# Oracle Cloud Coolify Deployment Guide for n8n Collaboration Backend

This comprehensive guide covers deploying the n8n collaboration backend service on Oracle Cloud Infrastructure (OCI) using Coolify, a self-hosted deployment platform.

## Table of Contents



## Prerequisites

- Oracle Cloud Infrastructure account with active compute instance
- Coolify installed and running on your OCI instance
- Domain name (optional but recommended for production)
- GitHub account for version control
- Basic knowledge of Docker and Node.js

## GitHub Repository Setup

### 1. Initialize Git Repository

```
cd /path/to/n8n-collaboration-backend
git init
git add .
git commit -m "Initial commit: n8n collaboration backend"
```

### 2. Create GitHub Repository

1. Create a new repository on GitHub named `n8n-collaboration-backend`
2. Add the remote origin:

```
git remote add origin https://github.com/yourusername/n8n-collaboration-backend.git
git branch -M main
git push -u origin main
```

### 3. Repository Structure Best Practices

Ensure your repository follows this structure:

```
n8n-collaboration-backend/
├── .github/
│   └── workflows/
│       ├── ci.yml
│       └── security.yml
├── docs/
│   └── ORACLE_CLOUD_COOLIFY_DEPLOYMENT.md
├── scripts/
│   ├── backup.sh
│   └── maintenance.sh
├── server.js
├── debug-server.js
├── package.json
├── package-lock.json
├── Dockerfile
├── docker-compose.yml
├── docker-compose.production.yml
├── .env.example
├── .gitignore
├── .dockerignore
├── LICENSE
└── README.md
```

## Project Structure

### Essential Files for Deployment

The repository includes all necessary files for deployment:

**`.gitignore`**

Excludes development files, logs, and sensitive data from version control.

**`.env.example`**

Template for environment variables with all required configuration options.

**`Dockerfile`**

Production-ready Docker configuration with security best practices.

**`docker-compose.yml`** & **`docker-compose.production.yml`**

Development and production Docker Compose configurations.

## Coolify Configuration Files

### 1. Dockerfile

The included Dockerfile provides:
- Multi-stage build optimization

- Non-root user for security
- Health checks
- Proper port exposure
- Log directory creation

### 2. docker-compose.yml (Development)

Development configuration with:
- Hot reload support
- Debug logging
- Local volume mounts
- Development environment variables

### 3. docker-compose.production.yml (Coolify)

Production configuration featuring:
- Environment variable injection
- Health checks
- Log rotation
- Restart policies
- Volume management

## Oracle Cloud Infrastructure Setup

### 1. Network Security Configuration

#### Security List Rules

In your OCI Console, navigate to your VCN's Security List and add these ingress rules:

```
Source CIDR: 0.0.0.0/0
IP Protocol: TCP
Destination Port Range: 80
Description: HTTP traffic

Source CIDR: 0.0.0.0/0
IP Protocol: TCP
Destination Port Range: 443
Description: HTTPS/WSS traffic

Source CIDR: 0.0.0.0/0
IP Protocol: TCP
Destination Port Range: 3000
Description: Application port (if needed)

Source CIDR: 0.0.0.0/0
IP Protocol: TCP
Destination Port Range: 3001
Description: WebSocket port (if needed)
```

### 2. Instance-Level Firewall Configuration

SSH into your Oracle Cloud instance and configure iptables:

```
# Open port 443 for HTTPS/WSS
sudo iptables -I INPUT 6 -m state --state NEW -p tcp --dport 443 -j ACCEPT

# Open port 80 for HTTP (redirects to HTTPS)
sudo iptables -I INPUT 6 -m state --state NEW -p tcp --dport 80 -j ACCEPT

# If using custom ports, open them as well
sudo iptables -I INPUT 6 -m state --state NEW -p tcp --dport 3000 -j ACCEPT
sudo iptables -I INPUT 6 -m state --state NEW -p tcp --dport 3001 -j ACCEPT

# Save the rules
sudo netfilter-persistent save
```

If you encounter issues with the `state` module, use `iptables-legacy`:

```
sudo iptables-legacy -I INPUT -m state --state NEW -p tcp --dport 443 -j ACCEPT
sudo iptables-legacy -I INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT
sudo iptables-legacy-save | sudo tee /etc/iptables/rules.v4
```

