

# Claude-Flow Alpha v86 - Vollständige CLI-Dokumentation








## Inhaltsverzeichnis

1. [Einführung](#)
  2. [Installation](#)
  3. [Grundlegende Befehle](#)
  4. [Erweiterte Befehle](#)
  5. [Kommandozeilenoptionen](#)
  6. [Swarm-Orchestrierung](#)
  7. [SPARC-Modus](#)
  8. [Memory Management](#)
  9. [Task Management](#)
  10. [MCP Integration](#)
  11. [Monitoring & Debugging](#)
  12. [Konfiguration](#)
  13. [Umgebungsvariablen](#)
  14. [Beispiel-Workflows](#)
  15. [Fehlerbehebung](#)
- 

## Einführung

Claude-Flow v2.0.0-alpha.86 ist eine Enterprise-Grade AI-Orchestrierungsplattform, die Hive-Mind Swarm Intelligence, neuronale Mustererkennung und 87 erweiterte MCP-Tools kombiniert.

## Hauptmerkmale

-  **Hive-Mind Intelligence:** Queen-geführte AI-Koordination
  -  **Neural Networks:** 27+ kognitive Modelle mit WASM SIMD-Beschleunigung
  -  **87 MCP Tools:** Umfassendes Toolkit für Swarm-Orchestrierung
  -  **Dynamic Agent Architecture (DAA):** Selbstorganisierende Agenten
  -  **SQLite Memory System:** Persistente .swarm/memory.db mit 12 Tabellen
  -  **Advanced Hooks System:** Automatisierte Workflows
  -  **GitHub Integration:** 6 spezialisierte Modi
- 

## Installation

## Schnellstart (Empfohlen)

```
bash

# Direkte Ausführung ohne Installation
npx claude-flow@alpha

# Mit SPARC-Umgebung initialisieren
npx claude-flow@alpha init --sparc --force
```

## Globale Installation

```
bash

# Global installieren
npm install -g claude-flow@alpha

# Version überprüfen
claude-flow --version

# Ausgabe: 2.0.0-alpha.86
```

## Projekt-Installation

```
bash

# Als Dev-Dependency
npm install --save-dev claude-flow@alpha

# In package.json scripts einbinden
"scripts": {
  "cf": "claude-flow",
  "cf:init": "claude-flow init --sparc"
}
```

## Voraussetzungen

```
bash

# Claude Code muss installiert sein
npm install -g @anthropic-ai/claude-code

# Optional: Berechtigungen überspringen (Vorsicht!)
claude --dangerously-skip-permissions
```

---

## Grundlegende Befehle

## init - Initialisierung

```
bash

# Basis-Initialisierung
npx claude-flow@alpha init

# Mit allen Features
npx claude-flow@alpha init --force --sparc --hive-mind --neural-enhanced

# Optionen:
--force           # Überschreibt existierende Konfiguration
--sparc           # Aktiviert SPARC-Entwicklungsumgebung
--hive-mind       # Aktiviert Hive-Mind-Architektur
--neural-enhanced # Aktiviert neuronale Verbesserungen
```

## start - System starten

```
bash

# Standard-Start
npx claude-flow@alpha start

# Mit Web-UI
npx claude-flow@alpha start --ui --port 3000

# Als Daemon
npx claude-flow@alpha start --daemon

# Optionen:
--ui           # Aktiviert Web-Interface
--port <number> # Port-Nummer (Standard: 3001)
--daemon       # Läuft im Hintergrund
--verbose      # Ausführliche Ausgabe
```

## status - Systemstatus

```
bash
```

*# Einfacher Status*

```
npx claude-flow@alpha status
```

*# Detaillierter Status*

```
npx claude-flow@alpha status --detailed
```

*# JSON-Format*

```
npx claude-flow@alpha status --json
```

*# Optionen:*

```
--detailed      # Zeigt erweiterte Informationen
```

```
--json          # Ausgabe als JSON
```

```
--health        # Nur Health-Check
```

## stop - System stoppen

bash

*# Normales Stoppen*

```
npx claude-flow@alpha stop
```

*# Erzwungenes Stoppen*

```
npx claude-flow@alpha stop --force
```

*# Mit Cleanup*

```
npx claude-flow@alpha stop --cleanup
```

*# Optionen:*

```
--force          # Erzwingt Beendigung
```

```
--cleanup        # Räumt temporäre Dateien auf
```

```
--save-state     # Speichert Zustand vor Beendigung
```

---

## Erweiterte Befehle

### agent - Agent-Management

bash

#### *# Agent spawnen*

```
npx claude-flow@alpha agent spawn <type> [options]
```

```
npx claude-flow@alpha agent spawn researcher --name "DataBot" --priority 8
```

#### *# Agent-Info*

```
npx claude-flow@alpha agent info <agent-id>
```

```
npx claude-flow@alpha agent list
```

```
npx claude-flow@alpha agent list --active
```

#### *# Agent terminieren*

```
npx claude-flow@alpha agent terminate <agent-id>
```

```
npx claude-flow@alpha agent terminate all --force
```

#### *# Optionen:*

```
--name <string>    # Agent-Name
```

```
--priority <1-10>  # Prioritätslevel
```

```
--memory <size>    # Memory-Allocation
```

```
--timeout <seconds> # Timeout-Einstellung
```

```
--auto-restart     # Automatischer Neustart bei Fehler
```

## swarm - Swarm-Orchestrierung

```
bash
```

#### *# Basis Swarm-Deployment*

```
npx claude-flow@alpha swarm "Build REST API" --claude
```

#### *# Erweiterte Swarm-Konfiguration*

```
npx claude-flow@alpha swarm "Build e-commerce platform" \
```

```
--strategy parallel \
```

```
--agents 10 \
```

```
--memory-namespace ecommerce \
```

```
--monitor
```

#### *# Optionen:*

```
--strategy <type>    # parallel|sequential|adaptive|research
```

```
--agents <number>    # Anzahl der Agenten (1-20)
```

```
--claude             # Claude-Integration aktivieren
```

```
--memory-namespace  # Memory-Namespace
```

```
--monitor            # Live-Monitoring
```

```
--timeout <minutes>  # Gesamttimeout
```

```
--max-retries <n>     # Maximale Wiederholungen
```

## hive-mind - Hive-Mind-Kontrolle

```
bash
```

*# Hive-Mind Wizard*

```
npx claude-flow@alpha hive-mind wizard
```

*# Hive-Mind spawnen*

```
npx claude-flow@alpha hive-mind spawn "Task description" \  
  --agents 8 \  
  --claude \  
  --coordination-level high
```

*# Test-Modus*

```
npx claude-flow@alpha hive-mind test \  
  --agents 5 \  
  --coordination-test
```

*# Optionen:*

```
--agents <number>      # Anzahl Worker-Agenten  
--coordination-level    # low|medium|high  
--neural-patterns       # enabled|disabled  
--memory-compression    # low|medium|high  
--queen-strategy        # balanced|aggressive|conservative
```

## Kommandozeilenoptionen

### Globale Optionen (für alle Befehle)

```
bash  
  
--help, -h          # Hilfe anzeigen  
--version, -v        # Version anzeigen  
--verbose            # Ausführliche Ausgabe  
--quiet, -q          # Minimale Ausgabe  
--json              # JSON-Ausgabe  
--no-color           # Keine Farbausgabe  
--config <path>      # Alternativer Config-Pfad  
--log-level <level>  # debug|info|warn|error  
--log-file <path>    # Log in Datei schreiben
```

