MCP Manager - User Guide

Contents

1 MCP Manager - Comprehensive User Guide

Version: 1.0
Date: July 2025
Target Audience: Developers, DevOps Engineers, AI Engineers

1.1 Table of Contents

1.2 Overview

MCP Manager is a comprehensive tool for managing Model Context Protocol (MCP) servers used by Claude Code. It provides discovery, installation, configuration, and synchronization capabilities across multiple server sources including Docker Desktop, NPM registry, Docker Hub, and custom implementations.

1.2.1 Key Features

- **Multi-Source Discovery**: Find servers from NPM, Docker Hub, Docker Desktop catalogs
- **One-Command Installation**: Install servers with unique install IDs

- **External Change Synchronization**: Automatic detection and sync of configuration changes
- **Multiple Interfaces**: Interactive menu, CLI commands, and TUI options
- **Sync Loop Protection**: Prevents conflicts during background operations
- **Comprehensive Monitoring**: Background service with configurable auto-sync

---

1.3 Installation

1.3.1 Prerequisites

Required: - Python 3.9+ (Python 3.11+ recommended) - Claude Code CLI installed and configured - Operating System: macOS, Linux, or Windows

Optional (for full functionality): - Docker Desktop (for Docker Desktop MCP integration) - Git (for development workflows)

1.3.2 Installation Methods

1.3.2.1 Method 1: PyPI Installation (Future Release)

> **Note**: PyPI distribution will be available in a future release

```
# Install latest stable version (coming soon)
pip install mcp-manager

# Install with all optional dependencies
pip install mcp-manager[all]

# Verify installation
mcp-manager --version
```

1.3.2.2 Method 2: Development Installation (Current)

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

# Install in development mode
pip install -e ".[dev]"

# Verify installation
python -m mcp_manager.cli.main --version
```

1.3.2.3 Method 3: Docker Installation (Future Release)

> **Note**: Docker images will be available in a future release

```
# Pull and run Docker container (coming soon)
docker run -it --rm mcpmanager/mcp-manager:latest

# With volume mounting for persistence
docker run -it --rm -v ~/.claude:/root/.claude mcpmanager/mcp-manager:latest
```

1.3.3 Post-Installation Setup

1. **Verify Claude Code Integration**:

```
# Test Claude Code CLI access
claude mcp list

# If not found, install Claude Code CLI first
# Follow: https://docs.anthropic.com/claude/docs/claude-code
```

2. **Initialize Configuration**:

```
# Create default configuration
mcp-manager config --init

# Verify system setup
mcp-manager system-info
```

1.4 Getting Started

1.4.1 Quick Start (5 Minutes)

1. **Launch Interactive Menu**:

```
mcp-manager
```

Example Screen:

```
                    MCP Manager v1.0


  1.  List Servers          7.  Discover Servers
  2.  Add Server            8.  Install Package
  3.  Remove Server         9.  Sync Changes
  4.  Enable Server         10.  Monitor Changes
  5.  Disable Server        11.   System Info
  6.  Configure Server      12.  Help

  Press number + Enter, or 'q' to quit
```

2. **Discover Available Servers**:

```
# Option 7 from menu, or direct command:
mcp-manager discover --query filesystem
```

Example Output:

```
 Discovering MCP servers...

            Discovery Results (3 servers)

  Install ID          Name              Type           Score  Description

  dd-filesystem       filesystem        docker-desktop  10.0  File ops
  mcp-filesystem      @mcp/filesystem   npm             8.2   File system
  docker-filesystem   mcp-filesystem    docker          6.1   MCP Files
```

```
Use: mcp-manager install-package <install-id>
```

3. **Install a Server**:

```
# Option 8 from menu, or direct command:
mcp-manager install-package dd-filesystem
```

Example Output:

```
Installing server: dd-filesystem

Resolving install ID...
Found: filesystem (Docker Desktop MCP)

Enabling in Docker Desktop...
Server enabled: filesystem

Importing to Claude Code...
Docker gateway imported successfully

Updating server catalog...
Server catalog updated

Installation complete!
 Server 'filesystem' is now available in Claude Code
```

4. **Verify Installation**:

```
# Option 1 from menu, or direct command:
mcp-manager list
```

Example Output:

```
                          MCP Servers

Name              Scope    Status         Type        Command

filesystem         user   enabled   docker-desktop   docker
test-server        user   enabled        npm         npx
```

1.5 Core Concepts

1.5.1 Server Types

Type	Description	Installation Method	Example
Docker Desktop	Official Docker Desktop MCP servers	Enable in DD, import gateway	SQLite, filesystem
NPM	JavaScript/TypeScript packages	NPM registry installation	@mcp/filesystem
Docker	Containerized MCP servers	Docker Hub or custom registry	mcp/server-name
Custom	User-defined commands	Manual command specification	echo, python scripts

1.5.2 Install IDs

Purpose: Unique identifiers to distinguish servers with identical names

Format Examples: - `dd-SQLite` - Docker Desktop SQLite server - `mcp-sqlite` - NPM sqlite package - `docker-sqlite` - Docker Hub sqlite container - `custom-sqlite` - User-defined SQLite implementation

1.5.3 Configuration Scopes

Scope	Location	Purpose	Example
System	`/etc/mcp-manager/`	Organization-wide policies	Corporate server whitelist
User	`~/.config/mcp-manager/`	Personal preferences	Default discovery sources
Project	`./.mcp-manager.toml`	Project-specific settings	Local development servers

1.5.4 External Change Detection

Purpose: Monitor and synchronize changes made by external tools

Sources Monitored: - Changes via `claude mcp` commands - Docker Desktop server enable/disable operations - Manual configuration file edits - Other tools modifying MCP configurations

---

1.6 User Interfaces

1.6.1 1. Interactive Menu (Default)

Launch: `mcp-manager` (no arguments)

Features: - Numbered menu options for all operations - Real-time server status display - Progress indicators for long operations - Contextual help and error messages

Navigation: - Enter number to select option - 'q' or 'quit' to exit - 'h' or 'help' for assistance

1.6.2 2. Command Line Interface (CLI)

Launch: `mcp-manager <command> [options]`

Example Commands:

```
# Discovery and installation
mcp-manager discover --query database
mcp-manager install-package dd-SQLite

# Server management
mcp-manager list
mcp-manager add myserver "python server.py"
mcp-manager remove myserver --force

# Configuration sync
mcp-manager sync --dry-run
mcp-manager detect-changes --watch
```

1.6.3 3. Terminal User Interface (TUI)

Launch: `mcp-manager tui`

> **Note**: `mcp-manager` with no arguments launches the interactive menu, not the TUI

Example Screen Layout:

```
              MCP Manager TUI

   Servers                Actions

    filesystem  [enabled ]          Add Server
    SQLite      [enabled ]          Discover Servers
    test-server [disabled]          Sync Changes
                                     Configure
                                     System Info


   Server Details
   Name: filesystem
   Type: docker-desktop
   Command: docker mcp server filesystem
   Status: enabled
   Description: File system operations for MCP


  [Tab] Switch panels  [Enter] Select  [q] Quit  [h] Help
```

1.7 Command Reference

1.7.1 Discovery Commands

1.7.1.1 `discover` Find available MCP servers from multiple sources

```
# Basic discovery
mcp-manager discover

# Search with query
mcp-manager discover --query "database sqlite"

# Filter by type
mcp-manager discover --type npm
mcp-manager discover --type docker-desktop

# Limit results
mcp-manager discover --limit 10

# Include detailed information
mcp-manager discover --detailed

# Update cached catalogs
mcp-manager discover --update-catalog
```

Example Output:

```
Discovering MCP servers across all sources...

             Docker Desktop MCP
3 servers available


 Install ID       Name           Tools    Description

 dd-SQLite        SQLite         3        Database operations
 dd-filesystem    filesystem     12       File system operations
 dd-search        search         2        Web search capabilities

               NPM Registry
2 servers available


 Install ID               Name                    Downloads  Description

 modelcontextprotocol-...  @modelcontextprotocol-  1.2K      File system
 mcp-server-sqlite        @mcp/server-sqlite      856        SQLite MCP
```

1.7.2 Installation Commands

1.7.2.1 `install-package` Install a server using its unique install ID

```
# Install Docker Desktop server
mcp-manager install-package dd-SQLite

# Install NPM server
mcp-manager install-package modelcontextprotocol-filesystem

# Install with specific configuration
mcp-manager install-package mcp-server --config config.json

# Force reinstallation
mcp-manager install-package dd-filesystem --force
```

1.7.2.2 `install` Install from discovery results (legacy)

```
# Install by index from last discovery
mcp-manager install 1

# Install specific server type
mcp-manager install --name SQLite --type docker-desktop
```

1.7.3 Server Management Commands

1.7.3.1 `add` Add a custom MCP server

```
# Basic custom server
mcp-manager add myserver "python /path/to/server.py"

# With arguments and environment
mcp-manager add database-server "npx @mcp/sqlite" \
  --args "--db-path /data/app.db" \
  --env "DEBUG=1"
```

```
# Docker container server
mcp-manager add containerized "docker run -i myimage:latest"
```

1.7.3.2 `remove` Remove an MCP server

```
# Interactive removal (prompts for confirmation)
mcp-manager remove myserver

# Force removal (no prompts)
mcp-manager remove myserver --force

# Remove with cleanup
mcp-manager remove myserver --cleanup
```

1.7.3.3 `enable` / `disable` Control server status

```
# Enable server
mcp-manager enable myserver

# Disable server
mcp-manager disable myserver

# Enable multiple servers
mcp-manager enable server1 server2 server3
```

1.7.3.4 `list` Display configured servers

```
# List all servers
mcp-manager list

# Filter by status
mcp-manager list --enabled
mcp-manager list --disabled

# Filter by type
mcp-manager list --type docker-desktop
mcp-manager list --type npm

# Detailed output
mcp-manager list --detailed

# JSON output
mcp-manager list --json
```

Example Outputs:

Standard List:

```
                          MCP Servers

 Name              Scope    Status         Type        Command

 filesystem         user   enabled   docker-desktop   docker
 SQLite             user   enabled   docker-desktop   docker
 test-server        user   enabled        npm         npx
```

