

# Claude-Flow@alpha Version 86 - Technische Dokumentation

## Inhaltsverzeichnis

1. Überblick
  2. Systemarchitektur
  3. Speicherorte und Verzeichnisstruktur
  4. Konfigurationsformat
  5. Beispielkonfigurationen
  6. Custom Agents erstellen
  7. Agent-Fähigkeiten definieren
  8. Presets und Teams
  9. Performance-Metriken
  10. Best Practices
- 

## Überblick

**Claude-flow@alpha (V86)** ist eine Enterprise-Grade AI-Orchestrierungsplattform mit revolutionärer Swarm-Intelligence-Koordination für Multi-Agent-Entwicklung.

## Kernmerkmale

- **64 spezialisierte AI-Agenten** in 16 Kategorien
- **87 MCP-Tools** für umfassende Funktionalität
- **27+ neuronale Modelle** mit WASM SIMD-Beschleunigung
- **SQLite-Datenbank** mit 12 spezialisierten Tabellen für persistente Memory
- **14 Lifecycle-Management-Hooks** für Automatisierung
- **Byzantine Fault Tolerance** für Enterprise-Zuverlässigkeit

## Performance-Charakteristiken

- **84.8% Erfolgsrate** bei SWE-Bench-Tests
  - **2.8-4.4x schnellere Ausführung** durch parallele Koordination
  - **32.3% Token-Reduktion** durch intelligente Optimierung
  - Unterstützung für bis zu **12 gleichzeitige Agenten**
- 

## Systemarchitektur

## Hauptkomponenten

1. **Orchestrator**: Zentrale Koordinationseinheit
2. **Agent System**: 64 spezialisierte Agenten
3. **Memory System**: Persistente SQLite-Datenbank
4. **Neural Networks**: 27 kognitive Modelle
5. **Hook System**: 14 Lifecycle-Hooks
6. **MCP Tools**: 87 spezialisierte Tools

## Topologie-Optionen

- `hierarchical`: Hierarchische Struktur mit Queen-Agent
  - `mesh`: Vollvernetzte Agenten
  - `ring`: Ringförmige Kommunikation
  - `star`: Sternförmige Struktur
  - `sequential`: Sequentielle Verarbeitung
- 

## Speicherorte und Verzeichnisstruktur

```
project-root/
├── .claude/
│   ├── settings.json      # Hauptkonfigurationsdatei
│   ├── settings.local.json # Lokale Überschreibungen (gitignored)
│   └── commands/          # Dokumentation für alle 87 Tools
│       ├── analysis/      # Analyse-Command-Dokumentation
│       ├── automation/    # Automatisierungs-Commands
│       ├── coordination/   # Koordinations-Commands
│       ├── github/        # GitHub-Integration-Commands
│       ├── hooks/         # Hook-Command-Dokumentation
│       ├── memory/        # Memory-Management-Commands
│       └── workflows/     # Workflow-Commands
├── .mcp.json              # MCP-Server-Konfiguration (Projektebene)
├── claude-flow.config.json # Claude-flow spezifische Einstellungen
├── CLAUDE.md              # Projektkontext für AI
├── .hive-mind/            # Hive-Session-Daten
│   ├── config.json        # Hive-Mind-Konfiguration
│   └── sessions/          # Aktive Session-Daten
├── .swarm/                # Swarm-Koordinationsdaten
│   └── memory.db          # SQLite Memory-Datenbank
├── agents/                # 64+ spezialisierte Agenten-Configs
│   ├── architect.json
│   ├── coder.json
│   ├── researcher.json
│   ├── tester.json
│   └── [weitere 60+ Agenten]
├── swarms/                # Swarm-spezifische Konfigurationen
│   ├── development/
│   ├── testing/
│   └── production/
└── workflows/             # Automatisierungs-Workflows
    ├── ci-cd/
    └── deployment/
```

## Konfigurationsformat

### Hauptkonfiguration (.claude/settings.json)

json

```

{
  "model": "sonnet",
  "permissions": {
    "allow": [
      "Bash(mkdir:*)",
      "Bash(npm:*)",
      "Write",
      "Edit",
      "MultiEdit"
    ],
    "deny": []
  },
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "npx claude-flow@alpha hooks pre-command --command \"{\\}\" --validate-safety true"
          }
        ]
      }
    ],
    "PostToolUse": [
      {
        "matcher": "Write|Edit|MultiEdit",
        "hooks": [
          {
            "type": "command",
            "command": "npx claude-flow@alpha hooks post-edit --file \"{\\}\" --memory-key \"swarm/{agent}/{step}\""
          }
        ]
      }
    ]
  },
  "env": {
    "BASH_DEFAULT_TIMEOUT_MS": "300000",
    "BASH_MAX_TIMEOUT_MS": "600000"
  }
}

