Docker Deployment Guide

This guide provides comprehensive instructions for deploying the Classroom Participation Tracker using Docker and Docker Compose.

🐳 Prerequisites

Required Software

- **Docker**: Version 20.0+ (Install Docker (https://docs.docker.com/get-docker/))
- **Docker Compose**: Version 2.0+ (Install Docker Compose (https://docs.docker.com/compose/install/))
- **Git**: For cloning the repository

System Requirements

- **Memory**: Minimum 2GB RAM (4GB recommended)
- **Storage**: 1GB free disk space
- **Network**: Ports 3000 (app), 5432 (database), 6379 (Redis) available

🚀 Quick Start

1. Clone Repository

```
git clone https://github.com/yourusername/classroom-participation-tracker.git
cd classroom-participation-tracker
```

2. Setup Docker Environment

```
# Run the setup script
chmod +x docker-scripts/setup.sh
./docker-scripts/setup.sh
```

3. Configure Environment

```
# Edit environment variables
cp .env.docker .env.docker.local
nano .env.docker.local  # Update with your settings
```

4. Deploy with Docker Compose

```
# Build and start all services
docker-compose up --build -d

# View logs
docker-compose logs -f

# Check service status
docker-compose ps
```

5. Access Application

- **Application**: http://localhost:3000
- **Health Check**: http://localhost:3000/api/health
- **Database**: localhost:5432 (PostgreSQL)
- **Redis**: localhost:6379 (optional)

📁 Docker File Structure

```
classroom-participation-tracker/
├── Dockerfile                    # Production app container
├── Dockerfile.dev                # Development app container
├── docker-compose.yml            # Production services
├── docker-compose.dev.yml        # Development services
├── docker-entrypoint.sh          # App startup script
├── .dockerignore                 # Docker ignore rules
├── .env.docker                   # Environment template
├── .env.docker.local             # Local environment (create this)
├── database/
│   └── init/
│       └── 01-init.sql           # Database initialization
└── docker-scripts/
    ├── setup.sh                  # Initial setup
    ├── backup.sh                 # Database backup
    ├── restore.sh                # Database restore
    └── wait-for-db.sh            # Database health check
```

⚙️ Environment Configuration

Required Environment Variables

Variable	Description	Default	Required
`DATABASE_URL`	PostgreSQL connection string	Generated	Yes
`NEXTAUTH_URL`	Application base URL	http://localhost:3000	Yes
`NEXTAUTH_SECRET`	Authentication secret key	Generated	Yes
`POSTGRES_DB`	Database name	participation_tracker	Yes
`POSTGRES_USER`	Database user	tracker_user	Yes
`POSTGRES_PASSWORD`	Database password	Generated	Yes

Optional Environment Variables

Variable	Description	Default
`NODE_ENV`	Application environment	production
`PORT`	Application port	3000
`REDIS_URL`	Redis connection string	redis://redis:6379
`AWS_BUCKET_NAME`	S3 bucket for file uploads	-
`AWS_FOLDER_PREFIX`	S3 folder prefix	participation-tracker/

Sample .env.docker.local

```
# Database Configuration
DATABASE_URL=postgresql://tracker_user:your_secure_password@database:5432/participa-
tion_tracker?connect_timeout=15

# Authentication Configuration
NEXTAUTH_URL=http://localhost:3000
NEXTAUTH_SECRET=your-very-secure-secret-key-here

# PostgreSQL Settings
POSTGRES_DB=participation_tracker
POSTGRES_USER=tracker_user
POSTGRES_PASSWORD=your_secure_password

# Optional: Redis
REDIS_URL=redis://redis:6379

# Application
NODE_ENV=production
PORT=3000
```

🔧 Docker Commands

Basic Operations

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

# Stop services
docker-compose down

# Restart services
docker-compose restart

# View logs
docker-compose logs -f [service_name]

# Check service status
docker-compose ps
```

Development Mode

```
# Start development environment with hot reload
docker-compose -f docker-compose.dev.yml up --build

# Access development app at http://localhost:3001
```

Database Operations

```
# Access PostgreSQL shell
docker-compose exec database psql -U tracker_user -d participation_tracker

# Create database backup
./docker-scripts/backup.sh

# Restore from backup
./docker-scripts/restore.sh TIMESTAMP

# Reset database
docker-compose exec database psql -U tracker_user -d participation_tracker -c "DROP
SCHEMA public CASCADE; CREATE SCHEMA public;"
```

Application Management

```
# Rebuild application only
docker-compose build app
docker-compose up -d app

# Run Prisma commands
docker-compose exec app npx prisma db push
docker-compose exec app npx prisma db seed
docker-compose exec app npx prisma studio

