

# MNIST on Google Kubernetes Engine (GKE): Training Job + Inference Service

## Abstract

This report documents an end-to-end workflow to train a simple MNIST model inside Google Kubernetes Engine (GKE) using a one-time **Job**, persist the trained model in a **PersistentVolumeClaim (PVC)**, and deploy a **FastAPI** inference server as a **Deployment** fronted by a **LoadBalancer Service**. The approach is intentionally minimal, reproducible, and suitable as a reference for ML lifecycle on Kubernetes.

## 1 Objective

Goal: containerize and run two workloads on GKE a one-off **training job** that saves `model.pth` to a PVC, and a **stateless inference service** that mounts the PVC to load the saved model and exposes `/predict` for image classification.

## 2 Architecture Overview

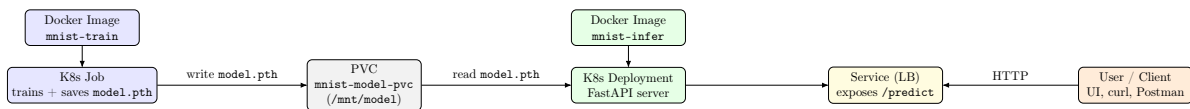


Figure 1: Training Job persists the model to the PVC; the Deployment reads it and serves predictions via a LoadBalancer Service.

## 3 Training Workflow

### Code

**File:** `training/training.py`. A small CNN trained on MNIST (CPU) saves `model.pth`.

```
# Saves /mnt/model/model.pth (DIR overridable via MODEL_DIR)
model_dir = Path(os.environ.get("MODEL_DIR", "/mnt/model"))
model_dir.mkdir(parents=True, exist_ok=True)
torch.save(model.state_dict(), str(model_dir / "model.pth"))
```

### Image

**File:** `training/Dockerfile` (CPU-only PyTorch base).

```
FROM python:3.10-slim
RUN pip install torch torchvision torchaudio \
    --index-url https://download.pytorch.org/whl/cpu
COPY training/training.py /app/training.py
CMD ["python", "-u", "training.py"]
```

## Persistent Volume

**File:** k8s/pvc.yaml (5Gi, standard-rwo).

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mnist-model-pvc
spec:
  accessModes: [ReadWriteOnce]
  resources: { requests: { storage: 5Gi } }
  storageClassName: standard-rwo
```

## Kubernetes Job

**File:** k8s/job-training.yaml.

```
apiVersion: batch/v1
kind: Job
metadata: { name: mnist-training-job }
spec:
  template:
    spec:
      restartPolicy: Never
      containers:
      - name: trainer
        image: asia-east1-docker.pkg.dev/.../mnist-train:latest
        env: [{ name: MODEL_DIR, value: /mnt/model }]
        volumeMounts: [{ name: model-vol, mountPath: /mnt/model }]
      volumes:
      - name: model-vol
        persistentVolumeClaim: { claimName: mnist-model-pvc }
```

## 4 Inference Workflow

### Code

**File:** inference/inference.py. Loads weights on startup; exposes /predict (POST multipart 'file') and /healthz. Minimal HTML upload UI at /.

```
@app.on_event("startup")
def _load_on_start():
    global model
    model = load_model(Path(os.environ.get("MODEL_DIR", "/mnt/model"))/"model.pth")
```

```
@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    img = Image.open(io.BytesIO(await file.read())).convert("L")
    x = transform(img).unsqueeze(0)
    with torch.no_grad():
        logits = model(x)
    return {"prediction": int(torch.argmax(logits,1)),
            "probs": torch.softmax(logits,1).squeeze().tolist()}
```

## Image

**File:** inference/Dockerfile (CPU PyTorch + FastAPI + Uvicorn).

```
RUN pip install fastapi uvicorn[standard] pillow python-multipart
CMD ["uvicorn", "inference:app", "--host", "0.0.0.0", "--port", "8080"]
```

## Kubernetes Deployment + Service

**Files:** k8s/deploy-inference.yaml, k8s/svc-inference.yaml.

