Deployment Guide - Cologne Open Data MCP Server

🚀 Deploying to Render

This guide explains how to deploy your Cologne Open Data MCP server to Render for web-based access.

Prerequisites

- GitHub account with the repository pushed
- Render account (free tier available)
- Basic understanding of environment variables

Step 1: Prepare Your Repository

Ensure all files are committed and pushed to GitHub:

```
git add .
git commit -m "Add SSE support for web deployment"
git push origin main
```

Step 2: Create Render Account

1. Go to https://render.com
2. Sign up with your GitHub account
3. Authorize Render to access your repositories

Step 3: Deploy from Dashboard

Option A: Using render.yaml (Recommended)

1. Go to your Render Dashboard
2. Click **"New +"** → **"Blueprint"**
3. Connect your GitHub repository: `ErtanOz/Cologne-Open-Data-Mcp`
4. Render will automatically detect `render.yaml`
5. Click **"Apply"**

Option B: Manual Setup

1. Go to your Render Dashboard
2. Click **"New +"** → **"Web Service"**
3. Connect your GitHub repository
4. Configure:
   - **Name**: `cologne-open-data-mcp`
   - **Runtime**: Node
   - **Region**: Frankfurt (or closest to you)

- **Branch**: main
- **Build Command**: `npm install && npm run build`
- **Start Command**: `npm run start:sse`
- **Plan**: Free

Step 4: Configure Environment Variables

Add these in Render Dashboard → Environment:

Key	Value	Required
`NODE_ENV`	`production`	Yes
`PORT`	`10000`	Yes (auto-set by Render)
`ALLOWED_ORIGINS`	`*`	Yes (or specific domains)
`PARKING_URL`	`https://www.stadt-koeln.de/externe-dienste/open-data/parking.php`	No (has default)
`BAUSTELLEN_WFS`	`https://geoportal.stadt-koeln.de/wss/service/baustellen_wfs/guest?SERVICE=WFS&REQUEST=GetCapabilities`	No (has default)
`RHEINPEGEL_URL`	`https://www.stadt-koeln.de/interne-dienste/hochwasser/pegel_ws.php`	No (has default)
`NEXTBIKE_URL`	`https://api.nextbike.net/maps/nextbike-live.xml?city=14`	No (has default)
`OPARL_BODIES_URL`	`https://buergerinfo.stadt-koeln.de/oparl/bodies`	No (has default)

Step 5: Deploy

1. Click **"Create Web Service"**
2. Wait for deployment (usually 2-5 minutes)
3. Your service will be available at: `https://cologne-open-data-mcp.onrender.com`

Step 6: Verify Deployment

Test your deployed server:

```
# Health check
curl https://your-app.onrender.com/health

# Expected response:
```

```
{
  "status": "healthy",
  "service": "cologne-open-data-mcp",
  "version": "0.1.0",
  "timestamp": "2025-01-29T22:00:00.000Z"
}
```

🔌 Connecting to MCP Clients

For Claude Desktop (Local)

Claude Desktop uses the local STDIO version:

Configuration file location:

- **macOS**: `~/Library/Application Support/Claude/claude_desktop_config.json`
- **Windows**: `%APPDATA%\Claude\claude_desktop_config.json`

Add this configuration:

```
{
  "mcpServers": {
    "cologne-data": {
      "command": "npx",
      "args": ["cologne-open-data-mcp"]
    }
  }
}
```

For Web-Based MCP Clients

Use the SSE endpoint:

Endpoint: `https://your-app.onrender.com/sse`

Example connection code:

```
import { Client } from '@modelcontextprotocol/sdk/client/index.js';
import { SSEClientTransport } from '@modelcontextprotocol/sdk/client/sse.js';

const transport = new SSEClientTransport(
  new URL('https://your-app.onrender.com/sse')
);

const client = new Client({
  name: 'my-client',
  version: '1.0.0'
}, {
  capabilities: {}
});

await client.connect(transport);
```

🤖 Important Note About ChatGPT

ChatGPT does NOT support MCP (Model Context Protocol).

If you want to use these Cologne Open Data APIs with ChatGPT, you have two options:

Option 1: Use Claude Desktop Instead

Claude Desktop fully supports MCP servers. Simply:

1. Install Claude Desktop
2. Configure as shown above
3. Access all Cologne data through Claude

Option 2: Create a Custom GPT (Requires API Wrapper)

To use with ChatGPT Custom GPTs, you'd need to create a REST API wrapper. This is a separate project that would:

1. Expose REST endpoints instead of MCP
2. Use OpenAPI specification
3. Be configured in ChatGPT's Custom GPT Actions

This MCP server is optimized for Claude and other MCP-compatible clients.

📊 Monitoring Your Deployment

Render Dashboard

- **Logs**: Real-time server logs
- **Metrics**: CPU, Memory usage
- **Events**: Deployment history

Health Check Endpoint

Monitor uptime:

```
curl https://your-app.onrender.com/health
```

🔧 Troubleshooting

Server Not Starting

Check logs in Render dashboard for:

- Missing environment variables
- Build errors
- Port binding issues

CORS Errors

Update `ALLOWED_ORIGINS` environment variable:

```
# Allow specific domains
ALLOWED_ORIGINS=https://example.com,https://app.example.com

# Allow all (development only)
ALLOWED_ORIGINS=*
```

Timeout Issues

Render free tier sleeps after 15 minutes of inactivity:

- First request may take 30-60 seconds to wake up
- Consider upgrading to paid tier for always-on service

📈 Scaling

Free Tier Limitations

- 512 MB RAM
- Shared CPU
- Sleeps after 15 min inactivity
- 750 hours/month

Upgrading

For production use, consider:

- **Starter Plan** ($7/month): Always-on, more resources
- **Standard Plan** ($25/month): Autoscaling, better performance

🔐 Security Best Practices

1. **Restrict CORS**: Set specific allowed origins
2. **Use HTTPS**: Render provides free SSL
3. **Monitor Logs**: Check for unusual activity
4. **Rate Limiting**: Consider adding rate limiting for production

📚 Additional Resources

- Render Documentation
- MCP Specification
- Claude Desktop
- Project Repository

🆘 Support

For issues:

- Check GitHub Issues
- Review Render logs
- Verify environment variables