User Guide
- Table of Contents
- Auto-Configuration
  - Quick Start
  - How It Works
  - Detected Technologies
  - Configuration
- Agent System
  - Agent Tiers
  - Agent Commands
  - Slash Commands
  - Agent Delegation
- Skills System
  - Overview
  - Access Skills Management
  - Auto-Linking (Recommended)
  - Three-Tier System
  - Custom Skills
  - Skill Selection
  - Best Practices
- Memory System
  - How It Works
  - Memory Categories
  - Memory Commands
  - Best Practices
- Local Process Management
  - Quick Start
  - Features
  - Configuration
  - Commands
- Session Management
  - Pause Session
  - Resume Session
  - Best Practices
- Real-Time Monitoring
  - Start Monitoring
  - Dashboard Features
  - WebSocket Events
- MCP Gateway
  - Overview
  - Available MCP Tools
  - MCP Tool Auto-Install
  - Configuration
- Best Practices
  - Project Setup
  - Agent Usage
  - Memory Management
  - Session Management
  - Monitoring

User Guide

Complete guide to Claude MPM features and workflows.

Table of Contents

- Auto-Configuration
- Agent System
- Skills System
- Memory System
- Local Process Management
- Session Management
- Real-Time Monitoring
- MCP Gateway
- Best Practices

Auto-Configuration

NEW in v4.10.0 - Automatically detect your project stack and configure agents.

Quick Start

```
# Auto-configure current project
claude-mpm auto-configure

# Preview without applying
claude-mpm auto-configure --preview

# Lower confidence threshold (default 80%)
claude-mpm auto-configure --threshold 60
```

How It Works

1. **Language Detection**: Scans project files to identify languages
2. **Framework Detection**: Analyzes configuration files to detect frameworks
3. **Tool Detection**: Identifies build tools, testing frameworks, databases
4. **Agent Recommendation**: Suggests agents with confidence scores
5. **Deployment**: Copies recommended agents to `.claude-agents/`

Detected Technologies

Languages: Python, JavaScript, TypeScript, Go, Rust, PHP, Ruby, Java, C++, C#

Frameworks: - Python: FastAPI, Flask, Django - JavaScript/TypeScript: Next.js, React, Express, Vue, Angular - Go: Gin, Echo, standard library - Rust: Axum, Actix, Rocket - PHP: Laravel, Symfony - Ruby: Rails, Sinatra

Tools: Docker, Kubernetes, PostgreSQL, MySQL, MongoDB, Redis, testing frameworks

Configuration

Settings in `.claude-mpm/config.yaml`:

```yaml
auto_configuration:
  enabled: true
  confidence_threshold: 80
  auto_deploy: true
  include_experimental: false
```

Agent System

Claude MPM uses a three-tier agent hierarchy: PROJECT > USER > SYSTEM

Agent Tiers

PROJECT Tier (`.claude-agents/`): - Project-specific agents - Highest priority - Overrides USER and SYSTEM agents

USER Tier (`~/.claude-agents/`): - Personal agents - Overrides SYSTEM agents - Shared across projects

SYSTEM Tier (bundled): - Built-in agents - Default agents like PM, Research, Engineer

Agent Commands

```bash
# List all agents
claude-mpm agents list

# List by tier
claude-mpm agents list --by-tier

# Create new agent
claude-mpm agents create <name>

# Deploy agents
claude-mpm agents deploy

# Validate agents
claude-mpm agents validate
```

Slash Commands

Use in Claude Code sessions:

```
# Manage agents
/mpm-agents

# List available agents
/mpm-agents list

# Deploy recommended agents
/mpm-agents deploy
```

Agent Delegation

The PM (Project Manager) agent orchestrates workflow by delegating to specialists:

1. **Research Agent**: Codebase analysis, documentation review
2. **Engineer Agent**: Implementation, refactoring
3. **QA Agent**: Testing, validation, quality checks
4. **Documentation Agent**: Documentation generation

Example Workflow:

```
User: "Add authentication to the API"
↓
PM: Plans approach, delegates to Research
↓
Research: Analyzes codebase, identifies patterns
↓
PM: Delegates to Engineer
↓
Engineer: Implements authentication
↓
PM: Delegates to QA
↓
QA: Creates tests, validates implementation
↓
PM: Delegates to Documentation
↓
Documentation: Updates API docs
```

Skills System

NEW in v4.15.0 - Enhance agent capabilities with reusable skill modules.

Overview

Claude MPM includes 15 bundled skills providing specialized expertise: `git-workflow`, `test-driven-development`, `code-review`, `refactoring-patterns`, `security-scanning`, `database-migration`, `docker-containerization`, `api-documentation`, `performance-profiling`, `systematic-debugging`, `async-testing`, `json-data-handling`, `pdf`, `xlsx`, `imagemagick`.