```
apiVersion: apps/v1
kind: Deployment
metadata: { name: mnist-inference-deploy }
spec:
  replicas: 1
  template:
    spec:
      containers:
        - name: server
          image: asia-east1-docker.pkg.dev/.../mnist-infer:latest
          ports: [{ containerPort: 8080 }]
          readinessProbe: { httpGet: { path: /healthz, port: 8080 } }
          livenessProbe: { httpGet: { path: /healthz, port: 8080 } }
          volumeMounts: [{ name: model-vol, mountPath: /mnt/model }]
      volumes:
        - name: model-vol
          persistentVolumeClaim: { claimName: mnist-model-pvc }
---
apiVersion: v1
kind: Service
metadata: { name: mnist-inference-svc }
spec:
  type: LoadBalancer
  selector: { app: mnist-inference }
  ports: [{ port: 80, targetPort: 8080 }]
```

## 5 Deployment Steps (CLI)

```
# Set context
export PROJECT_ID=csci-ga-3033085
export REGION=asia-east1
export REPO=mnist-repo
```

```

export TRAIN_IMG=$REGION-docker.pkg.dev/$PROJECT_ID/$REPO/mnist-train:latest
export INFER_IMG=$REGION-docker.pkg.dev/$PROJECT_ID/$REPO/mnist-infer:latest

# Build (Apple Silicon)
export DOCKER_DEFAULT_PLATFORM=linux/amd64
docker build -t $TRAIN_IMG -f training/Dockerfile .
docker build -t $INFER_IMG -f inference/Dockerfile .
docker push $TRAIN_IMG && docker push $INFER_IMG

# Kubernetes
kubectl apply -f k8s/pvc.yaml
kubectl apply -f k8s/job-training.yaml
kubectl logs -l job-name=mnist-training-job -f

# After Job Succeeded
kubectl apply -f k8s/deploy-inference.yaml
kubectl apply -f k8s/svc-inference.yaml
kubectl get svc mnist-inference-svc -w

```

## 6 Testing and Results

**Health:** ‘GET /healthz’ returns ‘{"status":"ok"}’ once the model is mounted.

**Predict:** ‘POST /predict’ with multipart form field ‘file’ (PNG/JPG). Minimal UI available at ‘/’.

```

# Generate simple samples
python3 samples/gen_samples.py

curl -s -X POST -F "file=@samples/zero.png" http://$EXTERNAL_IP/predict
# => {"prediction": 0, "probs": [...]}

curl -s -X POST -F "file=@samples/one.png" http://$EXTERNAL_IP/predict
# => {"prediction": 1, "probs": [...]}

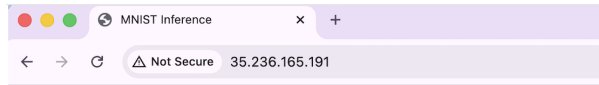
```

## 7 Self-Healing and Reliability

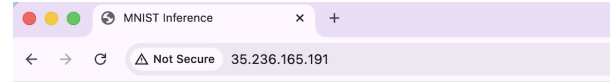
- **Deployment:** GKE *Deployment* ensures the inference pod is always running; if the pod crashes, it is recreated automatically.
- **Probes:** *Readiness* and *liveness* probes against ‘/healthz’ gate traffic and detect crashes/hangs.
- **Scaling:** Autopilot scales nodes up when workloads need capacity; the Job initially sits ‘Pending’ until nodes are available.
- **Persistence:** Model artifacts survive pod restarts through the PVC.

## 8 Issues Faced and Fixes

- **kubectl auth error:** Missing gke-gcloud-auth-plugin prevented API calls. *Fix:* install plugin and set USE\_GKE\_GCLOUD\_AUTH\_PLUGIN=True.



(a) Upload UI



(b) Prediction result

Figure 2: MNIST upload UI and the corresponding prediction response, shown side-by-side.

- **Cloud Build permissions:** API disabled initially. *Fix:* enable ‘cloudbuild.googleapis.com’ or build locally with Docker.
- **Apple Silicon builds:** Images built for arm64 failed on amd64 nodes. *Fix:* set `DOCKER_DEFAULT_PLATFORM=linux/amd64`.
- **Form upload 422/405:** /predict is POST-only and requires python-multipart. *Fix:* added dependency and clarified usage.
- **Scale-up wait:** Job ‘Pending’ due to no nodes. *Fix:* wait for Autopilot to provision nodes (minutes), then training proceeds.

## 9 Future Improvements

- CI/CD: automate build/push/deploy via Cloud Build triggers or GitHub Actions.
- Metrics/Tracing: add request latency, model latency; integrate Cloud Logging/Monitoring.
- Canary/Blue-Green: roll out new model versions with zero downtime.
- Data: move datasets to GCS; mount via GCSFuse or stage in container.
- Security: use Artifact Registry attestations; restrict Service exposure; add authn/z to API.

## References

PyTorch, torchvision MNIST dataset; GKE docs on Autopilot, Deployments, Jobs, PVCs; FastAPI documentation.