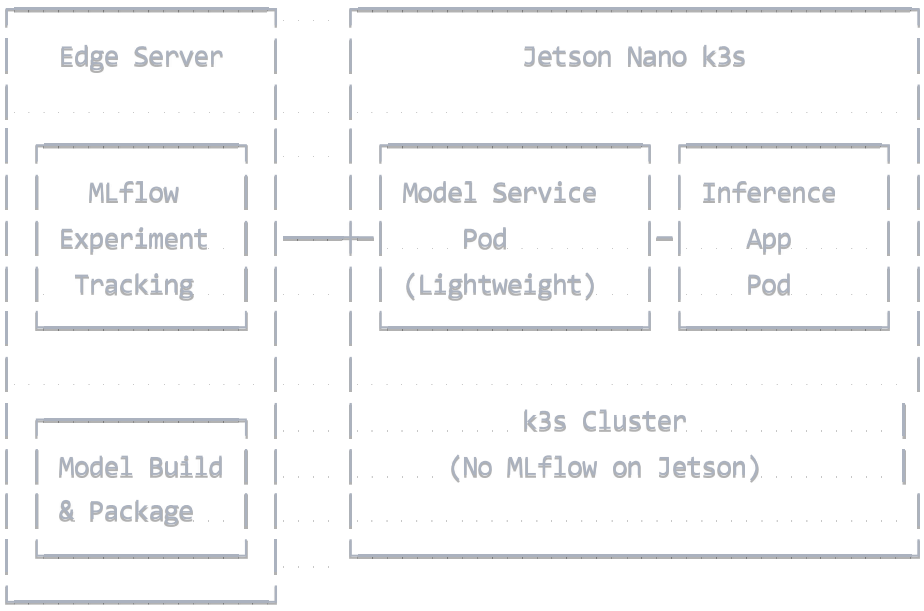
Face Recognition System - Complete Setup Guide

System Architecture Overview



Part 1: Edge Server Setup

1.1 Prerequisites

Hardware Requirements:

- CPU: 4+ cores
- RAM: 8GB+
- Storage: 50GB+ free space
- GPU: Optional (CUDA-compatible for faster training)

Software Requirements:

- Ubuntu 20.04+ or similar Linux distribution
- Python 3.8+
- Docker
- Git

1.2 Install System Dependencies

bash

```bash
# Update system
sudo apt update && sudo apt upgrade -y

# Install Python and development tools
sudo apt install -y python3 python3-pip python3-dev python3-venv
sudo apt install -y build-essential cmake pkg-config
sudo apt install -y libopencv-dev python3-opencv
sudo apt install -y git curl wget

# Install Docker
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER

# Logout and login again for Docker permissions to take effect
```

1.3 Create Project Structure

bash

```bash
# Create main project directory
mkdir -p ~/face_recognition_system
cd ~/face_recognition_system

# Create edge server directory
mkdir -p edge_server
cd edge_server

# Create subdirectories
mkdir -p images models few_shot_examples k8s_deployment
mkdir -p images/amrit images/john images/alice  # Example person directories
```

1.4 Install Python Dependencies

```bash
# Create virtual environment
python3 -m venv face_recognition_env
source face_recognition_env/bin/activate

# Install core dependencies
pip install --upgrade pip

# MLflow and experiment tracking
pip install mlflow==2.8.1
pip install matplotlib seaborn

# Computer Vision and ML
pip install opencv-python==4.8.0.76
pip install torch torchvision --index-url https://download.pytorch.org/whl/cpu
pip install ultralytics  # for YOLO
pip install scikit-learn
pip install numpy pandas Pillow

# Additional utilities
pip install pyyaml
pip install ipython jupyter  # Optional for development
```

1.5 Setup Project Files

The system automatically creates all necessary files when you initialize the trainer. You just need to copy the main scripts:

Required files to copy manually:

- `edge_training.py`
- `mlflow_config.py`
- `jetson_inference.py`

Files automatically created by the system:

- `Dockerfile.inference` ✅ (auto-generated)
- `requirements_jetson.txt` ✅ (auto-generated)
- `auto_deploy.py` ✅ (auto-generated)
- All Kubernetes manifests ✅ (auto-generated)

```bash
# Copy the main scripts to your edge_server/ directory
# (Download from the artifacts or copy from your implementation)

# Initialize the system - this creates all deployment files automatically
python -c "from edge_training import EdgeFaceRecognitionTrainer; EdgeFaceRecognitionTrainer()"

# Initialize MLflow
python mlflow_config.py

# Verify setup
ls -la
# Should see: mlflow_edge.db, mlruns/, Dockerfile.inference, requirements_jetson.txt, etc.
```

What gets automatically created:

- ✅ `Dockerfile.inference` - Docker container for inference app
- ✅ `requirements_jetson.txt` - Python dependencies for Jetson
- ✅ `auto_deploy.py` - Automatic k3s deployment script
- ✅ `build_k8s.sh` - Docker build and packaging script
- ✅ `deploy_to_jetson.sh` - Automated deployment script
- ✅ `mlflow_ui.sh` - MLflow UI launcher

1.6 Prepare Training Data

```bash
# Example directory structure for training images
mkdir -p images/person1 images/person2 images/person3

# Add at least 4-5 images per person:
# images/person1/person1_1.jpg
# images/person1/person1_2.jpg
# images/person1/person1_3.jpg
# images/person1/person1_4.jpg
# images/person1/person1_5.jpg

# Image requirements:
# - Format: JPG/PNG
# - Size: Any (will be resized)
# - Quality: Clear face visible
# - Variety: Different angles, lighting
```

Part 2: Jetson Nano Setup

2.1 Prerequisites

Hardware:

- Jetson Nano Developer Kit
- microSD card (64GB+ recommended)
- CSI camera or USB camera
- Network connection (Ethernet or WiFi)

Software:

- JetPack 4.6+ flashed on SD card
- SSH access enabled

2.2 Initial Jetson Setup

bash

```bash
# SSH into Jetson (replace IP with your Jetson's IP)
ssh newcastleuni@192.168.50.94

# Update system
sudo apt update && sudo apt upgrade -y

# Install essential packages
sudo apt install -y python3-pip python3-dev
sudo apt install -y curl wget git
```

2.3 Install k3s (Lightweight Kubernetes) - Enhanced Setup

bash

```bash
# Method 1: Automatic setup with troubleshooting script
wget https://raw.githubusercontent.com/your-repo/setup_k3s.sh
chmod +x setup_k3s.sh
./setup_k3s.sh install

# Method 2: Manual installation
curl -sfL https://get.k3s.io | sh -

# Enable k3s service
sudo systemctl enable k3s

# Set up kubectl access
sudo chmod 644 /etc/rancher/k3s/k3s.yaml
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
echo 'export KUBECONFIG=/etc/rancher/k3s/k3s.yaml' >> ~/.bashrc

# Verify installation
kubectl get nodes
```

2.4 Troubleshoot k3s Issues (if kubectl not found)

If you get "kubectl not found" errors, run the troubleshooting script:

bash

```
# Copy the setup script to Jetson
scp setup_k3s.sh newcastleuni@192.168.50.94:~/

# On Jetson - fix k3s issues
chmod +x setup_k3s.sh

# Check current status
./setup_k3s.sh check

# Fix common issues
./setup_k3s.sh fix

# Run full diagnostics if still having issues
./setup_k3s.sh diagnose

# Create kubectl wrapper (alternative access method)
./setup_k3s.sh wrapper
```

2.5 Alternative kubectl Access Methods

The auto-deployment script now handles multiple kubectl access methods:

1. **Standard kubectl**: `kubectl` (if in PATH)
2. **k3s kubectl**: `sudo k3s kubectl` (always works)
3. **With KUBECONFIG**: `KUBECONFIG=/etc/rancher/k3s/k3s.yaml kubectl`
4. **Wrapper script**: `~/kubectl_wrapper.sh` (auto-generated)