### Debug-Optionen

```
bash
```

```
--debug           # Debug-Modus aktivieren
--mcp-debug       # MCP-Debug-Modus
--trace           # Trace-Level-Logging
--profile         # Performance-Profiling
--dry-run         # Simulation ohne Ausführung
```

## Performance-Optionen

```
bash

--parallel        # Parallele Ausführung
--max-workers <n> # Maximale Worker-Threads
--batch-size <n>  # Batch-Größe für Operationen
--cache           # Caching aktivieren
--memory-limit <MB> # Memory-Limit setzen
```

## SPARC-Modus

### Verfügbare SPARC-Modi

```
bash

# Alle Modi anzeigen
npx claude-flow@alpha sparc modes

# Spezifischen Modus ausführen
npx claude-flow@alpha sparc run <mode> "task description"
```

## SPARC-Modi im Detail

### architect - System-Architektur

```
bash

npx claude-flow@alpha sparc run architect "design microservice architecture"
# Erstellt Architektur-Diagramme und Dokumentation
```

### code/coder - Code-Entwicklung

```
bash

npx claude-flow@alpha sparc run code "implement user authentication"
# Entwickelt produktionsfertigen Code
```

### tdd - Test-Driven Development

```
bash
```

```
npx claude-flow@alpha sparcs run tdd "create payment system tests"
```

```
# Erstellt Tests vor Implementierung
```

## security-review - Sicherheitsanalyse

```
bash
```

```
npx claude-flow@alpha sparcs run security-review "audit authentication flow"
```

```
# Führt Sicherheitsüberprüfung durch
```

## integration - System-Integration

```
bash
```

```
npx claude-flow@alpha sparcs run integration "connect microservices"
```

```
# Integriert verschiedene Systemkomponenten
```

## devops - Deployment & CI/CD

```
bash
```

```
npx claude-flow@alpha sparcs run devops "setup kubernetes deployment"
```

```
# Konfiguriert Deployment-Pipeline
```

## Weitere Modi:

- **ask** - Recherche und Analyse
- **answer** - Detaillierte Antworten
- **create** - Content-Erstellung
- **optimize** - Code-Optimierung
- **refactor** - Code-Refactoring
- **document** - Dokumentation
- **test** - Test-Erstellung
- **debug** - Debugging
- **review** - Code-Review
- **plan** - Projektplanung

---

## Memory Management



## memory store - Daten speichern

bash

*# Einfacher Store*

```
npx claude-flow@alpha memory store <key> "value"
```

*# Mit Namespace*

```
npx claude-flow@alpha memory store requirements "User auth with JWT" \  
--namespace project-x
```

*# Mit Tags*

```
npx claude-flow@alpha memory store architecture "Microservices" \  
--tags "design,backend"
```

*# Optionen:*

```
--namespace <name> # Memory-Namespace  
--tags <list>      # Komma-getrennte Tags  
--ttl <seconds>    # Time-to-live  
--compress         # Komprimierung aktivieren
```

## memory query - Daten abfragen

bash

*# Einfache Suche*

```
npx claude-flow@alpha memory query "authentication"
```

*# Erweiterte Suche*

```
npx claude-flow@alpha memory query \  
--filter "auth" \  
--recent \  
--limit 10
```

*# Optionen:*

```
--filter <pattern> # Suchmuster  
--recent          # Nur aktuelle Einträge  
--limit <n>       # Anzahl Ergebnisse  
--namespace <name> # Spezifischer Namespace  
--format <type>   # json|table|list
```

## memory export/import

bash

*# Export*

```
npx claude-flow@alpha memory export \  
--output backup.json \  
--filter "project-"
```

*# Import*

```
npx claude-flow@alpha memory import backup.json \  
--merge \  
--validate
```

*# Optionen:*

```
--output <file>    # Export-Datei  
--merge            # Mit existierenden Daten mergen  
--validate         # Daten validieren  
--overwrite        # Existierende überschreiben
```

## Task Management

### task create - Task erstellen

bash

*# Einfacher Task*

```
npx claude-flow@alpha task create research "AI trends analysis"
```

*# Mit Priorität und Deadline*

```
npx claude-flow@alpha task create development "Build API" \  
--priority high \  
--estimated-duration 4h \  
--deadline "2024-12-31"
```

*# Optionen:*

```
--priority        # low|medium|high|critical  
--estimated-duration # Zeitschätzung (1h, 2d, etc.)  
--deadline        # Deadline (ISO-Format)  
--assignee        # Agent-Zuweisung  
--dependencies     # Abhängige Tasks
```

### task workflow - Workflow ausführen

bash

*# Workflow aus Datei*

```
npx claude-flow@alpha task workflow workflow.json
```

*# Mit Validierung*

```
npx claude-flow@alpha task workflow pipeline.json \  
--validate \  
--dry-run
```

*# Optionen:*

```
--validate      # Workflow validieren  
--async         # Asynchrone Ausführung  
--watch         # Progress beobachten  
--checkpoint    # Checkpoints erstellen
```

## task monitor - Tasks überwachen

bash

*# Live-Monitoring*

```
npx claude-flow@alpha task monitor --follow
```

*# Spezifischer Task*

```
npx claude-flow@alpha task monitor <task-id> \  
--detailed \  
--metrics
```

*# Optionen:*

```
--follow      # Live-Updates  
--detailed    # Detaillierte Infos  
--metrics     # Performance-Metriken  
--export <file> # In Datei exportieren
```

## MCP Integration

### mcp setup - MCP konfigurieren

bash

*# Auto-Setup*

```
npx claude-flow@alpha mcp setup \  
--auto-permissions \  
--87-tools
```

*# Manuelles Setup*

```
npx claude-flow@alpha mcp add <server-name> \  
--command "npx" \  
--args "server-package"
```

*# Optionen:*

```
--auto-permissions  # Automatische Berechtigungen  
--87-tools          # Alle 87 Tools aktivieren  
--validate          # Konfiguration validieren
```

## mcp tools - Tools verwalten

bash

*# Tools auflisten*

```
npx claude-flow@alpha mcp tools
```

*# Tool-Info*

```
npx claude-flow@alpha mcp tools info <tool-name>
```

*# Tool aktivieren/deaktivieren*

```
npx claude-flow@alpha mcp tools enable <tool-name>
```

```
npx claude-flow@alpha mcp tools disable <tool-name>
```

## mcp server - Server-Management

bash

*# Server-Status*

```
npx claude-flow@alpha mcp status
```

*# Server starten/stoppen*

```
npx claude-flow@alpha mcp start <server-name>
```

```
npx claude-flow@alpha mcp stop <server-name>
```

*# Server-Logs*

```
npx claude-flow@alpha mcp logs <server-name> --tail 100
```

---

## Monitoring & Debugging

## monitor - System-Monitoring

bash

*# Basis-Monitoring*

`npx claude-flow@alpha monitor`

*# Mit Dashboard*

`npx claude-flow@alpha monitor \`  
`--dashboard \`  
`--real-time`

*# Erweiterte Optionen*

`npx claude-flow@alpha monitor \`  
`--metrics cpu,memory,network \`  
`--interval 5 \`  
`--alert-threshold 80`