Access Skills Management

```
claude-mpm configure
# Select option [2] Skills Management
```

Interactive menu provides: - View bundled skills - Select skills for agents - Auto-link skills to agents - Manage custom skills

Auto-Linking (Recommended)

Automatically maps skills to compatible agents based on agent type:

```
# In Skills Management menu
# Select option [4] Auto-link skills
```

Maps skills intelligently: - `git-workflow` → version control agents - `test-driven-development` → QA and engineer agents - `security-scanning` → security and ops agents

Three-Tier System

Skills follow the same hierarchy as agents:

1. **BUNDLED**: 15 included skills (system-wide)
2. **USER**: Personal skills in `~/.claude/skills/`
3. **PROJECT**: Project-specific skills in `.claude/skills/`

Priority: PROJECT > USER > BUNDLED (project skills override user, user overrides bundled)

Custom Skills

Create markdown files in `.claude/skills/` for project-specific expertise:

```
mkdir -p .claude/skills
cat > .claude/skills/custom-api-patterns.md << 'EOF'
# Custom API Patterns

## Project Guidelines
- Use async handlers for all endpoints
- Validate with Pydantic models
- Return structured JSON responses
EOF
```

Skills auto-discover on next run. Restart Claude Code session to activate.

Skill Selection

Manually assign skills to specific agents:

```
# In Skills Management menu
# Select option [2] Select skills for an agent
# Choose agent, then select skills from list
```

Configuration saved to `.claude-mpm/config.yaml`:

```yaml
skills:
  assignments:
    engineer:
      - git-workflow
      - test-driven-development
      - refactoring-patterns
```

Best Practices

Use auto-linking for quick setup: Covers most common scenarios efficiently.

Add project skills for domain expertise: Store project-specific patterns, conventions, and guidelines in `.claude/skills/`.

Keep skills focused: One skill per expertise area (e.g., separate database patterns from API patterns).

Update skills as patterns evolve: Skills are living documentation of project best practices.

Memory System

Persistent learning across sessions using project-specific knowledge graphs.

How It Works

Agents can store learnings via JSON response fields:

```json
{
  "memory-update": {
    "Project Architecture": ["Uses FastAPI with async endpoints"],
    "Implementation Guidelines": ["Always use Pydantic models for
        validation"],
    "Current Technical Context": ["Authentication uses JWT
        tokens"]
  }
}
```

Or simplified:

```json
{
  "remember": ["API uses REST conventions", "Tests use pytest
        fixtures"]
}
```

Memory Categories

- **Project Architecture**: Architectural patterns and structures
- **Implementation Guidelines**: Coding standards and practices
- **Current Technical Context**: Project-specific technical details

Memory Commands

```
# Query project memories
claude-mpm recall "authentication"

# View memory statistics
claude-mpm stats

# Enhance prompt with memories
claude-mpm enhance "how should I implement login?"
```

Best Practices

Store project-specific information: - ✅ "API uses FastAPI 0.104.0 with async" - ✅ "Tests require pytest-asyncio fixture" - ❌ "Python is a programming language" (too generic)

Keep memories relevant: - ✅ "Auth tokens expire after 24 hours" - ✅ "Database uses alembic for migrations" - ❌ "Function should return None" (too specific to one function)

Local Process Management

NEW in v4.13.0 - Professional-grade local deployment with health monitoring.

Quick Start

```
# Start development server
claude-mpm local-deploy start \
  --command "npm run dev" \
  --name "nextjs-dev" \
  --auto-restart

# Monitor deployment
claude-mpm local-deploy monitor <deployment-id>

# List all deployments
claude-mpm local-deploy list

# Stop deployment
claude-mpm local-deploy stop <deployment-id>
```

Features

Three-Tier Health Checks: 1. **HTTP Health**: Endpoint availability and response codes 2. **Process Health**: Process status and stability 3. **Resource Health**: Memory and CPU usage

Auto-Restart: - Automatic restart on crash - Exponential backoff (1s → 2s → 4s → 8s → 16s → 32s → 60s max) - Circuit breaker after 5 consecutive failures

Monitoring: - Memory leak detection (configurable threshold) - Resource exhaustion prevention - Log error pattern matching - Real-time metrics

Configuration

Create `.claude-mpm/local-ops-config.yaml`:

```yaml
deployments:
  nextjs-dev:
    command: "npm run dev"
    directory: "."
    health_check:
      enabled: true
      url: "http://localhost:3000"
      interval: 30
      timeout: 10
      initial_delay: 5
    auto_restart:
      enabled: true
      max_attempts: 5
    resource_monitoring:
      memory_threshold_mb: 512
      check_interval: 60
    log_monitoring:
      error_patterns:
        - "ERROR"
        - "FATAL"
        - "Exception"
```