```

## MCP-Server-Konfiguration (.mcp.json)

json

```
{  
  "mcpServers": {  
    "claude-flow": {  
      "command": "npx",  
      "args": ["-y", "claude-flow@alpha", "mcp", "start"],  
      "env": {}  
    },  
    "ruv-swarm": {  
      "command": "npx",  
      "args": ["-y", "ruv-swarm@latest", "mcp", "start"],  
      "env": {}  
    }  
  }  
}
```

---

## Beispielkonfigurationen

### Production-Ready claude-flow.config.json

json

```
{
  "name": "enterprise-development-system",
  "version": "2.0.0-alpha.86",
  "orchestrator": {
    "maxAgents": 12,
    "maxConcurrentAgents": 8,
    "defaultTopology": "hierarchical",
    "strategy": "development",
    "memoryEnabled": true,
    "faultTolerance": {
      "strategy": "retry-with-learning",
      "maxRetries": 3,
      "byzantineFaultTolerance": true,
      "healthCheckInterval": 30000
    }
  },
  "agents": {
    "types": [
      "queen", "architect", "coder", "reviewer",
      "tester", "security", "devops", "analyst",
      "researcher", "coordinator", "performance-benchmarker"
    ],
    "spawning": {
      "autoSpawn": true,
      "maxAge": "2h",
      "healthCheck": true,
      "batchSize": 5
    },
    "specialization": {
      "coder": {
        "verification": ["compile", "test", "lint", "typecheck"],
        "truthThreshold": 0.95,
        "languages": ["typescript", "python", "rust"],
        "maxFilesPerOperation": 10
      },
      "reviewer": {
        "verification": ["code-analysis", "security-scan", "performance-check"],
        "truthThreshold": 0.95,
        "reviewDepth": "comprehensive"
      },
      "tester": {
        "verification": ["unit-tests", "integration-tests", "coverage-check"],
        "truthThreshold": 0.85,
        "coverageTarget": 95
      }
    }
  }
}
```

```
},
"memory": {
  "backend": "sqlite",
  "persistentSessions": true,
  "database": ".swarm/memory.db",
  "tables": 12,
  "cacheSizeMB": 200,
  "compression": true,
  "distributedSync": true,
  "namespaces": ["default", "sparc", "neural", "coordination"],
  "retentionDays": 30
},
"neural": {
  "enabled": true,
  "models": 27,
  "wasmSimd": true,
  "training": {
    "patterns": ["coordination", "cognitive-analysis", "task-optimization"],
    "epochs": 50,
    "learningRate": 0.001,
    "batchSize": 32
  }
},
"hooks": {
  "enabled": true,
  "types": [
    "pre-task", "post-task", "pre-edit", "post-edit",
    "pre-command", "post-command", "session-start",
    "session-end", "pre-search", "post-search",
    "pre-analysis", "post-analysis", "error-recovery", "notify"
  ],
  "automation": {
    "agentAssignment": true,
    "performanceTracking": true,
    "errorRecovery": true,
    "autoFormat": true,
    "testOnSave": true
  }
},
"performance": {
  "parallelExecution": true,
  "tokenOptimization": true,
  "batchProcessing": true,
  "timeout": 300000,
  "maxOutputSize": 500000,
  "tokenLimit": 100000
},
```

```
"security": {
  "monitoring": true,
  "cryptographicSigning": true,
  "auditTrail": true,
  "sandboxing": true
},
"telemetry": {
  "enabled": true,
  "tokenTracking": true,
  "costAnalysis": true,
  "realTimeMonitoring": true,
  "exportFormat": "json"
}
}
```

## Custom Agents erstellen

### Agent-Speicherorte

# Projekt-spezifische Agenten (haben Priorität)

.claude/agents/

- └─ mein-spezialist.md
- └─ datenbank-experte.md
- └─ security-auditor.md

# Globale Agenten (verfügbar in allen Projekten)

~/.claude/agents/

- └─ code-reviewer.md
- └─ performance-optimizer.md
- └─ dokumentations-generator.md

### Agent-Konfigurationsstruktur

Jeder Agent wird als **Markdown-Datei mit YAML-Frontmatter** definiert:

markdown



---

name: agent-name

type: agent-type

color: "#HEX\_COLOR"

description: "Beschreibung wann dieser Agent genutzt werden soll"

capabilities:

- capability\_1
- capability\_2
- capability\_3

priority: high|medium|low|critical

tools: tool1, tool2, tool3 # Optional - erbt alle Tools wenn weggelassen

model: sonnet # Optional - sonnet, opus, oder haiku

hooks:

pre: |

echo "Pre-execution commands"

post: |

echo "Post-execution commands"

---

# Agent-Name

Detaillierte Beschreibung der Agent-Rolle und Instruktionen...