*# Optionen:*

`--dashboard`      *# Web-Dashboard*  
`--real-time`      *# Echtzeit-Updates*  
`--metrics <list>`   *# Spezifische Metriken*  
`--interval <sec>`   *# Update-Intervall*  
`--alert-threshold`   *# Alert-Schwellwert (%)*  
`--export <file>`   *# Metriken exportieren*

## logs - Log-Verwaltung

bash

*# Logs anzeigen*

`npx claude-flow@alpha logs`

*# Live-Logs*

`npx claude-flow@alpha logs --follow --tail 50`

*# Nach Level filtern*

`npx claude-flow@alpha logs --level error`

*# Optionen:*

`--follow, -f`      *# Live-Updates*  
`--tail <n>`      *# Letzte n Zeilen*  
`--level <level>`   *# Log-Level-Filter*  
`--since <time>`   *# Logs seit Zeitpunkt*  
`--grep <pattern>`   *# Pattern-Suche*  
`--json`      *# JSON-Format*

## debug - Debug-Modus

```
bash

# Debug-Session starten
npx claude-flow@alpha debug

# Spezifischen Agent debuggen
npx claude-flow@alpha debug agent <agent-id>

# Memory-Debug
npx claude-flow@alpha debug memory --trace

# Optionen:
--trace          # Trace-Level
--breakpoints    # Breakpoints setzen
--step           # Step-by-Step
--inspect        # Inspector aktivieren
```

---

## Konfiguration

### config - Konfigurationsverwaltung

```
bash

# Konfiguration anzeigen
npx claude-flow@alpha config show

# Wert abrufen
npx claude-flow@alpha config get orchestrator.maxConcurrentAgents

# Wert setzen
npx claude-flow@alpha config set orchestrator.maxConcurrentAgents 10

# Konfiguration initialisieren
npx claude-flow@alpha config init config.json

# Validieren
npx claude-flow@alpha config validate config.json

# Optionen:
--file <path>    # Config-Datei
--merge          # Mit Default mergen
--export <file>  # Exportieren
--reset          # Auf Default zurücksetzen
```

## Konfigurationsdatei-Struktur

json

```
{
  "orchestrator": {
    "maxConcurrentAgents": 5,
    "defaultTimeout": 300,
    "retryAttempts": 3,
    "memoryLimit": "2GB"
  },
  "terminal": {
    "maxSessions": 10,
    "sessionTimeout": 600,
    "recycleThreshold": 0.8
  },
  "memory": {
    "dbPath": ".swarm/memory.db",
    "cacheSize": "100MB",
    "compressionLevel": "medium"
  },
  "mcp": {
    "servers": {
      "filesystem": {
        "command": "npx",
        "args": ["-y", "@modelcontextprotocol/server-filesystem"]
      }
    }
  },
  "hooks": {
    "preToolUse": "./hooks/pre-tool.js",
    "postToolUse": "./hooks/post-tool.js"
  }
}
```

## Umgebungsvariablen

bash

*# API-Schlüssel*

```
export ANTHROPIC_API_KEY="your-api-key"
```

*# Claude-Flow-Konfiguration*

```
export CLAUDE_FLOW_CONFIG_DIR=~/.claude-flow"
```

```
export CLAUDE_FLOW_LOG_LEVEL="debug"
```

```
export CLAUDE_FLOW_MAX_AGENTS="10"
```

```
export CLAUDE_FLOW_MEMORY_PATH=".swarm/memory.db"
```

*# Performance*

```
export CLAUDE_FLOW_PARALLEL_EXECUTION="true"
```

```
export CLAUDE_FLOW_MAX_WORKERS="8"
```

```
export CLAUDE_FLOW_CACHE_SIZE="500MB"
```

*# Timeouts*

```
export CLAUDE_FLOW_DEFAULT_TIMEOUT="300"
```

```
export CLAUDE_FLOW_BASH_TIMEOUT="600"
```

```
export CLAUDE_FLOW_MCP_TIMEOUT="120"
```

*# Debug*

```
export CLAUDE_FLOW_DEBUG="true"
```

```
export CLAUDE_FLOW_TRACE="false"
```

```
export CLAUDE_FLOW_VERBOSE="true"
```

*# Telemetrie*

```
export CLAUDE_FLOW_DISABLE_TELEMETRY="true"
```

```
export CLAUDE_FLOW_DISABLE_ANALYTICS="true"
```

## Befehlsreihenfolge und Workflow-Regeln

### Kritische Regeln für die richtige Befehlsreihenfolge

Die korrekte Reihenfolge der Befehle ist entscheidend für den erfolgreichen Betrieb von Claude-Flow. Falsche Reihenfolgen können zu Fehlern, hängenden Prozessen oder inkonsistenten Zuständen führen.

#### ● **REGEL 1: Initialisierung IMMER zuerst**

```
bash
```



*# IMMER als erstes ausführen bei neuem Projekt:*

```
npx claude-flow@alpha init --force
```

*# Oder mit erweiterten Features:*

```
npx claude-flow@alpha init --force --sparc --hive-mind --neural-enhanced
```

*# NIEMALS andere Befehle vor init ausführen!*

## ● **REGEL 2: Claude Code Authentifizierung**

```
bash
```

*# Vor dem ersten Swarm/Hive-Mind MUSS Claude authentifiziert sein:*

```
claude --dangerously-skip-permissions
```

*# Verifizierung:*

```
which claude # Sollte Pfad zeigen
```

```
claude --version # Sollte Version zeigen
```

## ● **REGEL 3: Start vor Swarm/Hive-Mind (optional aber empfohlen)**

```
bash
```

*# Orchestrator starten für bessere Performance:*

```
npx claude-flow@alpha start --daemon
```

*# Dann erst:*

```
npx claude-flow@alpha swarm "task"
```

*# oder*

```
npx claude-flow@alpha hive-mind spawn "task"
```

## **Workflow-Patterns und Befehlssequenzen**

### **Pattern 1: Quick Task (Einmalige Aufgabe)**

```
bash
```

*# Sequenz für schnelle, einmalige Aufgaben*

```
1. npx claude-flow@alpha init --force
```

```
2. npx claude-flow@alpha swarm "Build REST API" --claude
```

```
3. npx claude-flow@alpha memory query --recent # Optional: Ergebnisse prüfen
```

**Verwendung:** Für isolierte Aufgaben ohne Fortsetzung

### **Pattern 2: Feature Development (Mit Session-Kontinuität)**

bash

*# Sequenz für Feature-Entwicklung mit mehreren Sessions*

1. npx claude-flow@alpha init --force
2. npx claude-flow@alpha hive-mind spawn "Implement user auth" --claude
3. npx claude-flow@alpha memory store project-context "OAuth2 with JWT"
4. npx claude-flow@alpha swarm "Add password reset" --continue-session
5. npx claude-flow@alpha hive-mind status *# Session prüfen*
6. npx claude-flow@alpha memory query "authentication" --recent