2.4 Install Python Dependencies (for local development/testing)

bash

```
# Install basic Python packages
pip3 install --user opencv-python numpy requests flask pillow scikit-learn

# Note: MLflow is NOT installed on Jetson - models run in containers
```

2.5 Test Camera Connection

bash

```
# Test CSI camera (if using)
gst-launch-1.0 nvarguscamerasrc ! nvoverlaysink

# Test USB camera (if using)
ls /dev/video*

# Install v4l-utils for camera testing
sudo apt install -y v4l-utils
v4l2-ctl --list-devices
```

Part 3: Network Configuration

3.1 SSH Key Setup (for passwordless deployment)

On Edge Server:

bash

```
# Generate SSH key if you don't have one
ssh-keygen -t rsa -b 4096

# Copy public key to Jetson
ssh-copy-id newcastleuni@192.168.50.94

# Test passwordless login
ssh newcastleuni@192.168.50.94 'echo "Connection successful"'
```

3.2 Network Discovery

bash

```
# Find Jetson IP address
# On Jetson:
hostname -I

# On Edge Server, test connectivity:
ping 192.168.50.94
```

Part 4: Training and Deployment Workflow

4.1 Train Your First Model

On Edge Server:

```bash
cd ~/face_recognition_system/edge_server
source face_recognition_env/bin/activate

# Start MLflow UI (optional, for monitoring)
./mlflow_ui.sh edge &
# Access at http://your-edge-server-ip:5000

# Run training (interactive mode)
python edge_training.py
```

When prompted, choose:

- Person name: `amrit` (or your person's name)
- Deployment type: `2` (Lightweight - recommended for Jetson)

4.2 Manual Training (programmatic)

python

```python
from edge_training import EdgeFaceRecognitionTrainer

# Initialize trainer
trainer = EdgeFaceRecognitionTrainer()

# Register a person
person_name = "amrit"
image_paths = [
    f"images/{person_name}/{person_name}_1.jpg",
    f"images/{person_name}/{person_name}_2.jpg",
    f"images/{person_name}/{person_name}_3.jpg",
    f"images/{person_name}/{person_name}_4.jpg",
    f"images/{person_name}/{person_name}_5.jpg"
]

success, person_id = trainer.train_few_shot_model(person_name, image_paths)
if success:
    print(f"Successfully registered {person_name}")

    # Create lightweight deployment
    package = trainer.create_lightweight_deployment()
    print(f"Deployment package: {package}")
```

4.3 Deploy to Jetson (Lightweight Method - Recommended)

bash

```bash
# On Edge Server - after training
ls -la lightweight-deployment-v*.tar.gz

# Copy to Jetson
scp lightweight-deployment-v1.tar.gz newcastleuni@192.168.50.94:~/

# SSH to Jetson and deploy
ssh newcastleuni@192.168.50.94

# On Jetson:
tar -xzf lightweight-deployment-v1.tar.gz
cd lightweight_deployment
ls -la  # Should see: Dockerfile, model_server.py, requirements.txt, *.json, *.pkl

# Build lightweight Docker image
./build.sh

# Verify image size (should be < 1GB)
sudo docker images | grep face-recognition-lightweight
```

4.4 Alternative: Full K8s Deployment (if you have enough space)

bash

```bash
# On Edge Server
./build_k8s.sh 1 192.168.50.94 newcastleuni
./deploy_to_jetson.sh 1 192.168.50.94 newcastleuni

# On Jetson
cd k8s_deployment
./deploy.sh
```

Part 5: Automated Kubernetes Deployment

5.1 Automated Deployment Process

The system now includes an **automatic deployment script** that handles all Kubernetes manifests and pod deployment automatically. No manual YAML creation needed!