Commands

```bash
# Start with config file
claude-mpm local-deploy start --config nextjs-dev

# Start with inline options
claude-mpm local-deploy start \
  --command "python -m uvicorn app:app --reload" \
  --name "api-server" \
  --health-url "http://localhost:8000/health" \
  --auto-restart

# Monitor specific deployment
claude-mpm local-deploy monitor <deployment-id>
```

```
# View logs
claude-mpm local-deploy logs <deployment-id>

# Restart deployment
claude-mpm local-deploy restart <deployment-id>

# Stop deployment
claude-mpm local-deploy stop <deployment-id>

# Stop all deployments
claude-mpm local-deploy stop --all
```

Session Management

Save and resume Claude Code sessions with full context preservation.

Pause Session

```
# In Claude Code session
/pause

# Or CLI
claude-mpm mpm-init pause
```

What's saved: - Conversation history - Current git state (branch, status) - Todo list - Accomplishments - Project context

Resume Session

```
# In Claude Code session
/resume

# Or CLI
claude-mpm mpm-init resume
```

What happens: - Session state restored - Automatic change detection since pause - Git diff shown if changes detected - Context enriched with changes

Best Practices

When to pause: - Before switching branches - Before major refactoring - End of work session - Before long breaks

What to check after resume: - Review detected changes - Check git status - Verify branch context - Review todos

Real-Time Monitoring

Live dashboard showing agent collaboration and system metrics.

Start Monitoring

```
# Interactive mode with dashboard
claude-mpm run --monitor

# Or start dashboard separately
claude-mpm monitor
```

Dashboard opens at http://localhost:8765

Dashboard Features

Agent Activity: - Active agents and their current tasks - Delegation flow visualization - Agent communication logs

System Metrics: - Memory usage - Request latency - Cache hit rates - Hook execution times

Session Information: - Current session ID - Session duration - Total requests - Active tickets

WebSocket Events

Real-time updates via WebSocket: - Agent started/stopped - Task delegated - Task completed - Memory updated - Error occurred

MCP Gateway

Model Context Protocol integration for extensible tools.

Overview

MCP Gateway provides: - External tool connectivity - Standardized tool interfaces - Dynamic tool loading - Error handling and fallbacks

Available MCP Tools

Included: - `kuzu-memory`: Project-specific knowledge graphs (bundled)

Optional: - `mcp-vector-search`: Semantic code search (auto-install on first use)

MCP Tool Auto-Install

When you first use semantic search features:

```
⚠️  mcp-vector-search not found
This package enables semantic code search (optional feature).

Installation options:
  1. Install via pip (recommended for this project)
  2. Install via pipx (isolated, system-wide)
  3. Skip (use traditional grep/glob instead)

Choose option (1/2/3) [3]:
```

Recommendations: - **Option 1 (pip)**: Best for project-specific work - **Option 2 (pipx)**: Best for system-wide availability - **Option 3 (skip)**: System continues with grep/glob

After installation, vector search works seamlessly without prompts.

Configuration

MCP settings in `.claude-mpm/config.yaml`:

```yaml
mcp_gateway:
  enabled: true
  tools:
    - name: kuzu-memory
      enabled: true
    - name: mcp-vector-search
      enabled: true
      auto_install: true
```

Best Practices

Project Setup

1. **Initialize once per project**: Run `claude-mpm init` or `/mpm-init`
2. **Auto-configure stack**: Run `claude-mpm auto-configure`
3. **Review deployed agents**: Check `.claude-agents/` directory
4. **Configure local deployments**: Set up `.claude-mpm/local-ops-config.yaml`

Agent Usage

1. **Let PM delegate**: Trust the PM agent to orchestrate workflow
2. **Provide clear context**: Give agents specific, actionable requests
3. **Review delegation**: Use `--monitor` to see agent collaboration
4. **Customize agents**: Add project-specific agents to `.claude-agents/`

Memory Management

1. **Store project-specific info**: Keep memories relevant to the project
2. **Use clear categories**: Organize under Architecture, Guidelines, Context

3. **Review memories**: Check what's stored with `claude-mpm recall`
4. **Update outdated info**: Memories persist across sessions

Session Management

1. **Pause before switching**: Save state before branch changes
2. **Resume with context**: Review detected changes after resume
3. **Track work**: Use todos to maintain continuity
4. **Keep sessions focused**: Separate major features into different sessions

Monitoring

1. **Use for complex tasks**: Enable `--monitor` for multi-step work
2. **Watch delegation flow**: Understand agent collaboration patterns
3. **Check metrics**: Monitor performance and identify bottlenecks
4. **Review logs**: Use dashboard to debug issues

Next Steps: - Troubleshooting: See troubleshooting.md - Developer docs: See ../developer/architecture.md - Agent creation: See ../agents/creating-agents.md