**Verwendung:** Für kontinuierliche Entwicklung an einem Feature

### Pattern 3: Complex Project (Enterprise-Level)

bash

*# Sequenz für komplexe Projekte mit mehreren Teams*

1. npx claude-flow@alpha init --force --hive-mind --neural-enhanced
2. npx claude-flow@alpha mcp setup --auto-permissions --87-tools
3. npx claude-flow@alpha hive-mind init --topology hierarchical --agents 12
4. npx claude-flow@alpha neural enable --pattern coordination
5. npx claude-flow@alpha hive-mind spawn "Build microservices" --agents 8 --claude
6. npx claude-flow@alpha monitor --dashboard *# In separatem Terminal*
7. npx claude-flow@alpha hive-mind status
8. npx claude-flow@alpha swarm "Add service X" --continue-session

**Verwendung:** Für große, komplexe Projekte mit mehreren Agenten

### Pattern 4: Session Resume (Arbeit fortsetzen)

bash

*# Sequenz zum Fortsetzen vorheriger Arbeit*

1. npx claude-flow@alpha hive-mind status *# Aktive Sessions prüfen*
2. npx claude-flow@alpha hive-mind sessions *# Alle Sessions anzeigen*
3. npx claude-flow@alpha hive-mind resume session-xxxxx *# Spezifische Session*
4. npx claude-flow@alpha memory query --recent --limit 10 *# Kontext laden*
5. npx claude-flow@alpha swarm "Continue feature" --continue-session

**Verwendung:** Nach Unterbrechung oder am nächsten Tag

---

## Entscheidungsbaum: Swarm vs Hive-Mind

Aufgabe zu erledigen?

└─ Einmalig/Isoliert?

| └─ JA → swarm "task"

| └─ Beispiel: npx claude-flow@alpha swarm "Fix bug #123" --claude

└─ Kontinuierlich/Komplex?

└─ JA → hive-mind spawn "task"

└─ Braucht Persistenz? → --namespace projekt-x

└─ Mehrere Agenten? → --agents 8

└─ Beispiel: npx claude-flow@alpha hive-mind spawn "Build app" --agents 8 --claude

## Wann SWARM verwenden:

- ☒ Schnelle, einmalige Aufgaben
- ☒ Prototyping
- ☒ Isolierte Features
- ☒ Wenn keine Session-Kontinuität benötigt wird

bash

npx claude-flow@alpha swarm "Create login form" --claude

## Wann HIVE-MIND verwenden:

- ☒ Komplexe, mehrstufige Projekte
- ☒ Wenn Session-Persistenz wichtig ist
- ☒ Bei Bedarf an mehreren spezialisierten Agenten
- ☒ Für kontinuierliche Entwicklung

bash

npx claude-flow@alpha hive-mind spawn "Develop e-commerce platform" --agents 10 --claude

## Häufige Fehler und deren Vermeidung

### ✗ FEHLER 1: Befehle ohne Initialisierung

bash

# FALSCH:

```
npx claude-flow@alpha swarm "task" # Fehler: Keine Konfiguration
```

# RICHTIG:

```
npx claude-flow@alpha init --force  
npx claude-flow@alpha swarm "task"
```

## ✗ FEHLER 2: Hive-Mind ohne Setup

bash

# FALSCH:

```
npx claude-flow@alpha hive-mind spawn "task" # Fehler: Hive nicht initialisiert
```

# RICHTIG:

```
npx claude-flow@alpha init --force --hive-mind  
npx claude-flow@alpha hive-mind init  
npx claude-flow@alpha hive-mind spawn "task"
```

## ✗ FEHLER 3: Memory-Abfrage vor Erstellung

bash

# FALSCH:

```
npx claude-flow@alpha memory query "data" # Fehler: Keine Daten
```

# RICHTIG:

```
npx claude-flow@alpha init --force  
npx claude-flow@alpha memory store key "value"  
npx claude-flow@alpha memory query "data"
```

## ✗ FEHLER 4: Parallele Swarms ohne Namespace

bash

# FALSCH:

```
npx claude-flow@alpha swarm "Task 1" &  
npx claude-flow@alpha swarm "Task 2" & # Konflikt!
```

# RICHTIG:

```
npx claude-flow@alpha swarm "Task 1" --namespace task1 &  
npx claude-flow@alpha swarm "Task 2" --namespace task2 &
```

---

## Best Practice Workflows

## Optimaler Development Workflow

```
bash

#!/bin/bash
# optimal-workflow.sh

# 1. Einmal pro Projekt
npx claude-flow@alpha init --force --sparc

# 2. Einmal pro Session
claude --dangerously-skip-permissions

# 3. Hauptentwicklung
npx claude-flow@alpha hive-mind spawn "Main development" --claude

# 4. Parallel-Tasks (in separaten Terminals)
npx claude-flow@alpha swarm "Write tests" --namespace tests
npx claude-flow@alpha swarm "Update docs" --namespace docs

# 5. Monitoring (separates Terminal)
npx claude-flow@alpha monitor --dashboard

# 6. Session beenden
npx claude-flow@alpha hive-mind status
npx claude-flow@alpha memory export --output ./session-backup.json
```

## Quick Setup mit Aliases

```
bash

# ~/.zshrc oder ~/.bashrc
alias flow='npx --y claude-flow@alpha init --force'
alias yolo='claude --dangerously-skip-permissions'
alias flowstart='npx claude-flow@alpha start --daemon'
alias flowswarm='npx claude-flow@alpha swarm'
alias flowhive='npx claude-flow@alpha hive-mind spawn'
alias flowstatus='npx claude-flow@alpha hive-mind status'

# Verwendung:
flow      # Initialisierung
yolo      # Claude auth
flowstart # Daemon starten
flowswarm "Build feature" --claude # Swarm starten
```

## Session Management Workflow

```
bash
```

```
# Morgens: Vorherige Arbeit fortsetzen
```

```
npx claude-flow@alpha hive-mind status
```

```
npx claude-flow@alpha memory query --recent --limit 5
```

```
npx claude-flow@alpha hive-mind resume session-xxxxx
```

```
# Während der Arbeit: Kontext speichern
```

```
npx claude-flow@alpha memory store checkpoint-1 "Completed auth module"
```

```
npx claude-flow@alpha memory store checkpoint-2 "API endpoints ready"
```

```
# Abends: Session sichern
```

```
npx claude-flow@alpha hive-mind status > session-status.txt
```

```
npx claude-flow@alpha memory export --output ./backups/$(date +%Y%m%d).json
```

---

## Fortgeschrittene Sequenzen

### Multi-Stage Pipeline

```
bash
```

```
# Stage 1: Research & Planning
```

```
npx claude-flow@alpha init --force --neural-enhanced
```

```
npx claude-flow@alpha neural train --pattern research
```

```
npx claude-flow@alpha swarm "Research best practices" --strategy research
```

```
# Stage 2: Architecture
```

```
npx claude-flow@alpha sparac run architect "Design system"
```

```
# Stage 3: Parallel Development
```

```
npx claude-flow@alpha hive-mind spawn "Development" --agents 10
```

```
parallel --jobs 3 << 'EOF'
```

```
npx claude-flow@alpha agent spawn coder --name "Backend Dev"
```

```
npx claude-flow@alpha agent spawn coder --name "Frontend Dev"
```

```
npx claude-flow@alpha agent spawn tester --name "QA Engineer"
```

```
EOF
```

```
# Stage 4: Integration
```

```
npx claude-flow@alpha swarm "Integrate services" --strategy sequential
```

```
# Stage 5: Testing & Deployment
```

```
npx claude-flow@alpha sparac run tdd "Complete test suite"
```

```
npx claude-flow@alpha sparac run devops "Deploy to production"
```