Option A: Integrated Deployment (Recommended)

bash

```
# On Edge Server - after training
python edge_training.py
# Choose option 2 (Lightweight deployment)

# Copy the deployment package to Jetson
scp lightweight-deployment-v1.tar.gz newcastleuni@192.168.50.94:~/

# SSH to Jetson and auto-deploy
ssh newcastleuni@192.168.50.94
tar -xzf lightweight-deployment-v1.tar.gz
cd lightweight_deployment

# Run the build script with auto-deployment
./build.sh
# When prompted, press 'y' for automatic k3s deployment
```

Option B: Standalone Auto-Deployment

bash

```
# Copy auto-deployment script to Jetson
scp auto_deploy.py newcastleuni@192.168.50.94:~/

# On Jetson - with pre-built images
ssh newcastleuni@192.168.50.94
python3 auto_deploy.py --model-version 1
```

5.2 What the Auto-Deployment Does

The `auto_deploy.py` script automatically:

1. ✅ **Checks Prerequisites**: Verifies k3s and Docker are available
2. 📁 **Creates Namespace**: Sets up isolated `face-recognition` namespace
3. 📦 **Loads Images**: Imports Docker images into k3s
4. 🚀 **Deploys Pods**: Creates model service and inference app deployments
5. ⏳ **Waits for Ready**: Monitors pod startup and health checks
6. 🔍 **Verifies**: Tests deployment and provides access URLs
7. 📝 **Creates Scripts**: Generates management scripts for ongoing operations

5.3 Generated Management Scripts

After deployment, you'll have these scripts on Jetson:

```bash
# Check system status
./status.sh

# View logs
./logs.sh model      # Model service logs
./logs.sh inference  # Inference app logs
./logs.sh all        # Both services

# Clean up deployment
./cleanup.sh
```

5.4 Deployment Output Example

```
🚀 Starting Face Recognition System Deployment
==================================================
🔍 Checking prerequisites...
✅ Prerequisites check passed
📁 Creating namespace: face-recognition
✅ Namespace face-recognition created/updated
📦 Loading Docker images into k3s...
✅ Model image loaded successfully
✅ Inference image built successfully
🚀 Deploying model service...
✅ Model service deployed successfully
📹 Deploying inference application...
✅ Inference application deployed successfully
⏳ Waiting for pods to be ready...
✅ Model service pod is running
✅ Inference application pod is running
🔍 Verifying deployment...
✅ Model service health check passed
📝 Creating management scripts...

==================================================
🎉 DEPLOYMENT SUCCESSFUL!
==================================================
🌐 Web Interface: http://192.168.50.94:30080
📊 System Status: ./status.sh
📋 View Logs: ./logs.sh
🧹 Cleanup: ./cleanup.sh
```

Part 6: Verification and Testing (Simplified)

6.1 Automatic Verification

The auto-deployment script handles verification automatically, but you can also check manually:

bash

```bash
# Check deployment status
./status.sh

# Or manually check pods
kubectl get pods -n face-recognition
kubectl get services -n face-recognition
```

6.2 Access Web Interface

The system automatically provides the access URL after deployment:

bash

```
# Access web interface (replace with your Jetson IP)
http://192.168.50.94:30080
```

6.3 Manual Testing (if needed)

bash

```
# Test model service API
kubectl port-forward -n face-recognition service/face-recognition-model-service 8080:80 &
curl http://localhost:8080/ping

# View detailed pod information
kubectl describe pod -n face-recognition -l app=face-recognition-model
```:
            path: /ping
            port: 5000
          initialDelaySeconds: 30
          periodSeconds: 10
---
apiVersion: v1
kind: Service
metadata:
  name: face-recognition-model-service
spec:
  selector:
    app: face-recognition-model
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: ClusterIP
EOF

