Database Deployment Guide

ActaLog supports three database backends with Docker deployment:

- **SQLite** - Single-file database (default, best for single-server)
- **PostgreSQL** - Production-ready SQL database (recommended for multi-instance)
- **MariaDB/MySQL** - Alternative production SQL database

Quick Start by Database

SQLite (Default - Simplest)

Best for: Development, single-server production, small-scale deployments

```
cd docker

# Use default compose file
cp .env.example .env
vim .env  # Set GITHUB_OWNER, TAG, JWT_SECRET

# Start
docker compose up -d
```

Pros:

- No separate database container
- Zero configuration
- Automatic backups (single file)
- Perfect for embedded deployments

Cons:

- Single connection writer
- Not ideal for high-concurrency

PostgreSQL (Recommended for Production)

Best for: Production deployments, high concurrency, data integrity

```
cd docker

# Use PostgreSQL compose file
cp .env.postgres .env
vim .env  # Set passwords and JWT_SECRET

# Start PostgreSQL + ActaLog
docker compose -f docker-compose.postgres.yml up -d

# View logs
docker compose -f docker-compose.postgres.yml logs -f

# Check health
curl http://localhost:8080/health
```

Configuration (.env):

```
GITHUB_OWNER=yourusername
TAG=latest
DB_NAME=actalog
DB_USER=actalog
DB_PASSWORD=super_secure_password_here
JWT_SECRET=your_jwt_secret_here
```

Pros:

- Best for concurrent users
- ACID compliance
- Advanced features (JSON, full-text search)
- Battle-tested in production

Cons:

- Requires separate container
- More memory usage (~50MB)

Connection String:

```
postgresql://actalog:password@postgres:5432/actalog?sslmode=disable
```

MariaDB (Alternative Production Option)
Best for: Teams familiar with MySQL, existing MySQL infrastructure

```
cd docker

# Use MariaDB compose file
cp .env.mariadb .env
vim .env  # Set passwords and JWT_SECRET

# Start MariaDB + ActaLog
docker compose -f docker-compose.mariadb.yml up -d

# View logs
docker compose -f docker-compose.mariadb.yml logs -f

# Check health
curl http://localhost:8080/health
```

Configuration (.env):

```
GITHUB_OWNER=yourusername
TAG=latest
DB_NAME=actalog
DB_USER=actalog
DB_PASSWORD=super_secure_password_here
DB_ROOT_PASSWORD=different_root_password_here
JWT_SECRET=your_jwt_secret_here
```

Pros:

- MySQL-compatible
- Good performance

- Wide ecosystem support
- Familiar to many developers

Cons:

- Slightly more resource usage than PostgreSQL
- Less advanced JSON support

Connection String:

```
mysql://actalog:password@mariadb:3306/actalog
```

Database Comparison

Feature	SQLite	PostgreSQL	MariaDB
Setup Complexity	⭐ Easiest	⭐⭐ Moderate	⭐⭐ Moderate
Concurrent Users	1-10	100+	100+
Memory Usage	~5MB	~50MB	~60MB
Backup	Copy file	pg_dump	mysqldump
JSON Support	Limited	Excellent	Good
Full-text Search	Basic	Advanced	Good
Recommended For	Dev, Small	Production	Production

Migration Between Databases

SQLite → PostgreSQL

```
# 1. Export from SQLite
sqlite3 actalog.db .dump > dump.sql

# 2. Convert to PostgreSQL format (manual)
# - Change AUTOINCREMENT to SERIAL
# - Fix datetime formats
# - Adjust syntax

# 3. Import to PostgreSQL
docker compose -f docker-compose.postgres.yml exec postgres \
  psql -U actalog -d actalog -f /tmp/dump.sql
```

Using ActaLog Backup/Restore

ActaLog has built-in backup endpoints that work across databases:

```
# 1. Create backup on SQLite deployment
curl -X POST http://localhost:8080/api/admin/backups \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"description":"Pre-migration backup"}'

# 2. Download backup
```

```
curl http://localhost:8080/api/admin/backups/1/download \
  -H "Authorization: Bearer $TOKEN" \
  -o backup.zip

# 3. Deploy with PostgreSQL
docker compose -f docker-compose.postgres.yml up -d

# 4. Restore backup
curl -X POST http://localhost:8080/api/admin/backups/restore \
  -H "Authorization: Bearer $TOKEN" \
  -F "file=@backup.zip"
```

Database Maintenance

PostgreSQL Maintenance

```
# Enter PostgreSQL shell
docker compose -f docker-compose.postgres.yml exec postgres \
  psql -U actalog -d actalog

# Vacuum (clean up)
VACUUM ANALYZE;

# Check database size
SELECT pg_size_pretty(pg_database_size('actalog'));

# List tables
\dt

# Backup
docker compose -f docker-compose.postgres.yml exec postgres \
  pg_dump -U actalog actalog > backup.sql

# Restore
docker compose -f docker-compose.postgres.yml exec -T postgres \
  psql -U actalog -d actalog < backup.sql
```

MariaDB Maintenance

```
# Enter MariaDB shell
docker compose -f docker-compose.mariadb.yml exec mariadb \
  mysql -u actalog -p actalog

# Check database size
SELECT table_schema AS "Database",
  ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS "Size (MB)"
FROM information_schema.TABLES
WHERE table_schema = 'actalog'
GROUP BY table_schema;

# Optimize tables
OPTIMIZE TABLE users, workouts, workout_movements;

# Backup
```

```
docker compose -f docker-compose.mariadb.yml exec mariadb \
  mysqldump -u actalog -p actalog > backup.sql

# Restore
docker compose -f docker-compose.mariadb.yml exec -T mariadb \
  mysql -u actalog -p actalog < backup.sql
```