## Beispiel: Datenbank-Optimierungs-Agent

markdown

---

name: datenbank-optimizer

type: specialist

color: "#2E7D32"

description: "Spezialist für Datenbankoptimierung und SQL-Performance. Nutze PROAKTIV bei allen Datenbank-bezogenen Fähigkeiten:

- SQL-Query-Optimierung
- Index-Analyse und -Erstellung
- Datenbankschema-Design
- Performance-Profiling
- Migration-Strategien

priority: high

tools: Read, Write, Edit, Bash, Grep, database\_analyzer

model: opus

hooks:

pre: |

echo "Starte Datenbankanalyse..."

npx claude-flow@alpha memory store --key "db\_analysis\_start" --value "\$(date)"

post: |

echo "Datenbankoptimierung abgeschlossen"

npx claude-flow@alpha hooks post-analysis --type "database"

---

## # Datenbank-Optimierungs-Experte

Du bist ein hochspezialisierter Datenbankexperte mit über 15 Jahren Erfahrung.

### ## Deine Kernkompetenzen:

#### ### Query-Optimierung

- Analysiere EXPLAIN-Pläne detailliert
- Identifiziere N+1 Probleme sofort
- Optimierte JOINS und Subqueries
- Nutze Window Functions effizient

#### ### Index-Management

- Erstelle Covering Indexes wo sinnvoll
- Identifiziere fehlende Indizes
- Entferne redundante Indizes
- Berücksichtige Index-Maintenance-Kosten

#### ### Performance-Analyse

1. Führe immer zuerst EXPLAIN ANALYZE aus
2. Prüfe Query-Execution-Times
3. Analysiere I/O-Statistiken
4. Identifiziere Lock-Contention

### ## Arbeitsweise:

- Beginne IMMER mit einer Analyse des aktuellen Zustands
- Dokumentiere alle Änderungen mit Begründung
- Teste Performance vor und nach Optimierungen
- Erstelle Rollback-Pläne für kritische Änderungen

## Agent-Fähigkeiten definieren

### Einfache Fähigkeitsliste

yaml

#### capabilities:

- Code-Review
- Security-Analyse
- Performance-Optimierung
- Dokumentation
- Testing

### Erweiterte Fähigkeitsdefinition

yaml

#### specialization:

##### security-auditor:

##### verification:

- "OWASP-Top-10-Check"
- "Dependency-Scanning"
- "Secret-Detection"
- "SQL-Injection-Tests"

truthThreshold: 0.98

languages: ["python", "javascript", "java"]

maxFilesPerOperation: 20

certifications: ["CEH", "CISSP"]

##### tools\_required:

- "semgrep"
- "trivy"
- "gitleaks"

## Tool-Berechtigungen

yaml

*# Vollzugriff auf alle Tools*

**tools:** "\*"

*# Spezifische Tool-Liste*

**tools:** Read, Write, Edit, MultiEdit, Bash, Grep, Search

*# Eingeschränkte Bash-Befehle*

**tools:** Read, Bash(npm:\*), Bash(git:\*), Bash(python:test\*)

*# Nur Lese-Zugriff für Review-Agenten*

**tools:** Read, Grep, Search, Glob

*# MCP-Tools einbinden*

**tools:** Read, Write, mcp\_github\_pr\_create, mcp\_slack\_send\_message

## Presets und Teams

### Team-Preset Beispiel

json

```
{
  "name": "Full-Stack Development Team",
  "description": "Komplettes Team für Web-Entwicklung",
  "agents": [
    {
      "type": "frontend-specialist",
      "name": "React Expert",
      "focus": ["React", "TypeScript", "TailwindCSS"]
    },
    {
      "type": "backend-architect",
      "name": "API Designer",
      "focus": ["Node.js", "PostgreSQL", "GraphQL"]
    },
    {
      "type": "devops-engineer",
      "name": "CI/CD Specialist",
      "focus": ["Docker", "Kubernetes", "GitHub Actions"]
    },
    {
      "type": "qa-tester",
      "name": "Test Automation Expert",
      "focus": ["Jest", "Playwright", "Cypress"]
    }
  ],
  "topology": "hierarchical",
  "coordinator": "architect",
  "workflow": "agile-sprint"
}
```