## Disaster Recovery Workflow

bash

*# Bei hängenden Prozessen:*

1. npx claude-flow@alpha hive-mind status *# Status prüfen*
2. npx claude-flow@alpha stop --force *# Stoppen*
3. npx claude-flow@alpha memory export --output emergency-backup.json
4. npx claude-flow@alpha init --force *# Neu initialisieren*
5. npx claude-flow@alpha memory import emergency-backup.json
6. npx claude-flow@alpha hive-mind spawn "Continue" --restore

---

## Zusammenfassung der Befehlsreihenfolge-Regeln

1. **IMMER** mit `init` beginnen
  2. **IMMER** Claude authentifizieren vor Swarm/Hive
  3. **NIE** mehrere init ohne --force ausführen
  4. **NIE** parallele Swarms ohne unterschiedliche Namespaces
  5. **IMMER** Status prüfen vor Resume
  6. **IMMER** Memory exportieren vor kritischen Änderungen
  7. **NIE** Hive-Mind ohne vorheriges Setup spawnen
  8. **IMMER** Monitor in separatem Terminal für lange Tasks
  9. **NIE** verschiedene Topologien mischen ohne Reset
  10. **IMMER** --continue-session für zusammenhängende Aufgaben
- 

## Beispiel-Workflows

### Vollständige Entwicklungs-Pipeline

bash

```
#!/bin/bash
```

```
# development-pipeline.sh
```

```
# 1. Initialisierung
```

```
npm run cflow@alpha init --sparc --force
```

```
# 2. System starten
```

```
npm run cflow@alpha start --daemon --ui --port 3000
```

```
# 3. Architektur erstellen
```

```
npm run cflow@alpha sparc run architect \  
  "Design scalable microservice architecture for e-commerce"
```

```
# 4. Parallele Entwicklung
```

```
npm run cflow@alpha swarm "Implement services" \  
  --strategy parallel \  
  --agents 5 \  
  --monitor << EOF
```

- User Service mit JWT Auth
- Product Catalog Service
- Shopping Cart Service
- Payment Processing Service
- Order Management Service

```
EOF
```

```
# 5. Tests erstellen
```

```
npm run cflow@alpha sparc run tdd "Create comprehensive test suite"
```

```
# 6. Integration
```

```
npm run cflow@alpha sparc run integration "Connect all services"
```

```
# 7. Security Review
```

```
npm run cflow@alpha sparc run security-review "Full security audit"
```

```
# 8. DevOps Setup
```

```
npm run cflow@alpha sparc run devops "Setup CI/CD with GitHub Actions"
```

```
# 9. Monitoring
```

```
npm run cflow@alpha monitor --dashboard
```

## Batch-Processing mit Parallelisierung

```
bash
```



```
#!/bin/bash
```

```
# batch-processing.sh
```

```
# Batch-Tool für parallele Ausführung
```

```
batchtool run --parallel \
```

```
"npx claude-flow@alpha sparc run architect 'design auth system'" \
```

```
"npx claude-flow@alpha sparc run code 'implement JWT tokens'" \
```

```
"npx claude-flow@alpha sparc run tdd 'auth test suite'" \
```

```
"npx claude-flow@alpha sparc run security-review 'auth vulnerabilities'" \
```

```
"npx claude-flow@alpha sparc run document 'API documentation'"
```

```
# Ergebnisse sammeln
```

```
npx claude-flow@alpha memory query --filter "auth" --recent
```

## Continuous Integration Workflow

```
bash
```

```
#!/bin/bash
```

```
# ci-workflow.sh
```

```
# Pre-commit Hook
```

```
npx claude-flow@alpha sparc run review "Check code quality" \
```

```
--headless \
```

```
--output-format json
```

```
# Test Suite
```

```
npx claude-flow@alpha sparc run test "Run all tests" \
```

```
--coverage \
```

```
--fail-fast
```

```
# Security Scan
```

```
npx claude-flow@alpha security scan ./src \
```

```
--severity high \
```

```
--format sarif
```

```
# Deploy
```

```
if [ "$BRANCH" = "main" ]; then
```

```
npx claude-flow@alpha deploy create "v${VERSION}" \
```

```
--strategy blue-green \
```

```
--auto-rollback
```

```
fi
```

## Research & Analysis Workflow

```
bash
```

```
#!/bin/bash
```

```
# research-workflow.sh
```

```
# 1. Research Phase
```

```
npx claude-flow@alpha swarm "Research AI safety in autonomous systems" \  
--strategy research \  
--neural-patterns enabled \  
--memory-compression high \  
--agents 10
```

```
# 2. Analyse
```

```
npx claude-flow@alpha cognitive analyze \  
--target research-results \  
--depth comprehensive
```

```
# 3. Report Generation
```

```
npx claude-flow@alpha sparx run document \  
"Create executive summary and technical report"
```

```
# 4. Export Results
```

```
npx claude-flow@alpha memory export \  
--filter "research-*" \  
--output research-findings.json
```

---

## Swarm-Presets und Konfigurationsdateien

### Dateistruktur und Speicherorte

Claude-Flow verwendet verschiedene Konfigurationsdateien für Swarm-Presets:

```
bash
```

*# Projekt-Struktur nach Initialisierung*

```
your-project/
├── .claude/
│   ├── settings.json      # Claude-Flow Hauptkonfiguration
│   ├── commands/         # Custom Slash-Commands
│   └── *.json             # Weitere Konfigurationen
├── .hive-mind/
│   ├── config.json       # Hive-Mind-Konfiguration
│   └── sessions.db       # SQLite Session-Daten
├── .swarm/
│   ├── memory.db         # SQLite Memory-Datenbank
│   └── presets/          # Swarm-Preset-Definitionen
├── swarms/               # Swarm-Konfigurationen
│   ├── development/      # Entwicklungs-Swarms
│   ├── testing/          # Test-Swarms
│   └── production/       # Produktions-Swarms
├── workflows/
│   ├── ci-cd/            # CI/CD-Workflows
│   └── deployment/       # Deployment-Workflows
└── examples/
    └── 01-configurations/ # Beispiel-Konfigurationen
```

## Globale Konfigurationspfade

bash

*# Standard-Pfade für globale Konfigurationen*

```
~/ .claude-flow/
├── config.json      # Globale Konfiguration
├── presets/         # Globale Presets
│   ├── python-development.json
│   ├── javascript-fullstack.json
│   └── enterprise-team.json
└── templates/       # Wiederverwendbare Templates
```

