

WeldVision X5: Docker Deployment Guide (Alternative)

For those who prefer containerized deployment

Overview

This guide provides Docker-based deployment of WeldVision X5 for easier portability and isolation.

Benefits of Docker

- **Consistency:** Same environment on all machines
- **Isolation:** No dependency conflicts
- **Portability:** Deploy to any system with Docker
- **Scalability:** Easy to run multiple instances
- **Cleanup:** Remove with single command

Requirements

- Docker & Docker Compose installed on RDK X5
 - 2GB+ available storage
 - 1GB+ RAM available
-

Part 1: Install Docker on RDK X5

Step 1: Install Docker

```
ssh root@<RDK_X5_IP>

# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker root

# Install Docker Compose
sudo apt-get install -y python3-pip
sudo pip3 install docker-compose

# Verify installation
docker --version
docker-compose --version
```

Step 2: Test Docker

```
# Test docker daemon
docker ps

# Should show: CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES
# (empty if no containers running)
```

Part 2: Create Docker Configuration

Create Dockerfile for Backend

File: /opt/weldvision/backend/Dockerfile

```

# Use Python 3.10 slim image
FROM python:3.10-slim

# Install system dependencies
RUN apt-get update && apt-get install -y \
    libopencv-dev \
    python3-opencv \
    libsm6 \
    libxext6 \
    libxrender-dev \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Copy requirements
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Expose port
EXPOSE 5000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD python3 -c "import requests; requests.get('http://localhost:5000/api/health')"

# Run application
CMD ["python3", "app.py"]

```

Create Dockerfile for Frontend

File: /opt/weldvision/Dockerfile

```

# Build stage
FROM node:18-alpine as builder

WORKDIR /app

COPY package.json package-lock.json ./
RUN npm ci

COPY . .
RUN npm run build

# Production stage
FROM node:18-alpine

WORKDIR /app

# Install serve to run the app
RUN npm install -g serve

COPY --from=builder /app/dist ./dist

EXPOSE 3000

CMD ["serve", "-s", "dist", "-l", "3000"]

```

Create docker-compose.yml

File: /opt/weldvision/docker-compose.yml

```

version: '3.8'

```

```
services:
  # Backend API
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: weldvision-backend
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
      - PYTHONUNBUFFERED=1
      - DATABASE_URL=sqlite:///app/weld_data.db
    volumes:
      - ./backend/weld_data.db:/app/weld_data.db
    networks:
      - weldvision-net
    restart: unless-stopped
    depends_on:
      - db

  # Frontend UI
  frontend:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: weldvision-frontend
    ports:
      - "3000:3000"
    environment:
      - VITE_API_URL=http://backend:5000
    networks:
      - weldvision-net
    restart: unless-stopped
    depends_on:
      - backend

  # SQLite Database (optional - for volume management)
  db:
    image: alpine:latest
    container_name: weldvision-db
    volumes:
      - ./backend/weld_data.db:/db/weld_data.db
    networks:
      - weldvision-net
    restart: unless-stopped

  # Nginx Reverse Proxy (optional)
  nginx:
    image: nginx:alpine
    container_name: weldvision-nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl:/etc/nginx/ssl:ro
    networks:
      - weldvision-net
    restart: unless-stopped
    depends_on:
      - frontend
      - backend

networks:
  weldvision-net:
    driver: bridge
```

```
volumes:  
  weldvision-data:  
    driver: local
```

Create Nginx Config (Optional)

File: `/opt/weldvision/nginx.conf`

```

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css text/xml text/javascript
        application/x-javascript application/xml+rss;

    upstream backend {
        server backend:5000;
    }

    upstream frontend {
        server frontend:3000;
    }

    server {
        listen 80;
        server_name _;

        # Frontend
        location / {
            proxy_pass http://frontend;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        # Backend API
        location /api/ {
            proxy_pass http://backend/api/;
            proxy_http_version 1.1;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}

```

Part 3: Build and Deploy with Docker

Step 1: Build Images

```
cd /opt/weldvision

# Build all images
docker-compose build

# View built images
docker images | grep weldvision
```

Step 2: Start Services

```
# Start all services in background
docker-compose up -d

# Check status
docker-compose ps

# View logs
docker-compose logs -f backend
docker-compose logs -f frontend
```

Step 3: Verify Services

```
# Test backend API
curl http://localhost:5000/api/health

# Expected response:
# {"status": "ok"}

# Test frontend
curl http://localhost:3000

# Should return HTML content
```

Part 4: Docker Operations

Viewing Logs

```
# All services
docker-compose logs

# Specific service
docker-compose logs backend
docker-compose logs frontend

# Follow logs
docker-compose logs -f backend

# Last 50 lines
docker-compose logs --tail=50 backend
```

Managing Containers

```
# Stop all services
docker-compose stop

# Start all services
docker-compose start

# Restart specific service
docker-compose restart backend

# Remove containers (keep images)
docker-compose down

# Remove everything (containers + volumes)
docker-compose down -v

# Remove images
docker-compose down --rmi all
```

Executing Commands in Container

```
# Interactive shell in backend
docker-compose exec backend /bin/bash

# Run Python command
docker-compose exec backend python3 -c "import cv2; print(cv2.__version__)"

# Check database
docker-compose exec backend sqlite3 /app/weld_data.db ".tables"
```

Resource Monitoring

```
# View container resource usage
docker stats weldvision-backend
docker stats weldvision-frontend

# View detailed info
docker inspect weldvision-backend

# View logs with timestamps
docker-compose logs --timestamps backend
```

