RBarros Docker Deployment

This Docker Compose setup provides a complete containerized environment for the RBarros application, including:

- **Frontend**: Vue.js application served by Nginx
- **Backend**: Node.js/Express API server
- **Database**: MySQL 8.0 database
- **Reverse Proxy**: Nginx for load balancing and SSL termination

Prerequisites

- Docker Engine 20.10+
- Docker Compose 2.0+
- At least 4GB of available RAM
- Ports 80, 3000, 3306, and 8080 available on your host machine

Docker Compose Configuration

This project uses a **clean 3-file setup**:

`docker-compose.base.yml` (Base Configuration)

- **Shared configuration** for all environments
- Defines services without environment-specific settings
- No port mappings (defined in override files)

`docker-compose.prod.yml` (Production Overrides)

- **Production-specific settings**
- Port mapping: `8080:80` (Nginx serving static files)
- Used with: `make prod`

`docker-compose.dev.yml` (Development Overrides)

- **Development-specific settings**
- Port mapping: `8080:8080` (Vue CLI dev server)
- Volume mounts for live code editing and hot reloading
- Used with: `make dev`

Quick Start

Production Mode (Default)

1. **Setup environment**:

```
# Copy the environment template
cp env.example .env

# Edit the .env file with your actual values
nano .env
```

2. **Start production services**:

```
# Using Makefile (recommended)
make prod

# Or directly with Docker Compose
docker-compose up -d
```

Development Mode

1. **Setup environment** (same as above)

2. **Start development services**:

```
# Using Makefile (recommended)
make dev

# Or directly with Docker Compose
docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d
```

3. **Check service status**:

```
docker-compose ps
```

Service URLs

Production Mode

- **Application**: http://localhost (via Nginx reverse proxy)
- **Frontend**: http://localhost:8080
- **Backend API**: http://localhost:3000
- **Database**: localhost:3306

Development Mode

- **Application**: http://localhost (via Nginx reverse proxy)
- **Frontend**: http://localhost:8080 (with hot reload)

- **Backend API**: http://localhost:3000 (with hot reload)
- **Database**: localhost:3306

Environment Configuration

Required Environment Variables

Copy `env.example` to `.env` and configure the following:

Database Settings

```
DB_HOST=database
DB_USER=rbarros_user
DB_PASSWORD=your-secure-password
DB_NAME=rbarros_db
MYSQL_ROOT_PASSWORD=your-root-password
```

Application Secrets

```
SECRET_KEY=your-jwt-secret-key-minimum-32-characters
SECRET_KEY_REFRESH_TOKEN=your-refresh-token-secret
```

External Services

```
SENDGRID_API_KEY=your-sendgrid-api-key
WEBHOOK_SECRET=your-webhook-secret
VUE_APP_API_URL=http://localhost:3000
```

Note: Production configuration requires all environment variables to be set. Development mode provides sensible defaults for most values.

Database Setup

Initial Database Setup

1. **Create database initialization scripts** (optional):

   ```
   mkdir -p rbarros-backend/database/init
   ```

2. **Add SQL initialization files** to `rbarros-backend/database/init/` directory:
   - Files will be executed in alphabetical order

- Use `.sql` or `.sh` extensions

Database Access

```
# Connect to MySQL container
docker-compose exec database mysql -u rbarros_user -p rbarros_db

# Or using root
docker-compose exec database mysql -u root -p
```

Development vs Production

Development Mode

For development with hot reloading:

```
# Override the docker-compose for development
docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d
```

Production Mode

The default configuration is optimized for production:

- Multi-stage builds for smaller images
- Non-root users for security
- Health checks for reliability
- Nginx reverse proxy with rate limiting

Common Commands

Service Management

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

# Stop all services
docker-compose down

# Restart a specific service
docker-compose restart backend

# Rebuild and restart
docker-compose up -d --build
```

Logs and Debugging

```
# View logs
docker-compose logs -f [service-name]

# Execute commands in containers
docker-compose exec backend sh
docker-compose exec frontend sh
docker-compose exec database bash
```

Database Operations

```
# Backup database
docker-compose exec database mysqldump -u root -p rbarros_db > backup.sql

# Restore database
docker-compose exec -T database mysql -u root -p rbarros_db < backup.sql
```

Scaling

Scale specific services:

```
# Scale backend to 3 instances
docker-compose up -d --scale backend=3

# Scale with load balancer update
docker-compose up -d --scale backend=3 --scale frontend=2
```

SSL/HTTPS Setup

1. **Generate SSL certificates**:

   ```
   mkdir -p nginx/ssl
   # Add your SSL certificates to nginx/ssl/
   ```

2. **Update nginx configuration** for HTTPS in `nginx/nginx.conf`

3. **Restart nginx**:

   ```
   docker-compose restart nginx
   ```

Monitoring and Health Checks

Health Check Endpoints

- Backend: http://localhost:3000/health
- Frontend: http://localhost:8080 (nginx status)

Container Health Status

```
docker-compose ps
```

Troubleshooting

Common Issues

1. **Port conflicts**:

```
# Check what's using the ports
netstat -tulpn | grep :80
netstat -tulpn | grep :3000
```

2. **Database connection issues**:

```
# Check database logs
docker-compose logs database

# Verify database is ready
docker-compose exec database mysqladmin ping -h localhost
```

3. **Frontend build issues**:

```
# Rebuild frontend
docker-compose build --no-cache frontend
```

4. **Backend API issues**:

```
# Check backend logs
docker-compose logs backend

# Test backend health
curl http://localhost:3000/health
```

Reset Everything

```
# Stop and remove all containers, networks, and volumes
docker-compose down -v --remove-orphans

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

# Start fresh
docker-compose up -d --build
```

Security Considerations

1. **Change default passwords** in production
2. **Use strong JWT secrets** (minimum 32 characters)
3. **Configure firewall** to restrict database access
4. **Enable SSL/HTTPS** for production
5. **Regular security updates** for base images
6. **Monitor logs** for suspicious activity

Performance Optimization

1. **Resource limits** in docker-compose.yml:

```
deploy:
  resources:
    limits:
      memory: 512M
      cpus: '0.5'
```

2. **Database optimization**:
   - Configure MySQL settings in `database/my.cnf`
   - Set up proper indexes
   - Regular database maintenance
3. **Nginx caching**:
   - Configure static asset caching
   - Enable gzip compression
   - Set up proxy caching

Backup Strategy

1. **Database backups**:

```
# Automated backup script
docker-compose exec database mysqldump -u root -p rbarros_db | gzip >
backup-$(date +%Y%m%d).sql.gz
```

2. **Volume backups**:

```
# Backup persistent volumes
docker run --rm -v rbarros-deployment_mysql_data:/data -v $(pwd):/backup
alpine tar czf /backup/mysql_data_backup.tar.gz -C /data .
```

Support

For issues and questions:

1. Check the logs: `docker-compose logs`
2. Verify service health: `docker-compose ps`
3. Review this documentation
4. Check the individual service README files