Docker Usage Guide for CineFlux-AutoXML

This guide explains how to use Docker to build, run, and deploy the CineFlux-AutoXML application.

Table of Contents

Prerequisites

- Docker (version 20.10.0 or higher)
- Docker Compose (version 2.0.0 or higher)

Docker Configuration Overview

The CineFlux-AutoXML application uses Docker for both development and production environments. The configuration includes:

- **Dockerfile**: Multi-stage build for production deployment
- **Dockerfile.dev**: Development environment with hot-reloading
- **docker-compose.yml**: Defines services for both development and production
- **.dockerignore**: Excludes unnecessary files from the Docker context
- **docker/nginx.conf**: Nginx configuration for serving the production build

Development Environment

The development environment uses a dedicated Dockerfile with volume mounting for hot-reloading.

Starting the Development Environment

```
# Start the development environment
docker-compose up app-dev

# Start in detached mode
docker-compose up -d app-dev
```

```
# Rebuild and start (after dependency changes)
docker-compose up --build app-dev
```

The development server will be available at http://localhost:3000.

Features of the Development Environment

- Hot-reloading: Changes to source files are immediately reflected
- Volume mounting: Local files are mounted into the container
- Node modules isolation: A dedicated volume for node_modules prevents platform-specific issues
- Environment variables: Development-specific environment variables are set

Production Environment

The production environment uses a multi-stage build to create an optimized, secure image.

Building and Running the Production Environment

```
# Build and start the production environment
docker-compose up app-prod

# Start in detached mode
docker-compose up -d app-prod

# Rebuild and start (after code changes)
docker-compose up --build app-prod
```

The production server will be available at http://localhost:8080.

Features of the Production Environment

- Multi-stage build: Smaller image size and improved security
- Nginx server: Optimized for serving static files
- Gzip compression: Reduces bandwidth usage
- Cache control: Improves loading performance for returning visitors
- Security headers: Includes CORS headers for WebAssembly support
- Non-root user: Runs as a non-privileged user for improved security

Environment Variables

Environment variables can be configured in the `docker-compose.yml` file or passed at runtime.

Default Environment Variables

- `NODE_ENV`: Sets the Node.js environment (development/production)

- `VITE_APP_VERSION`: Application version
- `VITE_WASM_CDN_URL`: URL for WebAssembly modules (empty for local files)
- `VITE_FEATURE_FLAGS`: JSON string of feature flags

Overriding Environment Variables

```
# Override environment variables at runtime
docker-compose up -e VITE_FEATURE_FLAGS='{"debugMode":true}' app-prod
```

Common Docker Commands

Container Management

```
# List running containers
docker ps

# List all containers (including stopped)
docker ps -a

# Stop a container
docker stop <container_id_or_name>

# Remove a container
docker rm <container_id_or_name>
```

Image Management

```
# List images
docker images

# Remove an image
docker rmi <image_id_or_name>

# Build an image
docker build -t cineflux-autoxml:latest .
```

Volume Management

```
# List volumes
docker volume ls

# Remove a volume
docker volume rm <volume_name>

# Clean up unused volumes
docker volume prune
```

Logs and Debugging

```
# View container logs
docker logs <container_id_or_name>

# Follow container logs
docker logs -f <container_id_or_name>

# Execute a command in a running container
docker exec -it <container_id_or_name> /bin/sh
```

Deployment Scenarios

Local Development

Use the development environment for local development with hot-reloading:

```
docker-compose up app-dev
```

Production Deployment

For production deployment, use the production environment:

```
docker-compose up -d app-prod
```

CI/CD Pipeline

For CI/CD pipelines, build and push the production image:

```
# Build the production image
docker build -t cineflux-autoxml:latest .

# Tag the image for your registry
docker tag cineflux-autoxml:latest your-registry.com/cineflux-autoxml:latest

# Push the image to your registry
docker push your-registry.com/cineflux-autoxml:latest
```

Kubernetes Deployment

For Kubernetes deployment, use the production image with appropriate resource limits:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cineflux-autoxml
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cineflux-autoxml
  template:
    metadata:
      labels:
        app: cineflux-autoxml
    spec:
      containers:
      - name: cineflux-autoxml
        image: your-registry.com/cineflux-autoxml:latest
        ports:
        - containerPort: 8080
        resources:
          limits:
            cpu: "1"
            memory: "512Mi"
          requests:
            cpu: "0.5"
            memory: "256Mi"
```

Troubleshooting

Common Issues and Solutions

Container Exits Immediately **Issue**: The container starts and then exits immediately.

Solution: Check the container logs for errors:

```
docker logs <container_id_or_name>
```

Hot-Reloading Not Working **Issue**: Changes to source files are not reflected in the development environment.

Solution: Ensure that the volume mounting is correct and that the development server is running with the correct options:

```bash
# Restart the development environment
docker-compose down
docker-compose up --build app-dev
```

WebAssembly Modules Not Loading **Issue**: WebAssembly modules fail to load in the browser.

Solution: Ensure that the CORS headers are correctly set in the Nginx configuration:

```bash
# Check the Nginx configuration
docker exec -it <container_id_or_name> cat /etc/nginx/conf.d/default.conf
```

```
# Rebuild and restart the container
docker-compose up --build app-prod
```

Out of Memory During Build **Issue**: The build process fails with an out of memory error.

Solution: Increase the memory allocated to Docker:

```
# Check current resource limits
docker info

# Increase memory limit in Docker Desktop settings
# or use the --memory option when running the container
docker-compose up --build --memory=4g app-prod
```

Permission Issues **Issue**: Permission denied errors when accessing files in the container.

Solution: Check the file permissions and ownership:

```
# Check file permissions in the container
docker exec -it <container_id_or_name> ls -la /usr/share/nginx/html

# Fix permissions if necessary
docker exec -it <container_id_or_name> chown -R appuser:appuser /usr/share/nginx/html
```

Getting Help

If you encounter issues not covered in this guide, please:

1. Check the Docker and Docker Compose documentation
2. Review the application logs
3. Open an issue in the project repository with detailed information about the problem

Additional Resources

- Docker Documentation
- Docker Compose Documentation
- Nginx Documentation
- Vite Documentation