MCP DeepWiki Server - Deployment Guide

This guide covers different ways to deploy and use the MCP DeepWiki Server.

Quick Start

1. Build and Test

```
# Build the project
npx tsc

# Run tests
./test.sh
```

2. Start the Server

```
# STDIO mode (for local editors)
./start.sh stdio

# HTTP mode (for remote access)
./start.sh http 4000
```

Local Development Setup

Prerequisites

- Node.js >= 18.0.0
- npm or yarn
- Git (optional)

Installation

```
# Clone or download the project
git clone <repository-url>
cd mcp-deepwiki-server

# Install dependencies
npm install

# Build the project
npx tsc

# Test everything works
./test.sh
```

Integration with AI Clients

Claude Desktop

1. **Install the server** (choose one):
   ```bash
   # Option A: Local installation
   git clone && cd mcp-deepwiki-server && npm install && npx tsc

# Option B: Global installation (if published to npm)
npm install -g mcp-deepwiki-server
```

1. **Configure Claude Desktop**:

Edit your Claude Desktop configuration file:
- **macOS**: `~/Library/Application Support/Claude/claude_desktop_config.json`
- **Windows**: `%APPDATA%\Claude\claude_desktop_config.json`

```
json
  {
    "mcpServers": {
      "deepwiki": {
        "command": "node",
        "args": ["/path/to/mcp-deepwiki-server/dist/index.js", "--stdio"],
        "env": {
          "LOG_LEVEL": "info"
        }
      }
    }
  }
```

1. **Restart Claude Desktop** and verify the server appears in the MCP section.

Cursor IDE

1. **Create MCP configuration**:

Create `.cursor/mcp.json` in your project root:

```
json
  {
    "mcpServers": {
      "deepwiki": {
        "command": "node",
        "args": ["/path/to/mcp-deepwiki-server/dist/index.js", "--stdio"],
        "description": "DeepWiki documentation fetcher"
      }
    }
  }
```

1. **Restart Cursor** and the server will be available for AI assistance.

VS Code GitHub Copilot

Add to your VS Code settings or workspace configuration:

```
{
  "github.copilot.mcp.servers": {
    "deepwiki": {
      "command": "node",
      "args": ["/path/to/mcp-deepwiki-server/dist/index.js", "--stdio"]
    }
  }
}
```

Remote Deployment

Docker Deployment

1. **Build the Docker image**:
   `bash`
   `docker build -t mcp-deepwiki-server .`

2. **Run with Docker**:
   ```bash
   # HTTP mode
   docker run -p 4000:4000 -e PORT=4000 mcp-deepwiki-server

# STDIO mode (for testing)
docker run -it –rm mcp-deepwiki-server node dist/index.js –stdio
```

1. **Using Docker Compose**:
   ```bash
   # Use the provided docker-compose.yml
   docker-compose up -d

# Check health
curl http://localhost:4000/health
```

VPS/Cloud Deployment

1. **Prepare the server**:
   `bash`
   `# On your VPS`
   `git clone <repository>`
   `cd mcp-deepwiki-server`
   `npm install`
   `npx tsc`

2. **Create a systemd service** (optional):
   ```ini
   # /etc/systemd/system/mcp-deepwiki.service
   [Unit]
   Description=MCP DeepWiki Server
   After=network.target

[Service]
Type=simple
   ```

User=ubuntu
WorkingDirectory=/path/to/mcp-deepwiki-server
ExecStart=/usr/bin/node dist/index.js
Environment=PORT=4000
Environment=NODE_ENV=production
Restart=always

[Install]
WantedBy=multi-user.target
```

1. **Start the service**:
   `bash`
   `sudo systemctl enable mcp-deepwiki`
   `sudo systemctl start mcp-deepwiki`
   `sudo systemctl status mcp-deepwiki`

### Nginx Reverse Proxy (Production)

```
server {
    listen 80;
    server_name your-domain.com;

    location /mcp {
        proxy_pass http://localhost:4000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;