SQLite Maintenance

```
# Backup (just copy the file)
docker compose cp actalog:/app/data/actalog.db ./backup.db

# Restore
docker compose cp ./backup.db actalog:/app/data/actalog.db
docker compose restart actalog

# Check database size
docker compose exec actalog ls -lh /app/data/actalog.db

# Vacuum (in container)
docker compose exec actalog sqlite3 /app/data/actalog.db "VACUUM;"
```

External Database (Not in Docker)

To use an existing external database:

PostgreSQL External

```
# docker-compose.external-postgres.yml
services:
  actalog:
    image: ghcr.io/yourusername/actalog:latest
    environment:
      - DB_DRIVER=postgres
      - DB_HOST=192.168.1.100  # External host
      - DB_PORT=5432
      - DB_NAME=actalog
      - DB_USER=actalog
      - DB_PASSWORD=${DB_PASSWORD}
      - DB_SSLMODE=require     # For production
```

MariaDB External

```
# docker-compose.external-mariadb.yml
services:
  actalog:
    image: ghcr.io/yourusername/actalog:latest
    environment:
      - DB_DRIVER=mysql
      - DB_HOST=192.168.1.101  # External host
      - DB_PORT=3306
      - DB_NAME=actalog
      - DB_USER=actalog
      - DB_PASSWORD=${DB_PASSWORD}
```

Troubleshooting

"Connection refused" errors

PostgreSQL:

```
# Check if PostgreSQL is ready
docker compose -f docker-compose.postgres.yml exec postgres pg_isready

# Check logs
docker compose -f docker-compose.postgres.yml logs postgres
```

MariaDB:

```
# Check if MariaDB is ready
docker compose -f docker-compose.mariadb.yml exec mariadb \
  healthcheck.sh --connect

# Check logs
docker compose -f docker-compose.mariadb.yml logs mariadb
```

"database locked" with SQLite

This happens with high concurrency. Consider migrating to PostgreSQL:

```
docker compose -f docker-compose.postgres.yml up -d
```

Lost database password

PostgreSQL:

```
# Reset password
docker compose -f docker-compose.postgres.yml exec postgres \
  psql -U actalog -c "ALTER USER actalog PASSWORD 'new_password';"
```

MariaDB:

```
# Reset password
docker compose -f docker-compose.mariadb.yml exec mariadb \
  mysql -u root -p -e "SET PASSWORD FOR 'actalog'@'%' =
PASSWORD('new_password');"
```

Performance Tuning

PostgreSQL

Add to compose file under `postgres.command`:

```
command:
  - postgres
  - -c
  - max_connections=200
  - -c
  - shared_buffers=256MB
  - -c
  - effective_cache_size=1GB
```

MariaDB

Add to compose file under `mariadb.command`:

```
command:
  - --max-connections=200
  - --innodb-buffer-pool-size=256M
  - --innodb-log-file-size=64M
```

Recommendation by Use Case

Use Case	Database	Reason
Personal use	SQLite	Simple, zero-config
Small team (< 10)	SQLite or PostgreSQL	Low overhead
Production team	PostgreSQL	Best concurrency
High traffic	PostgreSQL	Proven at scale
Existing MySQL infra	MariaDB	Easy integration
Edge/embedded	SQLite	Single file, portable
Multi-region	PostgreSQL	Replication support

Seed Data Import

ActaLog includes comprehensive seed data:

- **182 movements** (all CrossFit movements including Girl/Hero WOD movements)
- **314 WODs** (all benchmark Girl and Hero WODs)

Automatic Import (Recommended)

To automatically import seed data on first deployment:

1. **Set admin credentials in `.env`** before starting:

   ```
   ADMIN_EMAIL=admin@example.com
   ADMIN_PASSWORD=YourSecurePassword123
   ```

2. **Start the application:**

   ```
   docker compose up -d
   ```

3. **Register your first user** with the same email/password from `.env`

4. **Seeds import automatically** on first startup if credentials are set

The import script:

- Runs only once (creates marker file `/app/data/.seeds_imported`)
- Skips if admin credentials are not set
- Logs progress to container logs: `docker compose logs -f`

Manual Import (Alternative)

If you prefer to import manually or didn't set credentials:

1. **Via Web UI** (easiest):
   - Login as admin

- Navigate to Import page
- Upload `/app/seeds/movements.csv`
- Upload `/app/seeds/wods.csv`

2. **Via API**:

```
# Get admin token
TOKEN=$(curl -s -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@example.com","password":"YourPassword"}' \
  | jq -r '.access_token')

# Import movements
docker compose cp actalog:/app/seeds/movements.csv ./movements.csv
curl -X POST http://localhost:8080/api/import/movements/confirm \
  -H "Authorization: Bearer $TOKEN" \
  -F "file=@movements.csv" \
  -F "skip_duplicates=true"

# Import WODs
docker compose cp actalog:/app/seeds/wods.csv ./wods.csv
curl -X POST http://localhost:8080/api/import/wods/confirm \
  -H "Authorization: Bearer $TOKEN" \
  -F "file=@wods.csv" \
  -F "skip_duplicates=true"
```

Checking Import Status

```
# Check if seeds were imported
docker compose exec actalog ls -la /app/data/.seeds_imported

# View import logs
docker compose logs actalog | grep -i seed
```

Quick Commands

```
# SQLite deployment
docker compose up -d

# PostgreSQL deployment
docker compose -f docker-compose.postgres.yml up -d

# MariaDB deployment
docker compose -f docker-compose.mariadb.yml up -d

# Stop any deployment
docker compose down  # or add -f <file>

# View logs
docker compose logs -f

# Backup database
docker compose exec actalog wget -O- http://localhost:8080/api/admin/backups/
1/download
```