: "4"
        volumeMounts:
        - name: script
          mountPath: /workspace/train.py
```

```
          subPath: train.py
        - name: outputs
          mountPath: /outputs
        command: ["bash","-lc"]
        args:
        - |
          cd /workspace && \
          torchrun --standalone --nproc_per_node=4 \
            /workspace/train.py --epochs 2 --batch-size 256 --out-dir /outputs
      volumes:
      - name: script
        configMap:
          name: ddp-train
      - name: outputs
        emptyDir: {}
YAML

kubectl apply -f pytorch-ddp-1node.yaml
```

Watch logs and success:

```
kubectl -n ai-lab logs -f job/ddp-1node
```

You should see loss printing and a **"Saved checkpoint."** message at the end (rank 0 only).

Inspect GPU use on the node (optional):

```
# find the pod name
POD=$(kubectl -n ai-lab get pods -l job-name=ddp-1node -o jsonpath='{.items[0]
kubectl -n ai-lab exec -it $POD -- nvidia-smi
```

---

3) Advanced: Multi-node, Multi-GPU (StatefulSet + headless Service)

This runs **2 pods across 2 GPU nodes**, each pod requesting **2 GPUs** (= 4 GPUs total). It uses `torchrun` with a rendezvous on `ddp-0`.

> **Tip:** Ensure you have at least 2 GPU nodes schedulable. Label them if you want to steer placement (e.g., `kubectl label nodes <node> gpu=true` then add nodeSelectors).

(a) Headless Service (stable DNS for pods):

```
cat > ddp-hs.yaml <<'YAML'
apiVersion: v1
kind: Service
metadata:
  name: ddp-hs
  namespace: ai-lab
spec:
  clusterIP: None
  selector:
    app: ddp
  ports:
  - name: rdzv
    port: 29400
    targetPort: 29400
YAML

kubectl apply -f ddp-hs.yaml
```

(b) StatefulSet with 2 replicas:

```
cat > ddp-ss.yaml <<'YAML'
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: ddp
  namespace: ai-lab
spec:
  serviceName: ddp-hs
  replicas: 2                # <-- number of nodes
  selector:
    matchLabels:
      app: ddp
  template:
```

```yaml
metadata:
  labels:
    app: ddp
spec:
  restartPolicy: Always
  containers:
  - name: trainer
    image: nvcr.io/nvidia/pytorch:24.03-py3
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        nvidia.com/gpu: 2   # <-- GPUs per pod (node)
    ports:
    - containerPort: 29400  # rendezvous port
    env:
    - name: NCCL_ASYNC_ERROR_HANDLING
      value: "1"
    - name: NCCL_IB_DISABLE    # set "0" if you have InfiniBand
      value: "1"
    - name: OMP_NUM_THREADS
      value: "4"
    volumeMounts:
    - name: script
      mountPath: /workspace/train.py
      subPath: train.py
    - name: outputs
      mountPath: /outputs
    command: ["bash","-lc"]
    args:
    - |
      set -e
      # Derive this pod's ordinal (0-based) from HOSTNAME (ddp-0, ddp-1, .
      ORD=${HOSTNAME##*-}
      MASTER_ADDR="ddp-0.ddp-hs"
      MASTER_PORT=29400
      NNODES=2
      NPROC_PER_NODE=2
      NODE_RANK=${ORD}

      # Start rendezvous listener only on pod-0 (optional but harmless to
      # torchrun with --master_addr/port works even if port not pre-openec

      cd /workspace && \
```

```
          torchrun \
            --nnodes=${NNODES} \
            --nproc_per_node=${NPROC_PER_NODE} \
            --node_rank=${NODE_RANK} \
            --master_addr=${MASTER_ADDR} \
            --master_port=${MASTER_PORT} \
            /workspace/train.py --epochs 2 --batch-size 256 --out-dir /outputs
      volumes:
      - name: script
        configMap:
          name: ddp-train
      - name: outputs
        emptyDir: {}
YAML

kubectl apply -f ddp-ss.yaml
```

© Verify both pods are Running:

```
kubectl -n ai-lab get pods -l app=ddp -o wide
```

(d) Follow one pod's logs:

```
kubectl -n ai-lab logs -f ddp-0
# In another terminal:
kubectl -n ai-lab logs -f ddp-1
```

You should see all ranks (e.g., ranks 0–3) joining the process group and training.

> **If pods stay Pending:** Ensure enough GPUs per node and that the **device plugin** is running on those nodes (`kubectl -n kube-system get ds nvidia-device-plugin -o wide`).

---

4) Optional: Use a PersistentVolumeClaim for outputs

Replace the `emptyDir` with a PVC to persist checkpoints:

```
kubectl -n ai-lab apply -f - <<'YAML'
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ddp-outputs
spec:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 10Gi
YAML
```

Then in your Job/StatefulSet, swap:

```
volumeMounts:
- name: outputs
  mountPath: /outputs
volumes:
- name: outputs
  persistentVolumeClaim:
    claimName: ddp-outputs
```

---

5) Performance & Health Checks

- **GPU view per pod:**

  ```
  kubectl -n ai-lab exec -it ddp-0 -- nvidia-smi
  ```

- **Prometheus/Grafana** with **DCGM Exporter** (if you use them) show per-GPU utilization, memory, temperature, ECC errors.
- NCCL tips:

- If using InfiniBand: set `NCCL_IB_DISABLE=0`
- Ensure the correct interface with `NCCL_SOCKET_IFNAME=eth0` (or your fabric NIC)
- For debug: `NCCL_DEBUG=INFO`

---

6) Scale Up / Down

- **Change GPUs per node** (edit `resources.limits.nvidia.com/gpu`)
- **Change world size** by increasing:
  - Single-node: `--nproc_per_node`
  - Multi-node: `replicas` and `--nnodes` / `--nproc_per_node`
- **Blue-green jobs:** duplicate StatefulSet/Job with a new name and adjusted resources.

---

7) Cleanup

```
kubectl -n ai-lab delete job/ddp-1node || true
kubectl -n ai-lab delete statefulset/ddp service/ddp-hs || true
kubectl -n ai-lab delete configmap/ddp-train pvc/ddp-outputs || true
kubectl delete namespace ai-lab
```

✅ You've deployed a distributed, multi-GPU training job on Kubernetes

- **Single-node** DDP using 4 GPUs in one pod

- **Multi-node** DDP using 2 pods × 2 GPUs with a headless Service rendezvous
- Verified logs, GPU allocation, and produced a checkpoint