```
custom-script      user   disabled      custom       python
```

Detailed List:

```
MCP Server Details

 filesystem (docker-desktop)
Status: enabled
Scope: user
Command: docker mcp server filesystem
Tools: read_file, write_file, list_directory, create_directory
Description: Provides file system access for MCP clients


 SQLite (docker-desktop)
Status: enabled
Scope: user
Command: docker mcp server SQLite
Tools: query, execute, schema
Description: SQLite database operations for MCP
```

1.7.4 Configuration Synchronization Commands

1.7.4.1 `sync` Synchronize with external configuration changes

```
# Interactive sync (prompts before applying changes)
mcp-manager sync

# Dry run (show what would change)
mcp-manager sync --dry-run

# Automatic sync (apply all changes)
mcp-manager sync --auto-apply
```

Example Sync Session:

```
External Configuration Sync

Detecting external changes...
Detected 2 configuration changes:

               Claude Internal Config
1 changes


Change            Server                Details

  Added          new-server            cmd: python,
                                         (external_server_not_in_catalog)

               Docker Desktop MCP
1 changes


Change            Server                Details
```

```
   Removed        old-server             (catalog_server_not_external)

Apply these changes to synchronize configurations? (y/N): y

  Applying synchronization changes...
    Added server: new-server
    Removed server from catalog: old-server

  Successfully applied 2 changes
  Synchronization complete
```

1.7.4.2 `detect-changes` Monitor external configuration changes

```
# One-time change detection
mcp-manager detect-changes

# Continuous monitoring
mcp-manager detect-changes --watch

# Custom interval monitoring
mcp-manager detect-changes --watch --interval 30
```

Example Watch Output:

```
  Monitoring external changes (interval: 5s, press Ctrl+C to stop)...

[18:30:15]   No changes detected
[18:30:20]   No changes detected
[18:30:25]   1 new changes detected at 18:30:25
  • server_added:docker:new-database-server

[18:30:30]   No changes detected
```

1.7.5 Monitoring Commands

1.7.5.1 `monitor` Background monitoring service

```
# Start monitoring service with auto-sync
mcp-manager monitor --start --auto-sync

# Start with custom interval
mcp-manager monitor --start --interval 120

# Check service status
mcp-manager monitor --status

# Stop service
mcp-manager monitor --stop
```

1.7.5.2 `monitor-status` Quick monitor status check

```
mcp-manager monitor-status
```

1.7.6 System Commands

1.7.6.1 `system-info` Display system information and diagnostics

```
mcp-manager system-info
```

Example Output:

```
MCP Manager System Information

System Environment
OS: macOS 14.5
Python: 3.11.5
MCP Manager: 1.0.0
Install Method: PyPI


Dependencies
Claude Code CLI:  Available (v0.8.1)
Docker Desktop:  Available (v4.21.1)
Docker MCP:  Available (3 servers enabled)
Git:  Available (v2.39.2)


Configuration
Config File: ~/.config/mcp-manager/config.toml
Log Level: INFO
Cache Directory: ~/.mcp-manager/cache
Change Detection:  Enabled
Auto Sync:  Disabled


Server Statistics
Total Servers: 4
Enabled: 3
Docker Desktop: 2
NPM: 1
Custom: 1
```

1.7.6.2 `check-sync` Check synchronization status

```
mcp-manager check-sync
```

1.7.6.3 `cleanup` Clean up problematic configurations

```
# Interactive cleanup
mcp-manager cleanup

# Automatic cleanup
mcp-manager cleanup --auto

# Deep cleanup (removes all cached data)
mcp-manager cleanup --deep
```

1.7.7 Configuration Commands

1.7.7.1 `configure` Configure or reconfigure servers

```
# Configure server interactively
mcp-manager configure myserver
```

```
# Show current configuration
mcp-manager configure myserver --show

# Configure with specific values
mcp-manager configure myserver --set "key=value"
```

1.8 Common Workflows

1.8.1 Workflow 1: New Project Setup

Scenario: Setting up MCP servers for a new AI development project

```
# Step 1: Discover available servers for your domain
mcp-manager discover --query "filesystem database"

# Step 2: Install essential servers
mcp-manager install-package dd-filesystem
mcp-manager install-package dd-SQLite

# Step 3: Add custom project server
mcp-manager add project-api "python api_server.py" \
  --args "--port 8080 --project myproject"

# Step 4: Verify setup
mcp-manager list

# Step 5: Test in Claude Code
claude mcp list
```

Expected Result: 3 servers (filesystem, SQLite, project-api) available in Claude Code

1.8.2 Workflow 2: Server Discovery and Evaluation

Scenario: Finding the best MCP server for specific functionality

```
# Step 1: Broad discovery
mcp-manager discover --query "web search"

# Step 2: Detailed comparison
mcp-manager discover --query "web search" --detailed

# Step 3: Install top candidate
mcp-manager install-package dd-search

# Step 4: Test functionality
mcp-manager list --detailed | grep search

# Step 5: Remove if unsatisfactory
mcp-manager remove search --force
```

1.8.3 Workflow 3: Configuration Synchronization

Scenario: Maintaining consistency when multiple tools modify MCP configs