## JSON-Struktur für Swarm-Presets

### Beispiel 1: Python Development Preset

**Datei:** `~/ .claude-flow/presets/python-development.json` oder `./swarms/development/python-development.json`

json

```
{
  "version": "2.0.0",
  "name": "Python Development Swarm",
  "description": "Complete Python development team with TDD and DevOps",
  "metadata": {
    "author": "claude-flow",
    "created": "2025-08-18",
    "tags": ["python", "development", "tdd", "api"],
    "difficulty": "intermediate"
  },
  "orchestrator": {
    "topology": "hierarchical",
    "maxConcurrentAgents": 8,
    "coordinationLevel": "high",
    "memoryNamespace": "python-project",
    "defaultTimeout": 600,
    "retryAttempts": 3
  },
  "agents": [
    {
      "id": "queen",
      "type": "coordinator",
      "name": "Python Project Manager",
      "role": "queen",
      "priority": 10,
      "capabilities": ["planning", "delegation", "monitoring"],
      "prompt": "You are the lead coordinator for a Python development project using UV, SQLite, SQLAlchemy, and pyte",
      "memory": {
        "size": "500MB",
        "persistent": true
      }
    },
    {
      "id": "architect",
      "type": "architect",
      "name": "Python System Architect",
      "role": "worker",
      "priority": 9,
      "capabilities": ["design", "architecture", "database-modeling"],
      "prompt": "Design scalable Python architectures using modern patterns. Focus on SQLAlchemy ORM design, API st",
      "tools": ["Read", "Write", "WebSearch"],
      "specialization": {
        "frameworks": ["FastAPI", "Flask", "Django"],
        "databases": ["SQLite", "PostgreSQL", "Redis"],
        "patterns": ["Repository", "CQRS", "Event Sourcing"]
      }
    }
  ]
}
```

```
},
{
  "id": "backend-1",
  "type": "coder",
  "name": "Senior Python Developer",
  "role": "worker",
  "priority": 8,
  "capabilities": ["implementation", "api-development", "database"],
  "prompt": "Implement Python backend services using UV for package management, SQLAlchemy for ORM, and follo",
  "tools": ["Edit", "Write", "Bash", "Read"],
  "directory": "./src/backend",
  "specialization": {
    "expertise": ["async-programming", "orm", "rest-api"],
    "libraries": ["sqlalchemy", "pydantic", "asyncio", "uvicorn"]
  }
},
{
  "id": "backend-2",
  "type": "coder",
  "name": "Python API Developer",
  "role": "worker",
  "priority": 7,
  "capabilities": ["api-endpoints", "validation", "serialization"],
  "prompt": "Focus on API endpoint implementation, request validation with Pydantic, and proper error handling.",
  "tools": ["Edit", "Write", "Read"],
  "directory": "./src/api"
},
{
  "id": "tdd-specialist",
  "type": "tester",
  "name": "Python TDD Specialist",
  "role": "worker",
  "priority": 8,
  "capabilities": ["test-driven-development", "pytest", "coverage"],
  "prompt": "Write comprehensive tests using pytest before implementation. Ensure 90%+ code coverage. Use fixture",
  "tools": ["Write", "Edit", "Bash"],
  "directory": "./tests",
  "config": {
    "testFramework": "pytest",
    "coverageThreshold": 90,
    "testPatterns": ["unit", "integration", "e2e"],
    "mockLibrary": "pytest-mock"
  }
},
{
  "id": "data-engineer",
  "type": "specialist",
```

```
"name": "Python Data Engineer",
"role": "worker",
"priority": 7,
"capabilities": ["etl", "data-modeling", "migrations"],
"prompt": "Handle database migrations with Alembic, optimize SQLAlchemy queries, and manage data pipelines.",
"tools": ["Edit", "Write", "Bash"],
"directory": "./src/data",
"specialization": {
  "tools": ["alembic", "pandas", "numpy"],
  "focus": ["performance", "indexing", "query-optimization"]
},
{
  "id": "devops",
  "type": "devops",
  "name": "Python DevOps Engineer",
  "role": "worker",
  "priority": 6,
  "capabilities": ["ci-cd", "docker", "deployment"],
  "prompt": "Setup CI/CD with GitHub Actions, create Dockerfile for Python app, manage UV dependencies, and conf
  "tools": ["Write", "Bash"],
  "directory": "./devops",
  "config": {
    "containerization": "docker",
    "ci": "github-actions",
    "packageManager": "uv",
    "deployment": ["kubernetes", "aws-lambda"]
  },
  "id": "security",
  "type": "security",
  "name": "Python Security Analyst",
  "role": "worker",
  "priority": 7,
  "capabilities": ["security-audit", "vulnerability-scan", "authentication"],
  "prompt": "Implement secure authentication with JWT, audit dependencies for vulnerabilities, and ensure OWASP c
  "tools": ["Read", "Edit", "WebSearch"],
  "specialization": {
    "focus": ["jwt", "oauth2", "input-validation", "sql-injection"],
    "tools": ["bandit", "safety", "pip-audit"]
  },
  "id": "documenter",
  "type": "technical-writer",
  "name": "Python Documentation Specialist",
```

```
"role": "worker",
"priority": 5,
"capabilities": ["documentation", "api-docs", "docstrings"],
"prompt": "Create comprehensive documentation with Sphinx, write clear docstrings, and maintain API documentat
"tools": ["Write", "Read"],
"directory": "./docs",
"config": {
  "docTool": "sphinx",
  "apiDoc": "openapi",
  "style": "google-docstrings"
}
},
],
"workflow": {
  "phases": [
    {
      "name": "planning",
      "agents": ["queen", "architect"],
      "duration": "10m",
      "outputs": ["architecture.md", "requirements.txt"]
    },
    {
      "name": "implementation",
      "agents": ["backend-1", "backend-2", "data-engineer"],
      "parallel": true,
      "duration": "30m"
    },
    {
      "name": "testing",
      "agents": ["tdd-specialist"],
      "requires": ["implementation"],
      "duration": "15m"
    },
    {
      "name": "security-review",
      "agents": ["security"],
      "requires": ["testing"],
      "duration": "10m"
    },
    {
      "name": "deployment",
      "agents": ["devops"],
      "requires": ["security-review"],
      "duration": "10m"
    },
    {
      "name": "documentation",
```

```
"agents": ["documenter"],
"parallel": true,
"duration": "15m"
}
],
},
"memory": {
  "persistent": true,
  "sharedNamespaces": ["project-context", "api-design", "test-results"],
  "compression": "high",
  "maxSize": "1GB"
},
"hooks": {
  "preToolUse": "./hooks/python-pre-tool.sh",
  "postToolUse": "./hooks/python-post-tool.sh",
  "onError": "./hooks/error-handler.py",
  "onComplete": "./hooks/completion-notify.sh"
},
"environment": {
  "PYTHON_VERSION": "3.11",
  "UV_SYSTEM_PYTHON": "true",
  "SQLALCHEMY_WARN_20": "1",
  "PYTEST_ADDOPTS": "--cov=src --cov-report=term-missing"
},
"tools": {
  "allowed": ["Bash", "Edit", "Write", "Read", "WebSearch", "WebFetch"],
  "custom": [
    {
      "name": "uv-install",
      "command": "uv pip install",
      "description": "Install Python packages with UV"
    },
    {
      "name": "pytest-run",
      "command": "pytest",
      "args": ["--verbose", "--cov"],
      "description": "Run pytest with coverage"
    },
    {
      "name": "alembic-migrate",
      "command": "alembic",
      "args": ["upgrade", "head"],
      "description": "Run database migrations"
    }
  ]
},
"mcp": {
```



```
"servers": {
  "filesystem": {
    "command": "npx",
    "args": ["-y", "@modelcontextprotocol/server-filesystem", "./src"]
  },
  "memory": {
    "command": "npx",
    "args": ["-y", "@modelcontextprotocol/server-memory"]
  },
  "github": {
    "command": "npx",
    "args": ["-y", "@modelcontextprotocol/server-github"],
    "env": {
      "GITHUB_TOKEN": "${GITHUB_TOKEN}"
    }
  }
},
"monitoring": {
  "enabled": true,
  "metrics": ["cpu", "memory", "task-completion", "error-rate"],
  "dashboard": true,
  "alerts": {
    "errorThreshold": 5,
    "timeoutThreshold": 900,
    "memoryThreshold": "80%"
  }
},
"optimization": {
  "parallelExecution": true,
  "batchSize": 5,
  "caching": true,
  "autoScale": {
    "enabled": true,
    "minAgents": 3,
    "maxAgents": 10,
    "scaleUpThreshold": 0.8,
    "scaleDownThreshold": 0.3
  }
}
}
```

