MCP Manager - README

Contents

1 MCP Manager

Enterprise-grade MCP (Model Context Protocol) server management tool with modern TUI and CLI interfaces.

1.1 Overview

MCP Manager is a professional tool for managing MCP servers used by Claude Code. It provides both a beautiful terminal user interface (TUI) and comprehensive command-line interface (CLI) for discovering, installing, configuring, and managing MCP servers across different scopes.

1.2 Features

1.2.1 Core Functionality

- **One-Command Installation**: `mcp-manager install-package dd-SQLite` for instant setup
- **Smart Discovery**: Find MCP servers from NPM, Docker Hub, and Docker Desktop catalogs
- **Unique Server IDs**: No more confusion with servers having the same name
- **Duplicate Detection**: Automatic warnings when installing similar functionality
- **Docker Desktop Integration**: Seamless integration with Docker Desktop MCP servers
- **Configuration Cleanup**: Fix broken MCP configurations with backup safety
- **Scope Support**: Local (private), Project (shared), User (global) configurations

1.2.2 User Interfaces

- **Modern TUI**: Beautiful terminal interface built with Textual
- **Comprehensive CLI**: Full command-line interface with rich help
- **Interactive Menus**: Intuitive navigation and selection
- **Keyboard Shortcuts**: Efficient operation for power users

1.2.3 Enterprise Features

- **Structured Logging**: JSON and text logging with rotation
- **Configuration Management**: TOML-based configuration with validation
- **Dependency Checking**: Automatic validation of system requirements
- **Performance Monitoring**: Built-in profiling and metrics
- **Comprehensive Testing**: Unit and integration test coverage
- **Type Safety**: Full type hints and mypy validation

1.3 How It Works

1.3.1 Easy Installation Process

1. **Discovery**: `mcp-manager discover --query filesystem`
   - Finds MCP servers from NPM registry, Docker Hub, and Docker Desktop
   - Shows unique Install IDs to distinguish servers with same names
   - Displays exact install commands
2. **Installation**: `mcp-manager install-package dd-SQLite`
   - Automatically handles Docker Desktop, NPM, and Docker servers
   - Configures proper command arguments and paths
   - Provides duplicate detection warnings
   - Servers become immediately active in Claude Code
3. **Management**: `mcp-manager list` shows all installed servers
   - Clean unified view across all server types
   - Easy removal with `mcp-manager remove server-name`

1.3.2 Architecture Insights

Claude Code Configuration Hierarchy: 1. **Internal State**: `~/.claude.json` (source of truth managed by `claude mcp` commands) 2. **User Config**: `~/.config/claude-code/mcp-servers.json` 3. **Project Config**: `./.mcp.json`

Docker Desktop Integration: - MCP Manager uses `docker mcp server enable/disable` for Docker Desktop servers - Creates unified `docker-gateway` that aggregates all enabled Docker Desktop MCPs - Automatic synchronization with Claude Code's internal state

1.4 What's New

1.4.1 Major User Experience Improvements

One-Command Installation - `mcp-manager install-package dd-SQLite` - No more complex manual commands! - Unique Install IDs solve the "multiple servers with same name" problem - Discovery output shows exact install commands for copy/paste convenience

Smart Duplicate Detection - Automatically warns when installing servers with similar functionality - Prevents conflicts between filesystem, database, or browser automation servers - Cross-source detection (NPM vs Docker vs Docker Desktop)

Configuration Cleanup - `mcp-manager cleanup` fixes broken MCP configurations automatically - Creates safety backups before making changes - Removes problematic Docker commands that cause ENOENT errors

Enhanced Discovery - Real-time discovery from NPM registry, Docker Hub, and Docker Desktop catalogs - Improved search with multiple strategies and quality scoring - `--update-catalog` option to refresh Docker Desktop catalog

Cleaner Output - Moved verbose logs to debug mode for cleaner user experience - Clear visual distinction between different server types - Helpful inline guidance and examples

1.5 Quick Start

1.5.1 Installation

```
# Clone the repository
git clone https://github.com/anthropics/claude-mcp-manager
cd claude-mcp-manager/mcp-manager-python

# Install with pip (development mode)
pip install -e .

# Or install with optional dependencies
pip install -e ".[dev,test]"
```

1.5.2 Usage

1.5.2.1 Terminal User Interface (TUI)

```
# Launch the modern TUI
mcp-tui

# Or use the main command
mcp-manager tui
```

1.5.2.2 Command Line Interface (CLI) NEW: Easy Installation with install-package

```
# Discover available servers with unique Install IDs
mcp-manager discover --query filesystem

# Install servers using their unique ID (no complex commands needed!)
mcp-manager install-package modelcontextprotocol-filesystem  # Official NPX filesystem
```

```
mcp-manager install-package dd-SQLite                        # Docker Desktop SQLite
mcp-manager install-package mcp-filesystem                   # Docker Hub filesystem

# The discovery output shows the exact install command for each server:
#   Install ID                  Type   Install Command
#   modelcontextprotocol-...    npm    mcp-manager install-package model...
#   dd-SQLite                   dd     mcp-manager install-package dd-SQLite
```

Traditional Commands (still supported)

```
# List all servers
mcp-manager list

# Discover with advanced options
mcp-manager discover --query filesystem --type npm

# Clean up broken configurations
mcp-manager cleanup --dry-run  # Preview changes
mcp-manager cleanup           # Fix with automatic backup

# Manual server management
mcp-manager add filesystem "npx @modelcontextprotocol/server-filesystem" --scope user
mcp-manager remove filesystem

# Get help
mcp-manager --help
```

1.6 Understanding Server Operations

1.6.1 What is the difference technically between enable, disable and remove?

Great question! Let me explain the technical differences between these three operations in the context of MCP Manager and Docker Desktop servers:

1.6.1.1 Enable

- **Purpose**: Make a server available for use by Claude Code
- **Docker Desktop**: Activates the server in Docker Desktop's registry
- **Claude Code**: Adds server to Claude's configuration (directly or via docker-gateway)
- **Docker Image**: May trigger image pull when first accessed
- **Reversible**: Yes, can be disabled

For Docker Desktop servers:

```
docker mcp server enable SQLite        # Enable in Docker Desktop
claude mcp add docker-gateway "..."    # Add/update gateway in Claude
```

1.6.1.2 Disable

- **Purpose**: Stop a server from being available, but keep it installed
- **Docker Desktop**: Deactivates server in Docker Desktop's registry
- **Claude Code**: Removes from active configuration but server definition remains available
- **Docker Image**: Now cleaned up automatically (our fix)
- **Reversible**: Yes, can be re-enabled

For Docker Desktop servers:

```
docker mcp server disable SQLite       # Disable in Docker Desktop
claude mcp remove docker-gateway       # Remove from Claude
claude mcp add docker-gateway "..."    # Re-add gateway without this server
docker rmi mcp/sqlite:latest          # Clean up image (automated)
```

1.6.1.3 Remove

- **Purpose**: Completely uninstall and delete a server
- **Docker Desktop**: Not applicable (DD servers are built-in)
- **Claude Code**: Completely removes server configuration
- **Docker Image**: Should be cleaned up (for custom Docker servers)
- **Reversible**: No, must be re-added/re-installed

For regular MCP servers:

```
claude mcp remove server-name          # Remove from Claude completely
# For Docker-based servers, should also clean up images
```

1.6.1.4 Key Technical Differences

Operation	Server Config	Docker Image	Registry Entry	Reversible
Enable	Added	Pulled on-demand	Active	Yes
Disable	Removed	Cleaned up	Inactive	Yes
Remove	Deleted	Cleaned up	Deleted	No

1.6.1.5 Docker Desktop Specific Behavior Docker Desktop MCP servers are **built-in**, so: - **Enable/Disable**: Toggles availability in Docker Desktop's registry - **Remove**: Not possible - they're part of Docker Desktop itself - **Images**: Pulled from Docker Hub when needed, cleaned up when disabled

1.7 Architecture

1.7.1 Project Structure

```
mcp-manager-python/
  src/mcp_manager/          # Main package
     core/                 # Core business logic
     cli/                  # Command-line interface
     tui/                  # Terminal user interface
     utils/                # Utilities and helpers
  tests/                    # Test suite
  docs/                     # Documentation
  pyproject.toml           # Project configuration
```

1.7.2 Key Components

- **Core Module**: MCP server management, configuration, discovery
- **CLI Module**: Command-line interface using Click
- **TUI Module**: Terminal interface using Textual
- **Utils Module**: Logging, configuration, validation utilities

1.8 Development

1.8.1 Setup Development Environment

```
# Install development dependencies
pip install -e ".[dev]"
```

```
# Install pre-commit hooks
pre-commit install

# Run tests
pytest

# Format code
black src tests
isort src tests

# Type checking
mypy src

# Linting
flake8 src tests
```

1.8.2 Running Tests

```
# Run all tests
pytest

# Run with coverage
pytest --cov=mcp_manager

# Run specific test types
pytest -m unit
pytest -m integration
pytest -m "not slow"
```

1.9 Configuration

MCP Manager uses a hierarchical configuration system:

1. **System Configuration**: `/etc/mcp-manager/config.toml`
2. **User Configuration**: `~/.config/mcp-manager/config.toml`
3. **Project Configuration**: `./.mcp-manager.toml`
4. **Environment Variables**: `MCP_MANAGER_*`

1.9.1 Example Configuration

```
[logging]
level = "INFO"
format = "json"
file = "~/.config/mcp-manager/logs/app.log"

[claude]
cli_path = "claude"
config_path = "~/.config/claude-code/mcp-servers.json"

[discovery]
npm_registry = "https://registry.npmjs.org"
docker_registry = "docker.io"
cache_ttl = 3600
```

```
[ui]
theme = "dark"
animations = true
confirm_destructive = true
```

1.10 Scope Management

MCP Manager supports three configuration scopes:

1.10.1 Local Scope

- **Purpose**: Private to your user account
- **Storage**: User-specific configuration
- **Use Case**: Personal tools, experimental servers

1.10.2 Project Scope

- **Purpose**: Shared with team via git
- **Storage**: `.mcp-manager.toml` in project root
- **Use Case**: Project-specific tools, team environments

1.10.3 User Scope

- **Purpose**: Global user configuration
- **Storage**: `~/.config/mcp-manager/`
- **Use Case**: Common tools, personal preferences

1.11 Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Make your changes following the coding standards
4. Add tests for new functionality
5. Run the test suite (`pytest`)
6. Commit your changes (`git commit -m 'Add amazing feature'`)
7. Push to the branch (`git push origin feature/amazing-feature`)
8. Open a Pull Request

1.11.1 Coding Standards

- Follow PEP 8 style guidelines
- Use type hints for all functions and methods
- Write docstrings for all public APIs
- Maintain test coverage above 90%
- Keep functions under 50 lines when possible
- Keep files under 1000 lines

1.12 License

This project is licensed under the MIT License - see the LICENSE file for details.

1.13 Support

- **Documentation**: Full documentation
- **Issues**: GitHub Issues
- **Discussions**: GitHub Discussions

1.14 Acknowledgments

- Built with Textual for the TUI
- Uses Rich for beautiful terminal output
- Powered by Click for the CLI
- Configuration management with Pydantic