```
# Step 1: Enable background monitoring
mcp-manager monitor --start --auto-sync

# Step 2: Make external changes (e.g., via Docker Desktop UI)
# - Enable/disable servers in Docker Desktop
# - Use claude mcp commands directly

# Step 3: Monitor detects changes automatically
# Check logs: tail -f ~/.mcp-manager/logs/mcp-manager.log

# Step 4: Manual sync if needed
mcp-manager sync --dry-run
mcp-manager sync --auto-apply

# Step 5: Verify consistency
mcp-manager check-sync
```

1.8.4 Workflow 4: Development Environment Migration

Scenario: Moving MCP configuration to a new development machine

```
# On source machine:
# Step 1: Export current configuration
mcp-manager list --json > mcp-servers-backup.json

# Step 2: Document custom servers
mcp-manager list --type custom --detailed

# On target machine:
# Step 3: Install MCP Manager
pip install mcp-manager

# Step 4: Recreate servers
mcp-manager install-package dd-filesystem
mcp-manager install-package dd-SQLite
mcp-manager add custom-server "python server.py"

# Step 5: Verify migration
mcp-manager list
mcp-manager system-info
```

1.8.5 Workflow 5: Troubleshooting Server Issues

Scenario: Debugging MCP server connectivity or configuration problems

```
# Step 1: Check system health
mcp-manager system-info

# Step 2: Verify server status
mcp-manager list --detailed

# Step 3: Check Claude Code integration
claude mcp list

# Step 4: Detect configuration drift
mcp-manager detect-changes
```

```
# Step 5: Clean up if needed
mcp-manager cleanup

# Step 6: Re-sync configurations
mcp-manager sync --auto-apply

# Step 7: Verify fix
mcp-manager check-sync
```

1.9 Use Cases & Examples

1.9.1 Use Case 1: Data Science Team

Scenario: Data science team needs file system access and database connectivity

Requirements: - Read/write files in project directories - Query SQLite databases for analysis - Access web search for research

Implementation:

```
# Team lead sets up standard servers
mcp-manager install-package dd-filesystem
mcp-manager install-package dd-SQLite
mcp-manager install-package dd-search

# Create team configuration file
cat > .mcp-manager.toml << EOF
[discovery]
preferred_sources = ["docker-desktop"]
quality_threshold = 8.0

[change_detection]
enabled = true
auto_sync = false
EOF

# Verify team setup
mcp-manager list
```

Result: Standardized MCP environment across all team members

1.9.2 Use Case 2: DevOps Automation

Scenario: Automated deployment pipeline needs MCP server management

Requirements: - Install servers via CI/CD pipeline - Synchronize configurations across environments - Monitor for configuration drift

Implementation:

```
#!/bin/bash
# deploy-mcp-servers.sh

# Install required servers
mcp-manager install-package dd-filesystem
mcp-manager install-package modelcontextprotocol-kubernetes
```

```
# Configure custom deployment server
mcp-manager add deployment-helper "python /opt/deploy/mcp_server.py" \
  --env "ENVIRONMENT=production" \
  --args "--config /opt/deploy/config.yaml"

# Enable monitoring
mcp-manager monitor --start --auto-sync --interval 300

# Verify deployment
mcp-manager check-sync || exit 1
```

Result: Automated, consistent MCP server deployment

1.9.3 Use Case 3: Multi-Project Organization

Scenario: Organization with multiple projects, each with specific MCP requirements

Project Structure:

```
organization/
  project-a/
      .mcp-manager.toml
      custom-servers/
  project-b/
      .mcp-manager.toml
      requirements.txt
  shared/
      global-config.toml
```

Project A Configuration:

```
# project-a/.mcp-manager.toml
[servers]
filesystem = { install_id = "dd-filesystem", required = true }
database = { install_id = "dd-SQLite", required = true }
api-gateway = {
  command = "python custom-servers/api_gateway.py",
  type = "custom",
  args = ["--project", "project-a"]
}

[change_detection]
enabled = true
scope = "project"
```

Project B Configuration:

```
# project-b/.mcp-manager.toml
[servers]
filesystem = { install_id = "modelcontextprotocol-filesystem", required = true }
search = { install_id = "dd-search", required = true }
nlp-tools = {
  command = "npx @nlp/mcp-server",
  type = "npm",
  args = ["--model", "gpt-4"]
}
```

```
[change_detection]
enabled = true
auto_sync = true
```

Usage:

```
# In project-a directory
cd project-a
mcp-manager install-package dd-filesystem
mcp-manager install-package dd-SQLite
mcp-manager add api-gateway "python custom-servers/api_gateway.py" --args "--project project-a"

# In project-b directory
cd ../project-b
mcp-manager install-package modelcontextprotocol-filesystem
mcp-manager install-package dd-search
mcp-manager install-package nlp-mcp-tools
```

Result: Project-specific MCP configurations with shared organizational policies

1.9.4 Use Case 4: AI Research Lab

Scenario: Research lab with frequently changing experimental MCP servers

Requirements: - Easy installation of experimental servers - Version management for research reproducibility
- Quick server switching for A/B testing

Implementation:

```
# Research experiment setup script
#!/bin/bash
# setup-experiment.sh

EXPERIMENT_NAME="$1"
EXPERIMENT_CONFIG="experiments/${EXPERIMENT_NAME}.yaml"

# Create experiment-specific configuration
mcp-manager add "${EXPERIMENT_NAME}-processor" \
  "python experiments/${EXPERIMENT_NAME}/processor.py" \
  --args "--config ${EXPERIMENT_CONFIG}"

# Install supporting servers based on experiment type
case "$EXPERIMENT_NAME" in
  "nlp-"*)
    mcp-manager install-package modelcontextprotocol-text
    mcp-manager install-package dd-search
    ;;
  "vision-"*)
    mcp-manager install-package modelcontextprotocol-vision
    mcp-manager install-package dd-filesystem
    ;;
  "data-"*)
    mcp-manager install-package dd-SQLite
    mcp-manager install-package modelcontextprotocol-pandas
    ;;
esac
```

```
# Start monitoring for this experiment
mcp-manager monitor --start --interval 30

echo "Experiment ${EXPERIMENT_NAME} MCP environment ready"
mcp-manager list --type custom
```

Usage:

```
# Setup NLP experiment
./setup-experiment.sh nlp-sentiment-analysis

# Switch to computer vision experiment
mcp-manager cleanup --auto
./setup-experiment.sh vision-object-detection

# List experiment servers
mcp-manager list --grep "nlp-\|vision-"
```

Result: Flexible, reproducible MCP environments for research experiments

1.9.5 Use Case 5: Enterprise Security Compliance

Scenario: Enterprise environment with strict security and compliance requirements

Requirements: - Centralized server approval process - Audit logging of all MCP operations - Restricted server installation sources

System Configuration:

```
# /etc/mcp-manager/config.toml (system-wide)
[security]
approved_sources = ["docker-desktop", "internal-registry"]
require_approval = true
audit_logging = true

[discovery]
blocked_sources = ["docker-hub"]
quality_threshold = 9.0

[logging]
level = "INFO"
audit_file = "/var/log/mcp-manager/audit.log"
format = "json"
```

User Workflow:

```
# User requests server installation
mcp-manager discover --query "database" --source approved

# System shows only approved sources
# User submits installation request
mcp-manager install-package dd-SQLite --request-approval

# Admin approves via system
# User receives notification and completes installation
mcp-manager install-package dd-SQLite --approved-token abc123
```

```
# All operations logged
tail -f /var/log/mcp-manager/audit.log
```

Audit Log Example:

```
{
  "timestamp": "2025-07-21T18:30:00Z",
  "user": "developer1",
  "action": "install_package",
  "server": "dd-SQLite",
  "source": "docker-desktop",
  "approval_token": "abc123",
  "result": "success"
}
```

Result: Secure, auditable MCP server management meeting enterprise compliance

---

1.10 Configuration

1.10.1 Configuration File Locations

The MCP Manager uses a hierarchical configuration system:

1. **System**: `/etc/mcp-manager/config.toml` (admin-managed)
2. **User**: `~/.config/mcp-manager/config.toml` (user preferences)
3. **Project**: `./.mcp-manager.toml` (project-specific)
4. **Environment**: `MCP_MANAGER_*` variables (runtime overrides)

1.10.2 Complete Configuration Example

```
# ~/.config/mcp-manager/config.toml

[general]
default_interface = "interactive"  # interactive, cli, tui
auto_update_check = true
verbose_output = false

[logging]
level = "INFO"                     # DEBUG, INFO, WARNING, ERROR
format = "text"                    # text, json
file = "~/.mcp-manager/logs/mcp-manager.log"
max_size = "10MB"
backup_count = 5
console_output = true

[discovery]
sources = ["docker-desktop", "npm", "docker-hub"]
cache_ttl = 3600                   # seconds
quality_threshold = 5.0            # minimum score for results
max_results = 50
parallel_requests = true
timeout = 30                       # seconds

[installation]
default_scope = "user"             # user, system, project
```

```toml
auto_enable = true
backup_before_changes = true
verify_after_install = true

[change_detection]
enabled = true
check_interval = 60                # seconds for background monitoring
auto_sync = false
operation_cooldown = 2.0           # seconds to prevent sync loops
watch_docker_config = true
watch_claude_configs = true

[servers]
# Pre-configured server definitions
filesystem = { install_id = "dd-filesystem", auto_install = false }
database = { install_id = "dd-SQLite", auto_install = false }

[ui]
color_output = true
progress_indicators = true
table_style = "rounded"            # ascii, rounded, double
pager = "auto"                     # auto, always, never

[security]
verify_signatures = true
allowed_sources = ["docker-desktop", "npm"]  # empty = all allowed
require_confirmation = true
audit_logging = false

[performance]
cache_enabled = true
concurrent_operations = 4
connection_timeout = 10
retry_attempts = 3
retry_delay = 1.0

[docker]
docker_command = "docker"
docker_desktop_integration = true
auto_import_gateway = true