## Beispiel 2: Minimales Swarm-Preset

Datei: `./swarms/minimal-swarm.json`

json

```
{
  "version": "2.0.0",
  "name": "Minimal Development Swarm",
  "agents": [
    {
      "type": "coordinator",
      "name": "Lead"
    },
    {
      "type": "coder",
      "name": "Developer"
    },
    {
      "type": "tester",
      "name": "Tester"
    }
  ],
  "orchestrator": {
    "topology": "simple",
    "maxConcurrentAgents": 3
  }
}
```

### Beispiel 3: Enterprise Team Preset

Datei:

json

```
{
  "version": "2.0.0",
  "name": "Enterprise Development Team",
  "description": "Full enterprise team with 12 specialized agents",
  "orchestrator": {
    "topology": "hierarchical",
    "maxConcurrentAgents": 12,
    "coordinationLevel": "enterprise"
  },
  "agents": [
    {
      "id": "cto",
      "type": "coordinator",
      "name": "Chief Technical Officer",
      "role": "queen",
      "priority": 10
    },
    {
      "id": "architect-lead",
      "type": "architect",
      "name": "Lead Architect",
      "role": "lead",
      "priority": 9,
      "connections": ["frontend-lead", "backend-lead", "data-lead"]
    },
    {
      "id": "frontend-lead",
      "type": "coordinator",
      "name": "Frontend Team Lead",
      "role": "lead",
      "priority": 8,
      "connections": ["react-dev", "ui-designer"]
    },
    {
      "id": "backend-lead",
      "type": "coordinator",
      "name": "Backend Team Lead",
      "role": "lead",
      "priority": 8,
      "connections": ["api-dev", "database-dev"]
    },
    {
      "id": "data-lead",
      "type": "coordinator",
      "name": "Data Team Lead",
      "role": "lead",

```

```
    "priority": 8,
    "connections": ["data-engineer", "ml-engineer"]
  },
  {
    "id": "security-lead",
    "type": "security",
    "name": "Security Team Lead",
    "role": "lead",
    "priority": 9
  },
  {
    "id": "devops-lead",
    "type": "devops",
    "name": "DevOps Team Lead",
    "role": "lead",
    "priority": 8
  },
  {
    "id": "qa-lead",
    "type": "tester",
    "name": "QA Team Lead",
    "role": "lead",
    "priority": 8
  }
]
```

## Preset-Verwendung in Claude-Flow

### Laden von Presets

```
bash
```

*# Preset direkt verwenden*

```
npx claude-flow@alpha swarm create --preset python-development
```

*# Preset aus spezifischer Datei laden*

```
npx claude-flow@alpha swarm create --config ./swarms/development/python-development.json
```

*# Preset mit Überschreibungen*

```
npx claude-flow@alpha swarm create \  
  --preset python-development \  
  --max-agents 5 \  
  --timeout 30
```

*# Mehrere Presets kombinieren*

```
npx claude-flow@alpha swarm create \  
  --config base-config.json \  
  --merge python-agents.json \  
  --merge security-config.json
```

## Preset-Verwaltung

bash

*# Verfügbare Presets anzeigen*

```
npx claude-flow@alpha preset list
```

*# Preset-Details anzeigen*

```
npx claude-flow@alpha preset show python-development
```

*# Neues Preset erstellen*

```
npx claude-flow@alpha preset create my-preset \  
  --template python-development \  
  --output ./swarms/my-preset.json
```

*# Preset validieren*

```
npx claude-flow@alpha preset validate ./swarms/my-preset.json
```

*# Preset installieren (global)*

```
npx claude-flow@alpha preset install ./my-preset.json --global
```

## Integration in Workflows

bash

```
#!/bin/bash
```

```
# workflow-with-preset.sh
```

```
# 1. Preset laden und Swarm initialisieren
```

```
npx claude-flow@alpha swarm init --preset python-development
```

```
# 2. Memory mit Projekt-Kontext füllen
```

```
npx claude-flow@alpha memory store project-type "FastAPI REST API"
```

```
npx claude-flow@alpha memory store requirements "OAuth2, PostgreSQL, Redis Cache"
```

```
# 3. Swarm mit spezifischer Aufgabe starten
```

```
npx claude-flow@alpha swarm execute \
```

```
--preset python-development \
```

```
--task "Build user authentication service with JWT" \
```

```
--monitor
```

```
# 4. Ergebnisse exportieren
```

```
npx claude-flow@alpha memory export \
```

```
--namespace python-project \
```

```
--output ./results/auth-service.json
```

## Wichtige Konfigurationsfelder

### Pflichtfelder

- `version`: String - Version der Konfiguration
- `name`: String - Name des Presets
- `agents`: Array - Liste der Agenten

### Orchestrator-Optionen

- `topology`: `"hierarchical"` | `"mesh"` | `"ring"` | `"star"` | `"simple"`
- `maxConcurrentAgents`: Number (1-20)
- `coordinationLevel`: `"low"` | `"medium"` | `"high"` | `"enterprise"`
- `memoryNamespace`: String - Namespace für gemeinsamen Speicher
- `strategy`: `"parallel"` | `"sequential"` | `"adaptive"` | `"research"` | `"development"`

### Agent-Felder

- `id`: String - Eindeutige ID
- `type`: `"coordinator"` | `"architect"` | `"coder"` | `"tester"` | `"security"` | `"devops"` | `"specialist"` | `"researcher"` | `"analyst"`
- `name`: String - Anzeigename

- `role`: `"queen"` | `"lead"` | `"worker"`
- `priority`: Number (1-10)
- `capabilities`: Array - Fähigkeiten
- `prompt`: String - System-Prompt
- `tools`: Array - Erlaubte Tools
- `directory`: String - Arbeitsverzeichnis
- `connections`: Array - Verbundene Agenten
- `model`: String - Modell (`"opus"` | `"sonnet"` | `"haiku"`)

## Memory-Konfiguration

- `persistent`: Boolean
- `sharedNamespaces`: Array
- `compression`: `"low"` | `"medium"` | `"high"`
- `maxSize`: String (z.B. "1GB")

## Hook-System

- `preToolUse`: String - Pfad zum Pre-Hook-Script
- `postToolUse`: String - Post-Hook-Script
- `onError`: String - Error-Handler
- `onComplete`: String - Completion-Handler