        # CORS headers for browser clients
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Methods 'GET, POST, DELETE, OPTIONS';
        add_header Access-Control-Allow-Headers 'Content-Type, mcp-session-id';
        add_header Access-Control-Expose-Headers 'Mcp-Session-Id';
    }

    location /health {
        proxy_pass http://localhost:4000;
    }
}
```

Environment Configuration

Environment Variables

- `PORT`: HTTP port (default: 4000)
- `NODE_ENV`: Environment (development/production)
- `LOG_LEVEL`: Logging level (debug/info/warn/error)
- `DEBUG`: Enable debug logging (true/false)
```

Example Configurations

Development:

```
export NODE_ENV=development
export LOG_LEVEL=debug
export DEBUG=true
./start.sh stdio
```

Production:

```
export NODE_ENV=production
export LOG_LEVEL=info
export PORT=4000
./start.sh http
```

Monitoring and Troubleshooting

Health Checks

```
# Basic health check
curl http://localhost:4000/health

# Expected response:
# {"status":"ok","server":"deepwiki-mcp-server","version":"1.0.0"}
```

Logs

```
# View logs in real-time
tail -f /var/log/mcp-deepwiki.log

# Or with systemd
journalctl -u mcp-deepwiki -f
```

Common Issues

1. **“Command not found” errors**:
   - Ensure the path in MCP configuration is correct
   - Check file permissions (`chmod +x dist/index.js`)

2. **Network timeouts**:
   - Check internet connectivity to deepwiki.com
   - Verify firewall settings for outbound HTTPS

3. **Permission denied**:
   - Run with appropriate user permissions
   - Check file ownership and permissions

4. **Port already in use**:
   - Change the PORT environment variable
   - Kill existing processes on the port

Security Considerations

Production Security

1. **Domain Allowlisting**: Only deepwiki.com is allowed (built-in)
2. **Input Validation**: All inputs are validated and sanitized
3. **HTML Sanitization**: Fetched content is cleaned before processing
4. **Rate Limiting**: Built-in safeguards prevent excessive requests

Network Security

```
# Firewall rules (example with ufw)
sudo ufw allow 4000/tcp  # Allow MCP server port
sudo ufw allow 80/tcp    # Allow HTTP (if using nginx)
sudo ufw allow 443/tcp   # Allow HTTPS (if using nginx)
```

HTTPS Setup (Production)

Use Let's Encrypt with nginx for HTTPS:

```
# Install certbot
sudo apt install certbot python3-certbot-nginx

# Get certificate
sudo certbot --nginx -d your-domain.com

# Auto-renewal (already configured by certbot)
sudo crontab -l | grep certbot
```

Performance Tuning

Node.js Optimization

```
# Increase memory limit if needed
node --max-old-space-size=4096 dist/index.js

# Enable cluster mode (for high load)
# Consider using PM2 or similar process manager
```

Caching (Optional)

Consider implementing Redis for caching frequently accessed repository data:

```
// Example caching layer (not included in base implementation)
const redis = require('redis');
const client = redis.createClient();

// Cache repository content for 1 hour
await client.setex(`repo:${owner}/${repo}`, 3600, content);
```

Backup and Maintenance

Regular Maintenance

```
# Update dependencies
npm audit fix

# Rebuild after updates
npx tsc

# Test functionality
./test.sh

# Restart service
sudo systemctl restart mcp-deepwiki
```

Monitoring

Consider setting up monitoring with:
- **Uptime monitoring**: Pingdom, UptimeRobot
- **Log aggregation**: ELK stack, Grafana
- **Performance monitoring**: New Relic, DataDog

Support and Development

Getting Help

1. Check the main README.md for basic setup
2. Review this deployment guide for advanced configurations
3. Check logs for specific error messages
4. Test with provided example repositories first

Contributing

1. Fork the repository
2. Create a feature branch
3. Make changes and add tests
4. Ensure all tests pass
5. Submit a pull request

For additional support, please refer to the main documentation or create an issue in the repository.