# Access application shell
docker-compose exec app sh
```

📊 Service Architecture

Services Overview

1. Application Service (`app`)

- **Image**: Custom Next.js build
- **Port**: 3000
- **Features**: Standalone Next.js application with Prisma ORM
- **Health Check**: GET /api/health
- **Dependencies**: Database service

2. Database Service (`database`)

- **Image**: postgres:15-alpine
- **Port**: 5432
- **Features**: PostgreSQL with automatic initialization
- **Health Check**: pg_isready command
- **Persistence**: Named volume (postgres_data)

3. Redis Service (`redis`) - Optional

- **Image**: redis:7-alpine
- **Port**: 6379
- **Features**: Session storage and caching
- **Health Check**: Redis ping command
- **Persistence**: Named volume (redis_data)

Docker Networks

- **participation_network**: Bridge network for service communication

Docker Volumes

- **postgres_data**: PostgreSQL data persistence
- **redis_data**: Redis data persistence
- **uploads_data**: Application file uploads

🔒 Security Considerations

Production Security

1. **Environment Variables**: Use strong passwords and secure secrets
2. **Network Security**: Configure firewall rules for exposed ports
3. **Database Security**: Use non-default credentials and enable SSL
4. **Container Security**: Regular image updates and vulnerability scanning

Security Best Practices

```
# Generate secure secrets
openssl rand -base64 32  # For NEXTAUTH_SECRET
openssl rand -base64 16  # For database password

# Use Docker secrets in production
docker secret create db_password password.txt
```

📈 Performance Optimization

Production Optimizations

1. **Resource Limits**: Set memory and CPU limits in docker-compose.yml
2. **Caching**: Enable Redis for session storage
3. **Database**: Configure PostgreSQL for production workloads
4. **Monitoring**: Use Docker health checks and monitoring tools

Example Resource Configuration

```yaml
services:
  app:
    deploy:
      resources:
        limits:
          cpus: '2.0'
          memory: 1G
        reservations:
          cpus: '1.0'
          memory: 512M
```

🔍 Monitoring and Logging

Health Monitoring

```bash
# Check all service health
docker-compose ps

# Application health check
curl http://localhost:3000/api/health

# Database health check
docker-compose exec database pg_isready -U tracker_user
```

Log Management

```bash
# View application logs
docker-compose logs -f app

# View database logs
docker-compose logs -f database

# View all logs
docker-compose logs -f

# Log rotation (production)
docker-compose logs --tail=100 -f
```

🗃️ Backup and Restore

Automated Backup

```bash
# Create full backup
./docker-scripts/backup.sh

# Schedule daily backups (crontab)
0 2 * * * /path/to/classroom-participation-tracker/docker-scripts/backup.sh
```

Manual Backup

```
# Database only
docker-compose exec database pg_dump -U tracker_user participation_tracker > backup.sq
l

# Files and configuration
tar -czf backup.tar.gz uploads/ .env.docker.local
```

Restore Process

```
# Restore from automated backup
./docker-scripts/restore.sh 20240919_143000

# Manual restore
docker-compose exec -T database psql -U tracker_user -d participation_tracker < backup
.sql
```

🚨 Troubleshooting

Common Issues

Application Won't Start

```
# Check logs
docker-compose logs app

# Verify database connection
docker-compose exec app npx prisma db push

# Restart services
docker-compose restart
```

Database Connection Issues

```
# Check database status
docker-compose exec database pg_isready -U tracker_user

# Verify connection string
docker-compose exec app env | grep DATABASE_URL

# Test connection
docker-compose exec app npx prisma db push
```

Port Conflicts

```
# Check port usage
netstat -tulpn | grep :3000

# Use different ports
docker-compose -f docker-compose.yml up -d --force-recreate
```

Permission Issues

```
# Fix script permissions
chmod +x docker-entrypoint.sh
chmod +x docker-scripts/*.sh

# Fix volume permissions
docker-compose exec app chown -R nextjs:nodejs /app
```

Debug Commands

```
# Interactive shell in app container
docker-compose exec app sh

# Check environment variables
docker-compose exec app env

# Test Prisma connection
docker-compose exec app npx prisma studio

# Check network connectivity
docker network ls
docker network inspect classroom-participation-tracker_participation_network
```

🔄 Updates and Maintenance

Application Updates

```
# Pull latest changes
git pull origin main

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

# Run database migrations
docker-compose exec app npx prisma db push
```

System Maintenance

```
# Clean unused Docker resources
docker system prune -f

# Update Docker images
docker-compose pull
docker-compose up -d

# Monitor disk usage
docker system df
```

Version Management

```
# Tag current deployment
docker tag classroom-participation-tracker_app:latest classroom-participation-track-
er_app:v2.3.1

# Rollback if needed
docker-compose down
docker tag classroom-participation-tracker_app:v2.3.0 classroom-participation-track-
er_app:latest
docker-compose up -d
```

📞 Support

Getting Help

1. **Check Logs**: Always start with `docker-compose logs -f`
2. **Health Checks**: Verify service status with health endpoints
3. **Documentation**: Reference this guide and the main README.md
4. **Community**: Submit issues on GitHub with full log output

Useful Links

- Docker Documentation (https://docs.docker.com/)
- Docker Compose Reference (https://docs.docker.com/compose/)
- Next.js Docker Guide (https://nextjs.org/docs/deployment#docker-image)
- PostgreSQL Docker Guide (https://hub.docker.com/_/postgres)

Version: 2.3.1 (Docker)
Last Updated: September 2024
Docker Compose: v2.0+
Supported Platforms: Linux, macOS, Windows (WSL2)