Apply the deployment
kubectl apply -f model-service-lightweight.yaml
```

## 5.2 Deploy Inference Application Pod
```

bash

```
# Copy inference application files
scp jetson_inference.py newcastleuni@192.168.50.94:~/
scp Dockerfile.inference newcastleuni@192.168.50.94:~/
scp requirements_jetson.txt newcastleuni@192.168.50.94:~/

# On Jetson - build inference app
sudo docker build -t face-recognition-inference:v1 -f Dockerfile.inference .

# Create inference app manifest
cat > inference-app.yaml << 'EOF'
apiVersion: apps/v1
kind: Deployment
metadata:
  name: face-recognition-inference
  labels:
    app: face-recognition-inference
spec:
  replicas: 1
  selector:
    matchLabels:
      app: face-recognition-inference
  template:
    metadata:
      labels:
        app: face-recognition-inference
    spec:
      containers:
      - name: inference-app
        image: face-recognition-inference:v1
        env:
        - name: MODEL_SERVICE_URL
          value: "http://face-recognition-model-service"
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        volumeMounts:
        - name: camera-device
```

```
        mountPath: /dev/video0
      securityContext:
        privileged: true
    volumes:
    - name: camera-device
      hostPath:
        path: /dev/video0
    nodeSelector:
      kubernetes.io/arch: arm64
---
apiVersion: v1
kind: Service
metadata:
  name: face-recognition-inference-service
spec:
  selector:
    app: face-recognition-inference
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
    nodePort: 30080
  type: NodePort
EOF

# Deploy inference application
kubectl apply -f inference-app.yaml
```

Part 6: Verification and Testing

6.1 Check Deployment Status

bash

```
# On Jetson
kubectl get pods
kubectl get services
kubectl describe pod <pod-name>  # if any issues

# Check logs
kubectl logs deployment/face-recognition-model
kubectl logs deployment/face-recognition-inference
```

6.2 Test Model Service

bash

```
# Test model service health
kubectl port-forward service/face-recognition-model-service 8080:80 &
curl http://localhost:8080/ping

# Test inference (need base64 encoded image)
curl -X POST http://localhost:8080/invocations \
  -H "Content-Type: application/json" \
  -d '{"instances": [{"face_image": "BASE64_ENCODED_IMAGE_HERE"}]}'
```

6.3 Access Web Interface

bash

```
# Get Jetson IP and NodePort
JETSON_IP=$(hostname -I | awk '{print $1}')
NODE_PORT=$(kubectl get service face-recognition-inference-service -o jsonpath='{.spec.ports[0]

echo "Access web interface at: http://$JETSON_IP:$NODE_PORT"
```

Part 7: Monitoring and Maintenance (Automated)

7.1 System Monitoring

bash

```
# On Jetson - check overall system status
./status.sh

# Monitor resource usage
kubectl top pods -n face-recognition
tegrastats  # Jetson system stats

# View real-time logs
./logs.sh model     # Follow model service logs
./logs.sh inference # Follow inference app logs
```

7.2 MLflow Monitoring (Edge Server)

bash

```
# On Edge Server - monitor training experiments
./mlflow_ui.sh edge
# Access at http://edge-server-ip:5000
```

7.3 Automated Health Checks

The auto-deployment script includes built-in health checks:

- **Readiness Probes**: Ensure pods are ready to receive traffic
- **Liveness Probes**: Restart unhealthy pods automatically
- **Service Health**: Automatic `/ping` endpoint testing

7.4 Troubleshooting with Auto-Generated Scripts

bash

```
# Quick system diagnosis
./status.sh

# Check specific issues
kubectl describe pod -n face-recognition <pod-name>

# View recent events
kubectl get events -n face-recognition --sort-by='.lastTimestamp'

# Restart a deployment
kubectl rollout restart deployment/face-recognition-model -n face-recognition
```

Part 8: Model Updates (Streamlined)

8.1 Update Workflow

bash

```bash
# On Edge Server - train new model version
python edge_training.py
# Add new person images and retrain

# Deploy new version to Jetson
scp lightweight-deployment-v2.tar.gz newcastleuni@192.168.50.94:~/

# On Jetson - update deployment
ssh newcastleuni@192.168.50.94
tar -xzf lightweight-deployment-v2.tar.gz
cd lightweight_deployment

# Auto-deploy new version
python3 auto_deploy.py --model-version 2
```