### 3. Verify Port Accessibility

Test your ports using an online port scanner or:

```
# Test from another machine
telnet your-server-ip 443
telnet your-server-ip 80
```

## Environment Variables and Secrets Management

### 1. Coolify Environment Variables

Coolify provides three levels of environment variables:

#### Team-Level Variables

Access via `{{team.VARIABLE_NAME}}`:
- `NODE_ENV`
- `LOG_LEVEL`

#### Project-Level Variables

Access via `{{project.VARIABLE_NAME}}`:
- `CORS_ORIGIN`
- `DB_HOST`

#### Environment-Specific Variables

Access via `{{environment.VARIABLE_NAME}}`:
- `JWT_SECRET`
- `DB_PASSWORD`

### 2. Predefined Coolify Variables

Coolify automatically provides:
- `COOLIFY_FQDN`: Your application’s domain
- `COOLIFY_URL`: Full URL of your application

- `COOLIFY_BRANCH` : Git branch being deployed
- `SOURCE_COMMIT` : Git commit hash

### 3. Setting Up Secrets in Coolify

1. Navigate to your resource in Coolify
2. Go to “Environment Variables” section
3. Add the following variables:

```
JWT_SECRET=your_secure_jwt_secret_here
CORS_ORIGIN=https://yourdomain.com
PORT=3000
WS_PORT=3001
HOST=0.0.0.0
WS_HOST=0.0.0.0
NODE_ENV=production
LOG_LEVEL=info
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
WS_PING_TIMEOUT=60000
WS_PING_INTERVAL=25000
WS_MAX_HTTP_BUFFER_SIZE=1000000
```

### 4. Build vs Runtime Variables

Mark these as “Build Variables” if needed during build:
- `NODE_ENV`
- `PORT`

Keep these as runtime variables:
- `JWT_SECRET`
- `CORS_ORIGIN`
- Database credentials

## Domain and SSL Configuration

### 1. Domain Setup in Coolify

1. In your Coolify resource settings, go to “Domains”
2. Add your domain: `api.yourdomain.com`
3. Coolify will automatically handle SSL certificate generation via Let’s Encrypt

### 2. DNS Configuration

Point your domain to your Oracle Cloud instance:

```
Type: A
Name: api (or your subdomain)
Value: YOUR_OCI_INSTANCE_PUBLIC_IP
TTL: 300
```

### 3. SSL Certificate Management

Coolify automatically:
- Generates Let’s Encrypt certificates

- Handles certificate renewal
- Configures Traefik for HTTPS termination

### 4. Custom SSL Configuration (Optional)

If you need custom SSL certificates, you can configure them in Coolify's Traefik settings.

## WebSocket Configuration for Production

### 1. Traefik Configuration for WebSockets

Coolify uses Traefik as a reverse proxy. The application handles WebSocket upgrades properly with the included configuration.

### 2. WebSocket Health Checks

The application includes WebSocket-specific health checks at `/ws-health` endpoint.

### 3. Production WebSocket Configuration

The server is configured with production-ready WebSocket settings including:
- Proper CORS configuration
- Connection timeouts
- Ping/pong intervals
- Buffer size limits

## Monitoring and Logging Setup

### 1. Application Logging

The application uses structured logging with configurable levels and file output.

### 2. Docker Logging Configuration

Production docker-compose includes log rotation and size limits.

### 3. Coolify Monitoring

Coolify provides built-in monitoring:
- Container health checks
- Resource usage monitoring
- Deployment logs
- Application logs

### 4. Custom Monitoring Endpoints

The application includes monitoring endpoints:
- `/health` - Basic health check
- `/status` - Application status
- `/metrics` - Performance metrics
- `/ws-health` - WebSocket status

## CI/CD Pipeline Configuration

### 1. GitHub Actions Workflow

The repository includes comprehensive GitHub Actions workflows:

### CI Pipeline (`ci.yml`)

- Multi-Node.js version testing
- Code linting and formatting
- Security scanning
- Docker image building
- Automated deployment to Coolify

### Security Pipeline (`security.yml`)

- Weekly security scans
- Dependency vulnerability checks
- Docker image security scanning
- CodeQL analysis