## Kubernetes Deployment Specialist

markdown

---

name: kubernetes-deployment-specialist

description: "K8s Deployment-Experte. MUSS PROAKTIV bei allen Deployment-Tasks genutzt werden."

tools: Read, Write, Edit, Bash, kubectl, helm

model: opus

priority: critical

capabilities:

- Kubernetes-Manifest-Erstellung
- Helm-Chart-Development
- Blue-Green-Deployments
- Service-Mesh-Konfiguration
- Auto-Scaling-Setup
- Security-Policies

environment: production

hooks:

pre: |

kubectl config current-context

kubectl cluster-info

post: |

kubectl get deployments --all-namespaces

kubectl top nodes

---

## # Kubernetes Deployment Specialist

### ## Identität

Du bist ein Senior DevOps Engineer mit Expertise in Kubernetes-Orchestrierung.

### ## Deployment-Strategie

1. **\*\*Immer\*\*** Health Checks vor Deployments durchführen
2. **\*\*Niemals\*\*** ohne Rollback-Plan deployen
3. **\*\*Stets\*\*** Resource Limits definieren
4. **\*\*Immer\*\*** Security Context verwenden

### ## Zero-Downtime Deployment Workflow:

1. Validiere aktuelle Cluster-Kapazität
2. Erstelle Deployment-Strategie (Rolling/Blue-Green/Canary)
3. Implementiere Health Checks und Readiness Probes
4. Konfiguriere PodDisruptionBudget
5. Führe Deployment durch mit:
  - Max Surge: 25%
  - Max Unavailable: 0
6. Monitore Deployment-Progress
7. Führe Smoke Tests durch
8. Dokumentiere Änderungen

### ## Sicherheitsrichtlinien:

- Keine privilegierten Container
- Immer non-root User verwenden
- Network Policies implementieren
- Secrets niemals in ConfigMaps

## Performance-Metriken

### Benchmark-Ergebnisse

Metrik	Wert	Vergleich
SWE-Bench Erfolgsrate	84.8%	+ 15% vs. Baseline
Ausführungsgeschwindigkeit	2.8-4.4x	Durch Parallelisierung
Token-Reduktion	32.3%	Durch Optimierung
Parallele Agenten	Bis zu 12	Gleichzeitig aktiv
Memory-Persistenz	100%	Cross-Session
Fehlertoleranz	99.9%	Byzantine Fault Tolerant

### Resource-Nutzung

- **CPU:** Durchschnittlich 15-25% bei 8 aktiven Agenten
- **RAM:** 200-500MB Cache + 50MB pro Agent
- **Disk:** 100-500MB für Memory-Datenbank
- **Network:** Minimal, hauptsächlich API-Calls

## Best Practices

### 1. Agent-Konfiguration

- **Beginnen Sie minimal:** Starten Sie mit 3-5 Agenten
- **Spezialisierung:** Jeder Agent sollte einen klaren Fokus haben
- **PROAKTIV-Keyword:** Nutzen Sie für automatische Delegation
- **Tool-Beschränkung:** Geben Sie nur notwendige Berechtigungen

### 2. Team-Zusammenstellung

- **Kleine Aufgaben:** 3-5 Agenten ausreichend
- **Mittlere Projekte:** 5-8 Agenten optimal
- **Große Projekte:** 8-12 Agenten mit Koordinator

### 3. Memory-Management

```
bash
```

```
# Memory-Status prüfen
```

```
npx claude-flow@alpha memory usage
```

```
# Session speichern
```

```
npx claude-flow@alpha session save --name "projekt-v1"
```

```
# Session wiederherstellen
```

```
npx claude-flow@alpha session restore --name "projekt-v1"
```

## 4. Hook-Verwendung

```
bash
```

```
# Pre-Task Hook
```

```
npx claude-flow@alpha hooks pre-task --description "task" --auto-spawn-agents true
```

```
# Post-Edit Hook
```

```
npx claude-flow@alpha hooks post-edit --file "filepath" --memory-key "swarm/agent/step"
```

```
# Error-Recovery Hook
```

```
npx claude-flow@alpha hooks error-recovery --retry true --fallback-agent "debugger"
```