8.2 Rolling Updates

bash

```bash
# Update model service with new version
kubectl set image deployment/face-recognition-model model-server=face-recognition-lightweight:v

# Monitor rollout
kubectl rollout status deployment/face-recognition-model -n face-recognition
```

Part 9: Advanced Features

9.1 Auto-Deployment Script Options

bash

```bash
# Deploy specific model version
python3 auto_deploy.py --model-version 3

# Check deployment status
python3 auto_deploy.py --status

# Clean up existing deployment
python3 auto_deploy.py --cleanup
```

9.2 Scaling and Resource Management

The auto-deployment script automatically configures:

- **Resource Limits**: Prevents pods from consuming too much memory/CPU
- **Namespace Isolation**: Keeps face recognition system separate
- **Health Monitoring**: Automatic restart of failed pods
- **Storage Volumes**: Persistent camera device access

9.3 Development Features

```bash
# Build and test locally before deployment
cd lightweight_deployment
sudo docker build -t face-recognition-lightweight:test .
sudo docker run -p 5000:5000 face-recognition-lightweight:test

# Quick deployment for testing
python3 auto_deploy.py --model-version test
```

Part 10: Troubleshooting (Automated Solutions)

10.1 Common Issues and Auto-Fixes

Pod Not Starting:

```bash
# Auto-diagnosis
./status.sh
./logs.sh model  # Check specific logs

# Auto-fix: The script includes automatic retries and health checks
python3 auto_deploy.py --cleanup
python3 auto_deploy.py --model-version 1
```

Camera Access Issues:

bash

```
# The auto-deployment script automatically:
# - Mounts /dev/video0 device
# - Sets privileged security context
# - Creates proper volume mounts

# Manual check
ls -la /dev/video*
kubectl describe pod -n face-recognition -l app=face-recognition-inference
```

Resource Constraints:

bash

```
# Auto-configured resource limits prevent system overload
# View current usage
kubectl top pods -n face-recognition

# The script automatically sets:
# - Memory: 256Mi request, 512Mi limit
# - CPU: 250m request, 500m limit
```

10.2 Auto-Recovery Features

The system includes automatic recovery:

1. **Pod Restart**: Unhealthy pods restart automatically
2. **Health Checks**: Built-in readiness and liveness probes
3. **Resource Management**: Prevents resource exhaustion
4. **Error Logging**: Comprehensive logging for debugging

Quick Start Summary (Automated)

1-Step Edge Training:

bash

```
python edge_training.py  # Choose lightweight deployment
```

1-Step Jetson Deployment:

bash

```
# Copy and auto-deploy
scp lightweight-deployment-v1.tar.gz newcastleuni@192.168.50.94:~/
ssh newcastleuni@192.168.50.94 'tar -xzf lightweight-deployment-v1.tar.gz && cd lightweight_dep
# Press 'y' when prompted for auto-deployment
```

1-Step Access:

bash

```
# Web interface automatically available at:
http://192.168.50.94:30080
```

Management Commands:

bash

```
./status.sh      # System status
./logs.sh all    # View logs
./cleanup.sh     # Remove deployment
```

Benefits of Auto-Deployment

✅ **No Manual YAML**: Scripts generate all Kubernetes manifests automatically
✅ **Error Handling**: Built-in validation and error recovery
✅ **Health Monitoring**: Automatic health checks and pod restarts
✅ **Resource Management**: Optimal resource allocation for Jetson
✅ **Management Tools**: Auto-generated scripts for ongoing operations
✅ **Namespace Isolation**: Clean separation from other workloads
✅ **Rolling Updates**: Seamless model version updates
✅ **Troubleshooting**: Comprehensive logging and diagnostic tools

The automated deployment eliminates the need for manual Kubernetes manifest creation and provides a production-ready deployment with monitoring, health checks, and management tools.