[npm]
npm_command = "npx"
npm_registry = "https://registry.npmjs.org"
install_timeout = 120
```

1.10.3 Environment Variables

All configuration options can be overridden with environment variables:

```bash
# General settings
export MCP_MANAGER_DEFAULT_INTERFACE="cli"
export MCP_MANAGER_VERBOSE_OUTPUT="true"

# Logging
```

```
export MCP_MANAGER_LOG_LEVEL="DEBUG"
export MCP_MANAGER_LOG_FORMAT="json"
export MCP_MANAGER_LOG_FILE="/tmp/mcp-manager.log"

# Discovery
export MCP_MANAGER_DISCOVERY_SOURCES="docker-desktop,npm"
export MCP_MANAGER_CACHE_TTL="7200"
export MCP_MANAGER_QUALITY_THRESHOLD="8.0"

# Change detection
export MCP_MANAGER_CHANGE_DETECTION_ENABLED="true"
export MCP_MANAGER_AUTO_SYNC="true"
export MCP_MANAGER_CHECK_INTERVAL="30"

# Security
export MCP_MANAGER_REQUIRE_CONFIRMATION="false"
export MCP_MANAGER_AUDIT_LOGGING="true"
```

1.10.4 Configuration Commands

```
# Show current configuration
mcp-manager config

# Initialize default configuration
mcp-manager config --init

# Show configuration for specific section
mcp-manager config --section logging

# Set configuration value
mcp-manager config --set "discovery.quality_threshold=8.0"

# Validate configuration
mcp-manager config --validate

# Reset to defaults
mcp-manager config --reset
```

1.11 Troubleshooting

1.11.1 Common Issues and Solutions

1.11.1.1 Issue 1: "Claude Code CLI not found" Symptoms:

```
Error: claude command not found
Failed to execute: claude mcp list
```

Solutions:

```
# Check if Claude Code is installed
which claude

# Install Claude Code CLI if missing
# Follow: https://docs.anthropic.com/claude/docs/claude-code
```

```
# Add to PATH if installed but not found
export PATH="$PATH:/usr/local/bin"

# Verify installation
claude --version
```

1.11.1.2 Issue 2: "Docker Desktop servers not appearing" Symptoms:

```
Discovery shows no Docker Desktop servers
docker mcp commands fail
```

Solutions:

```
# Check Docker Desktop installation
docker --version

# Ensure Docker Desktop is running
docker info

# Check Docker MCP plugin availability
docker mcp --help

# Enable Docker Desktop MCP servers
docker mcp server enable SQLite
docker mcp server enable filesystem

# Import to Claude Code
claude mcp add-from-claude-desktop docker-gateway

# Verify integration
mcp-manager check-sync
```

1.11.1.3 Issue 3: "Permission denied errors" Symptoms:

```
Permission denied: ~/.config/mcp-manager/
Failed to write configuration file
```

Solutions:

```
# Check file permissions
ls -la ~/.config/mcp-manager/

# Create directories if missing
mkdir -p ~/.config/mcp-manager/logs
mkdir -p ~/.config/mcp-manager/cache

# Fix permissions
chmod 755 ~/.config/mcp-manager
chmod 644 ~/.config/mcp-manager/config.toml

# Run with proper user context
# Avoid running with sudo unless necessary
```

1.11.1.4 Issue 4: "Server installation fails" Symptoms:

```
Failed to install package: dd-SQLite
Server not found in Docker Desktop catalog
```

Solutions:

```
# Update discovery cache
mcp-manager discover --update-catalog

# Check available servers
mcp-manager discover --query SQLite

# Try alternative install ID
mcp-manager discover --detailed | grep -i sqlite

# Manual installation
docker mcp server enable SQLite
claude mcp add-from-claude-desktop docker-gateway
mcp-manager sync
```

1.11.1.5 Issue 5: "Sync conflicts and loops" Symptoms:

```
Continuous sync operations
Background monitor consuming CPU
Configuration changes keep reverting
```

Solutions:

```
# Stop background monitoring
mcp-manager monitor --stop

# Check sync protection status
mcp-manager check-sync

# Clear sync history
mcp-manager cleanup --deep

# Reset change detection
mcp-manager detect-changes --reset

# Restart with fresh state
mcp-manager monitor --start --interval 300
```

1.11.2 Diagnostic Commands

```
# Comprehensive system check
mcp-manager system-info

# Verify all dependencies
mcp-manager system-info --verify-deps

# Check configuration validity
mcp-manager config --validate

# Test Claude Code integration
claude mcp list

# Test Docker Desktop integration
docker mcp server list
```

```
# Check log files
tail -f ~/.mcp-manager/logs/mcp-manager.log

# Enable debug logging
export MCP_MANAGER_LOG_LEVEL="DEBUG"
mcp-manager discover --query test
```

1.11.3 Getting Help

Community Support: - GitHub Issues: https://github.com/blemis/mcp-manager-python/issues - Discussions: https://github.com/blemis/mcp-manager-python/discussions - Documentation: https://github.com/blemis/mcp-manager-python/wiki

Bug Reports: When reporting bugs, include:

```
# System information
mcp-manager system-info

# Configuration (remove sensitive data)
mcp-manager config

# Recent log entries
tail -50 ~/.mcp-manager/logs/mcp-manager.log

# Steps to reproduce the issue
```

1.12 Advanced Usage

1.12.1 Custom Discovery Sources

Create Custom Discovery Plugin:

```
# ~/.config/mcp-manager/plugins/custom_discovery.py

from mcp_manager.core.discovery import DiscoverySource
from typing import List, Dict, Any

class CustomRegistrySource(DiscoverySource):
    """Custom internal registry discovery source."""

    def __init__(self):
        super().__init__("custom-registry", "Internal Registry")

    async def discover_servers(self, query: str = "") -> List[Dict[str, Any]]:
        # Implement custom discovery logic
        servers = await self._fetch_from_internal_registry(query)
        return [self._format_server(s) for s in servers]

    async def _fetch_from_internal_registry(self, query: str):
        # Custom implementation
        pass

    def _format_server(self, server_data: Dict) -> Dict[str, Any]:
        return {
            'install_id': f"custom-{server_data['name']}",
```

```
            'name': server_data['name'],
            'type': 'custom',
            'description': server_data.get('description', ''),
            'command': server_data['command'],
            'args': server_data.get('args', []),
            'quality_score': server_data.get('rating', 5.0)
        }
```

Register Custom Source:

```
# ~/.config/mcp-manager/config.toml
[discovery]
sources = ["docker-desktop", "npm", "custom-registry"]
plugin_paths = ["~/.config/mcp-manager/plugins"]
```

1.12.2 Scripting and Automation

Batch Server Management:

```
#!/bin/bash
# batch-server-setup.sh

# Read server list from file
SERVERS_FILE="servers.txt"

while IFS= read -r server_id; do
    echo "Installing: $server_id"

    if mcp-manager install-package "$server_id"; then
        echo "  Installed: $server_id"
    else
        echo "  Failed: $server_id"
        # Log failure for later review
        echo "$server_id" >> failed-installs.txt
    fi

    # Rate limiting
    sleep 2
done < "$SERVERS_FILE"

# Verify all installations
mcp-manager list --json > installation-report.json
echo "Installation report saved to installation-report.json"
```

Configuration Backup and Restore:

```
#!/bin/bash
# backup-mcp-config.sh

BACKUP_DIR="mcp-backup-$(date +%Y%m%d-%H%M%S)"
mkdir -p "$BACKUP_DIR"

# Export current server configuration
mcp-manager list --json > "$BACKUP_DIR/servers.json"

# Backup configuration files
cp ~/.config/mcp-manager/config.toml "$BACKUP_DIR/"
```

```
cp ~/.claude.json "$BACKUP_DIR/" 2>/dev/null || true

# Create restore script
cat > "$BACKUP_DIR/restore.sh" << 'EOF'
#!/bin/bash
echo "Restoring MCP configuration from backup..."

# Stop any monitoring
mcp-manager monitor --stop 2>/dev/null || true

# Clear current configuration
mcp-manager cleanup --deep --auto

# Restore servers from backup
while IFS= read -r line; do
    server_id=$(echo "$line" | jq -r '.install_id // empty')
    if [ -n "$server_id" ]; then
        mcp-manager install-package "$server_id"
    fi
done < <(jq -r '.[] | @json' servers.json)

echo "Restore complete"
EOF

chmod +x "$BACKUP_DIR/restore.sh"
echo "Backup created in: $BACKUP_DIR"
```

1.12.3 Integration with External Tools

Jenkins Pipeline Integration:

```
// Jenkinsfile
pipeline {
    agent any

    stages {
        stage('Setup MCP Environment') {
            steps {
                sh '''
                    # Install MCP Manager if not present
                    pip install mcp-manager

                    # Setup project-specific servers
                    mcp-manager install-package dd-filesystem
                    mcp-manager install-package dd-SQLite

                    # Configure project server
                    mcp-manager add ci-helper "python ci/mcp_server.py" \
                        --env "JENKINS_BUILD_ID=${BUILD_ID}" \
                        --args "--project ${JOB_NAME}"
                '''
            }
        }

        stage('Verify MCP Setup') {
```

```
            steps {
                sh '''
                    # Verify MCP environment
                    mcp-manager check-sync
                    mcp-manager list --json > mcp-servers.json
                '''

                archiveArtifacts artifacts: 'mcp-servers.json'
            }
        }
    }

    post {
        always {
            sh 'mcp-manager cleanup --auto || true'
        }
    }
}
```

Docker Compose Integration (Future Release):

Note: Docker Compose support will be available in a future release