## 2. Coolify Webhook Setup

1. In Coolify, go to your resource's "Webhooks" section
2. Copy the webhook URL
3. Add it to GitHub repository secrets as `COOLIFY_WEBHOOK`
4. Generate a Coolify API token and add as `COOLIFY_TOKEN`

## 3. Automatic Deployment Configuration

In Coolify:
1. Enable "Auto Deploy" for your resource
2. Set up GitHub App integration for private repositories
3. Configure webhook events (push, pull request)

## 4. Preview Deployments

For pull request previews:
1. Enable "Preview Deployments" in Coolify
2. Configure preview environment variables
3. Set up preview domain pattern: `pr-{PR_NUMBER}.yourdomain.com`

# Backup and Maintenance Procedures

## 1. Application Data Backup

The repository includes backup scripts in the `scripts/` directory:

- `backup.sh` - Database and application data backup
- `maintenance.sh` - System maintenance tasks

## 2. Automated Backup with Cron

```
# Add to crontab (crontab -e)
# Daily backup at 2 AM
0 2 * * * /path/to/scripts/backup.sh

# Weekly maintenance on Sundays at 3 AM
0 3 * * 0 /path/to/scripts/maintenance.sh
```

### 3. Coolify Backup

Coolify automatically backs up:
- Application configurations
- Environment variables (encrypted)
- Deployment history

## Troubleshooting Common Issues

### 1. Deployment Failures

#### Issue: Build fails with missing files

**Solution:**
- Check `.dockerignore` file
- Ensure all required files are committed to Git
- Verify Dockerfile COPY commands

#### Issue: Port allocation errors

**Solution:**
- Remove port mappings from docker-compose.production.yml
- Let Coolify handle port exposure through Traefik

### 2. WebSocket Connection Issues

#### Issue: WebSocket connections fail

**Solution:**
- Verify CORS_ORIGIN environment variable
- Check SSL certificate status
- Ensure proper domain DNS configuration

#### Issue: SSL/WSS certificate problems

**Solution:**
1. Verify domain DNS points to correct IP
2. Check Coolify SSL certificate status
3. Ensure Let’s Encrypt can reach your domain

### 3. Environment Variable Issues

#### Issue: Environment variables not loading

**Solution:**
1. Check variable names in Coolify UI
2. Verify docker-compose.yml syntax
3. Restart the application after changes

### 4. Oracle Cloud Networking Issues

#### Issue: Cannot access application externally

**Solution:**
- Check security list rules in OCI Console
- Verify iptables rules on the instance
- Test port accessibility

### 5. Performance Issues

#### Issue: High memory usage

**Solution:**
- Monitor application metrics via `/metrics` endpoint
- Check for memory leaks in WebSocket connections
- Review log files for errors

### 6. Logging and Debugging

#### Enable debug logging:

```
# In Coolify environment variables
DEBUG=socket.io*
LOG_LEVEL=debug
```

#### Check application logs:

- Use Coolify's built-in log viewer
- SSH into server and check Docker logs
- Review application log files in `/app/logs`

### 7. Recovery Procedures

#### Application Recovery:

1. Check Coolify deployment logs
2. Verify environment variables
3. Test health check endpoints
4. Restart the application through Coolify

#### Database Recovery (if applicable):

```
# Restore from backup
gunzip db_backup_YYYYMMDD_HHMMSS.sql.gz
psql -h localhost -U postgres n8n_collaboration < db_backup_YYYYMMDD_HHMMSS.sql
```

## Conclusion

This deployment guide provides a comprehensive approach to deploying your n8n collaboration backend service on Oracle Cloud using Coolify. The configuration ensures:

- Secure and scalable deployment
- Proper WebSocket handling for real-time features
- Automated CI/CD pipeline
- Comprehensive monitoring and logging
- Robust backup and recovery procedures

Remember to:
- Test all configurations in a staging environment first
- Monitor application performance and logs regularly
- Keep backups current and test recovery procedures
- Update dependencies and security patches regularly
- Review and update firewall rules as needed

For additional support, refer to:
- Coolify Documentation (https://coolify.io/docs/)
- Oracle Cloud Infrastructure Documentation (https://docs.oracle.com/en-us/iaas/)
- Docker Documentation (https://docs.docker.com/)
- Node.js Best Practices (https://nodejs.org/en/docs/guides/)