Part 5: Advanced Docker Configuration

Using Docker Hub (Optional)

```
# Login to Docker Hub
docker login

# Tag image
docker tag weldvision-backend:latest username/weldvision-backend:latest

# Push to registry
docker push username/weldvision-backend:latest
docker push username/weldvision-frontend:latest

# Pull on another machine
docker pull username/weldvision-backend:latest
docker pull username/weldvision-frontend:latest
```

Docker Networking

```
# List networks
docker network ls

# Inspect network
docker network inspect weldvision_weldvision-net

# Custom network bridge
docker network create weldvision-net

# Connect container to network
docker network connect weldvision-net weldvision-backend
```

Volume Management

```
# List volumes
docker volume ls | grep weldvision

# Create named volume
docker volume create weldvision-data

# Inspect volume
docker volume inspect weldvision-data

# Backup volume
docker run --rm -v weldvision_weldvision-data:/data -v $(pwd):/backup \
  alpine tar czf /backup/weldvision-data.tar.gz -C /data .

# Restore volume
docker run --rm -v weldvision_weldvision-data:/data -v $(pwd):/backup \
  alpine tar xzf /backup/weldvision-data.tar.gz -C /data
```

Troubleshooting Docker

Problem: Container Won't Start

```
# View detailed logs
docker-compose logs backend

# Check image exists
docker images | grep weldvision

# Rebuild image
docker-compose build --no-cache backend

# Start with verbose output
docker-compose up backend
```

Problem: Port Already in Use

```
# Find what's using port 5000
sudo lsof -i :5000

# Kill process
sudo kill -9 <PID>

# Or use different port in docker-compose.yml:
# ports:
#   - "5001:5000"
```

Problem: Slow Performance

```
# Check resource limits
docker stats weldvision-backend

# Increase memory limit in docker-compose.yml:
# services:
#   backend:
#     mem_limit: 2g
#     cpus: "2"

# Restart services
docker-compose restart
```

Problem: Camera Access in Container

```
# Docker needs access to /dev/video*
# Modify docker-compose.yml:

services:
  backend:
    devices:
      - /dev/video0:/dev/video0
      - /dev/video1:/dev/video1
    privileged: true
```

Docker Compose Reference

Command Syntax

```
docker-compose [OPTIONS] COMMAND

# Common commands:
docker-compose build           # Build images
docker-compose up -d          # Start services (background)
docker-compose down           # Stop and remove
docker-compose ps             # List containers
docker-compose logs           # View logs
docker-compose exec SERVICE CMD # Execute command
docker-compose restart SERVICE # Restart service
docker-compose pull           # Update images
```

Full docker-compose.yml Options

```
services:
  SERVICE_NAME:
    image: image_name
    build:
      context: .
      dockerfile: Dockerfile
    container_name: container_name
    ports:
      - "HOST:CONTAINER"
    volumes:
      - /host/path:/container/path
    environment:
      - VAR=value
    networks:
      - network_name
    restart: always|unless-stopped|on-failure
    depends_on:
      - other_service
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000"]
      interval: 30s
      timeout: 10s
      retries: 3
```

Production Deployment

Security Best Practices

1. Use named volumes for data:

```
volumes:
  weldvision-data:
    driver: local
```

2. Set resource limits:

```
services:
  backend:
    mem_limit: 1g
    cpus: "1.5"
```

3. Use environment files:

```
# Create .env file
echo "FLASK_ENV=production" > .env

# Reference in docker-compose.yml
env_file:
  - .env
```

4. Enable restart policy:

```
restart_policy:
  condition: on-failure
  delay: 5s
  max_attempts: 5
```

5. Use health checks:

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:5000/api/health"]
  interval: 30s
  timeout: 10s
  retries: 3
```

Comparison: Docker vs Systemd

Feature	Docker	Systemd
Setup Time	~5 min	~10 min
Isolation	Complete	Minimal
Performance	~2% overhead	Minimal
Portability	Excellent	Limited
Debugging	Easy (logs)	Terminal access
Resource Control	Fine-grained	Basic
Learning Curve	Medium	Low

Use Docker if:

- Deploying to multiple systems
- Need strict isolation
- Prefer containerized approach
- Plan to scale horizontally

Use Systemd if:

- Single deployment
- Need bare-metal performance
- Simpler maintenance
- Direct system access

Quick Reference

```
# Deploy
docker-compose up -d

# View status
docker-compose ps

# View logs
docker-compose logs -f

# Stop
docker-compose stop

# Remove everything
docker-compose down -v

# Access backend
docker-compose exec backend /bin/bash

# Access frontend
docker-compose exec frontend /bin/sh
```

Alternative: Run Individual Containers

```
# Build backend image
docker build -t weldvision-backend:latest ./backend

# Run backend
docker run -d \
  --name weldvision-backend \
  -p 5000:5000 \
  -v $(pwd)/backend/weld_data.db:/app/weld_data.db \
  weldvision-backend:latest

# Build frontend image
docker build -t weldvision-frontend:latest .

# Run frontend
docker run -d \
  --name weldvision-frontend \
  -p 3000:3000 \
  -e VITE_API_URL=http://localhost:5000 \
  weldvision-frontend:latest
```

Summary

Docker deployment is recommended for:

- Cloud deployments
- Multiple RDK X5 devices
- Microservices architecture
- Standardized environments

For single RDK X5 on-site, systemd is simpler and faster.

Choose based on your deployment needs!

See *COMPLETE_DEPLOYMENT_AND_REMOTE_ACCESS_GUIDE.md* for *systemd*-based deployment.