## MCP-Server

- Beliebige MCP-Server-Konfigurationen
- Standard: filesystem, memory, github
- Custom Server möglich

## Best Practices für Presets

### 1. Strukturierung

- Verwende klare, beschreibende Namen
- Gruppiere Agenten nach Funktion
- Definiere klare Hierarchien

### 2. Wiederverwendbarkeit

- Erstelle modulare Presets

- Nutze Template-Vererbung
- Dokumentiere Abhängigkeiten

### 3. Performance

- Limitiere concurrent Agents basierend auf System
- Nutze Caching und Memory-Kompression
- Setze sinnvolle Timeouts

### 4. Versionierung

- Versioniere Presets mit Git
- Nutze semantische Versionierung
- Dokumentiere Breaking Changes

---

## Fehlerbehebung

### Häufige Probleme und Lösungen

#### Installation schlägt fehl

```
bash

# Cache löschen
npm cache clean --force

# Mit Legacy-Peer-Deps
npm install -g claude-flow@alpha --legacy-peer-deps

# Alternative: Yarn verwenden
yarn global add claude-flow@alpha
```

#### SQLite-Fehler (Windows)

```
bash

# In-Memory-Storage verwenden
export CLAUDE_FLOW_MEMORY_TYPE="inmemory"

# Oder alternativen Pfad setzen
export CLAUDE_FLOW_MEMORY_PATH="C:/Users/$USER/.claude-flow/memory.db"
```

#### Timeout-Probleme

```
bash
```



*# Timeouts erhöhen*

```
npx claude-flow@alpha config set orchestrator.defaultTimeout 600
```

```
export CLAUDE_FLOW_BASH_TIMEOUT="1200"
```

## MCP-Verbindungsprobleme

bash

*# Debug-Modus aktivieren*

```
npx claude-flow@alpha --mcp-debug
```

*# MCP-Server manuell starten*

```
npx claude-flow@alpha mcp start filesystem --verbose
```

*# Logs prüfen*

```
npx claude-flow@alpha mcp logs --tail 100
```

## Performance-Probleme

bash

*# Ressourcen prüfen*

```
npx claude-flow@alpha system resources
```

*# Limits anpassen*

```
npx claude-flow@alpha config set orchestrator.maxConcurrentAgents 3
```

```
npx claude-flow@alpha config set terminal.maxSessions 5
```

*# Cache löschen*

```
npx claude-flow@alpha cache clear --all
```

## Debug-Befehle

bash

*# System-Health-Check*

`npx claude-flow@alpha doctor`

*# Vollständiger System-Report*

`npx claude-flow@alpha diagnostic --full > diagnostic.log`

*# Test-Suite ausführen*

`npx claude-flow@alpha test --self-test`

*# Reset auf Werkseinstellungen*

`npx claude-flow@alpha reset --factory --confirm`

## Erweiterte Optionen (Experimentell)

Diese Optionen sind möglicherweise in v86 verfügbar, aber nicht offiziell dokumentiert:

`bash`

*# Worker-Output anzeigen (experimentell)*

`--show-worker-output`      *# Zeigt Output aller Worker-Agenten*

*# Status-Anzeige (experimentell)*

`--show-status`              *# Kontinuierliche Status-Updates*

*# Farb-Optionen (experimentell)*

`--color`                      *# Farbausgabe erzwingen*

`--no-color`                  *# Farbausgabe deaktivieren*

*# Erweiterte Debug-Optionen (experimentell)*

`--debug-network`            *# Netzwerk-Debug*

`--debug-memory`            *# Memory-Debug*

`--debug-agents`            *# Agent-Debug*

*# Performance-Profiling (experimentell)*

`--profile-cpu`              *# CPU-Profiling*

`--profile-memory`          *# Memory-Profiling*

`--profile-io`               *# I/O-Profiling*

## Best Practices

### 1. Projekt-Setup

`bash`

*# Optimale Initialisierung*

```
npx claude-flow@alpha init --sparc --force
```

```
echo "ANTHROPIC_API_KEY=your-key" > .env
```

## 2. Resource Management

bash

*# Ressourcen überwachen*

```
npx claude-flow@alpha monitor --dashboard &
```

*# Limits setzen*

```
npx claude-flow@alpha config set orchestrator.maxConcurrentAgents 5
```

```
npx claude-flow@alpha config set memory.cacheSize "200MB"
```

## 3. Error Handling

bash

*# Mit Retry-Logic*

```
npx claude-flow@alpha swarm "task" \
```

```
--max-retries 3 \
```

```
--retry-delay 5 \
```

```
--fallback-strategy sequential
```

## 4. Logging

bash

*# Strukturiertes Logging*

```
npx claude-flow@alpha start \
```

```
--log-level info \
```

```
--log-file ./logs/claude-flow.log \
```

```
--log-format json
```

## 5. Security

bash

*# Sichere Konfiguration*











```
chmod 600 ~/.claude-flow/config.json
```

```
export CLAUDE_FLOW_SECURE_MODE="true"
```

---

## Versionsinformationen

## v2.0.0-alpha.86 Features

-  Hive-Mind-Architektur
-  87 MCP-Tools
-  Neural Pattern Recognition
-  SQLite Memory System
-  SPARC-Integration (17 Modi)
-  Batch-Tool-Unterstützung
-  GitHub-Integration
-  Web-UI Dashboard
-  Erweiterte Hooks
-  Multi-Cloud-Support

## Bekannte Einschränkungen

- Alpha-Version - nicht für Produktion empfohlen
  - Windows: SQLite kann Probleme verursachen
  - Maximale Agenten: 20
  - Memory-Limit: 4GB
  - Timeout-Maximum: 3600 Sekunden
- 

## Support & Ressourcen

### Offizielle Ressourcen

- GitHub: <https://github.com/ruvnet/claude-flow>
- NPM: <https://www.npmjs.com/package/claude-flow>
- Dokumentation: <https://github.com/ruvnet/claude-flow/docs>

### Community

- Issues: GitHub Issues für Bug-Reports
- Discussions: GitHub Discussions für Fragen
- Wiki: GitHub Wiki für Anleitungen

### Hilfe erhalten

```
bash
```

# Inline-Hilfe

npx claude-flow@alpha --help

npx claude-flow@alpha <command> --help

# Interaktive Hilfe

npx claude-flow@alpha repl

> help

---

## Changelog Highlights (v86)

- **v2.0.0-alpha.86** (Aktuell)
  - Verbesserte Hive-Mind-Koordination
  - Erweiterte SPARC-Modi
  - 87 MCP-Tools Integration
  - Performance-Optimierungen
- **v2.0.0-alpha.80**
  - Web-UI Dashboard
  - Batch-Tool-Support
  - Neural Pattern Recognition
- **v2.0.0-alpha.70**
  - SQLite Memory System
  - GitHub-Integration
  - Hook-System

---

## Lizenz

MIT License - siehe LICENSE-Datei für Details

---

## Haftungsausschluss

⚠ **Alpha-Software:** Diese Version ist eine Alpha-Release und für Testing und Feedback gedacht. Verwendung in Produktionsumgebungen wird nicht empfohlen.

---

*Dokumentation erstellt für Claude-Flow v2.0.0-alpha.86 Letzte Aktualisierung: August 2025*