```
# docker-compose.yml (coming soon)
version: '3.8'

services:
  app:
    image: myapp:latest
    depends_on:
      - mcp-manager
    environment:
      - MCP_MANAGER_HOST=mcp-manager
    volumes:
      - mcp-data:/mcp

  mcp-manager:
    image: mcpmanager/mcp-manager:latest
    ports:
      - "8080:8080"  # API server mode
    volumes:
      - mcp-data:/data
      - ./mcp-config:/config
    environment:
      - MCP_MANAGER_CONFIG_PATH=/config/config.toml
      - MCP_MANAGER_DATA_PATH=/data
      - MCP_MANAGER_LOG_LEVEL=INFO
    command: ["mcp-manager", "monitor", "--start", "--auto-sync", "--api-mode"]

volumes:
  mcp-data:
```

1.13 Uninstallation

1.13.1 Complete Removal

Step 1: Stop Background Services

```
# Stop any running monitoring services
mcp-manager monitor --stop

# Kill any background processes
pkill -f mcp-manager
```

Step 2: Remove Servers (Optional)

```
# List all managed servers
mcp-manager list

# Remove specific servers if desired
mcp-manager remove server-name --force

# Or clean up all managed servers
mcp-manager cleanup --deep --auto
```

Step 3: Remove MCP Manager

```
# If installed via pip
pip uninstall mcp-manager

# If installed via development mode
pip uninstall mcp-manager
rm -rf /path/to/mcp-manager-python

# If installed via Docker (future release)
docker rmi mcpmanager/mcp-manager:latest
```

Step 4: Remove Configuration and Data

```
# Remove user configuration
rm -rf ~/.config/mcp-manager

# Remove system configuration (if admin)
sudo rm -rf /etc/mcp-manager

# Remove logs and cache
rm -rf ~/.mcp-manager

# Remove any project configurations
find . -name ".mcp-manager.toml" -delete
```

Step 5: Clean Up Environment

```
# Remove environment variables from shell profile
# Edit ~/.bashrc, ~/.zshrc, etc. and remove MCP_MANAGER_* exports

# Unset current session variables
unset $(env | grep MCP_MANAGER_ | cut -d= -f1)
```

1.13.2 Partial Removal (Keep Servers)

If you want to remove MCP Manager but keep your configured servers:

```
# Export current configuration
mcp-manager list --json > mcp-servers-backup.json

# Note: Servers will remain in Claude Code's configuration
# They can still be managed via claude mcp commands

# Remove only MCP Manager
pip uninstall mcp-manager
rm -rf ~/.config/mcp-manager
```

1.13.3 Verification of Removal

```
# Verify MCP Manager is removed
mcp-manager --version  # Should return "command not found"

# Check that servers are still accessible in Claude Code (if kept)
claude mcp list

# Verify no background processes
ps aux | grep mcp-manager

# Check for remaining files
find ~ -name "*mcp-manager*" -type f
```

1.14 Quick Reference

1.14.1 Essential Commands

```
# Interactive menu (most common)
mcp-manager

# Discover and install servers
mcp-manager discover --query filesystem
mcp-manager install-package dd-filesystem

# Manage servers
mcp-manager list
mcp-manager enable myserver
mcp-manager remove myserver --force

# Synchronization
mcp-manager sync --dry-run
mcp-manager sync --auto-apply
mcp-manager detect-changes --watch

# System maintenance
mcp-manager system-info
mcp-manager cleanup
mcp-manager check-sync
```

1.14.2 Configuration Locations

```
System:     /etc/mcp-manager/config.toml
User:       ~/.config/mcp-manager/config.toml
```

```
Project:    ./.mcp-manager.toml
Logs:       ~/.mcp-manager/logs/mcp-manager.log
Cache:      ~/.mcp-manager/cache/
```

1.14.3 Environment Variables

```
export MCP_MANAGER_LOG_LEVEL="DEBUG"
export MCP_MANAGER_AUTO_SYNC="true"
export MCP_MANAGER_CHECK_INTERVAL="60"
```

1.14.4 Help and Documentation

```
mcp-manager --help                    # General help
mcp-manager <command> --help          # Command-specific help
mcp-manager system-info               # System diagnostics
```

1.15 Roadmap & Future Releases

The following features are planned for future releases:

1.15.1 v1.1 - PyPI Distribution

- **PyPI Package**: Official package distribution via `pip install mcp-manager`
- **Simplified Installation**: One-command installation without git clone
- **Version Management**: Semantic versioning and upgrade paths

1.15.2 v1.2 - Container Support

- **Docker Images**: Official Docker images on Docker Hub
- **Docker Compose**: Pre-configured compose files for containerized deployments
- **API Server Mode**: REST API for programmatic access and integration
- **Kubernetes**: Helm charts and K8s deployment manifests

1.15.3 v1.3 - Enterprise Features

- **Centralized Management**: Organization-wide server policies and approval workflows
- **Audit Logging**: Enhanced audit trails and compliance reporting
- **Multi-Tenant**: Project isolation and team-based access controls
- **Metrics Dashboard**: Web-based monitoring and analytics interface

1.15.4 v2.0 - Advanced Integration

- **IDE Extensions**: VS Code and JetBrains plugin support
- **CI/CD Integration**: Native GitHub Actions and Jenkins plugins
- **Plugin Architecture**: Custom discovery sources and server types
- **Distributed Discovery**: Organizational server registries and catalogs

---

This user guide covers MCP Manager version 1.0. For the latest updates and additional examples, visit the project repository at https://github.com/blemis/mcp-manager-python