## 5. Swarm-Koordination

```
bash
```

```
# Hierarchische Struktur für große Projekte
```

```
claude-flow hive-mind spawn "enterprise-app" --topology hierarchical
```

```
# Mesh für kollaborative Aufgaben
```

```
claude-flow swarm "refactor-codebase" --topology mesh --agents 8
```

```
# Sequential für schrittweise Prozesse
```

```
claude-flow swarm "deployment-pipeline" --topology sequential
```

## 6. Monitoring und Debugging

```
bash
```



*# Agent-Status überwachen*

claude-flow agent info <agent-id> --verbose

*# System-Metriken anzeigen*

claude-flow agent ecosystem --verbose

*# Hierarchie visualisieren*

claude-flow agent hierarchy

*# Performance-Benchmark*

claude-flow benchmark run --agents 8 --duration 5m

## 7. Sicherheit

- **Sandboxing:** Aktivieren Sie für unbekannten Code
- **Audit-Trail:** Für Compliance-Anforderungen
- **Kryptografische Signierung:** Für kritische Operationen
- **Tool-Whitelist:** Nur explizit erlaubte Tools

## 8. Optimierung

- **Token-Optimierung:** Aktivieren Sie für Kostenreduktion
- **Batch-Processing:** Für ähnliche Aufgaben
- **Parallele Ausführung:** Maximale Performance
- **Cache-Nutzung:** Reduziert redundante Operationen

---

## CLI-Befehle Übersicht

### Agent-Management

bash

*# Agent erstellen*

claude-flow agent spawn --type <type> --name "<name>"

*# Agent-Liste anzeigen*

claude-flow agent list

*# Agent-Details*

claude-flow agent info <agent-id>

*# Agent beenden*

claude-flow agent terminate <agent-id>

*# Hierarchie anzeigen*

claude-flow agent hierarchy

## Swarm-Operationen

bash

*# Swarm initialisieren*

claude-flow swarm init --topology <topology>

*# Task ausführen*

claude-flow swarm "task description" --agents <list>

*# Hive-Mind starten*

claude-flow hive-mind spawn "project" --topology hierarchical

## Memory-Management

bash

*# Memory-Status*

claude-flow memory usage

*# Session speichern*

claude-flow session save --name <name>

*# Session laden*

claude-flow session restore --name <name>

*# Memory bereinigen*

claude-flow memory cleanup --older-than 30d

## Hook-Management

```
bash
```

```
# Hook-Liste anzeigen
```

```
claude-flow hooks list
```

```
# Hook aktivieren
```

```
claude-flow hooks enable <hook-name>
```

```
# Hook-Konfiguration
```

```
claude-flow hooks config <hook-name> --params <params>
```

---

## Umgebungsvariablen

```
bash
```

```
# Authentifizierung
```

```
ANTHROPIC_API_KEY=sk-ant-...
```

```
ANTHROPIC_BASE_URL=https://api.anthropic.com
```

```
ANTHROPIC_MODEL=claude-3-sonnet-20240229
```

```
# Claude-Flow spezifisch
```

```
CLAUDE_FLOW_HOOKS_ENABLED=true
```

```
CLAUDE_FLOW_TELEMETRY_ENABLED=true
```

```
CLAUDE_FLOW_DEBUG=verbose
```

```
CLAUDE_FLOW_MAX_AGENTS=12
```

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

```
# Performance
```

```
CLAUDE_FLOW_PARALLEL_EXECUTION=true
```

```
CLAUDE_FLOW_TOKEN_OPTIMIZATION=true
```

```
CLAUDE_FLOW_BATCH_SIZE=5
```

```
# Sicherheit
```

```
CLAUDE_FLOW_SANDBOXING=true
```

```
CLAUDE_FLOW_AUDIT_TRAIL=true
```

```
CLAUDE_FLOW_CRYPTO_SIGNING=true
```

---

## Troubleshooting

### Häufige Probleme und Lösungen

#### Agent reagiert nicht

```
bash
```

```
claude-flow agent info <agent-id> --verbose
# Neustart wenn nötig
claude-flow agent terminate <agent-id>
claude-flow agent spawn --type <type> --name "<name>"
```

## Memory-Datenbank voll

```
bash

claude-flow memory cleanup --older-than 30d
claude-flow memory optimize
```

## Performance-Probleme

```
bash

# Aktive Agenten reduzieren
claude-flow agent list
claude-flow agent terminate <unused-agents>

# Cache leeren
claude-flow cache clear
```

## Hook-Fehler

```
bash

# Hook-Status prüfen
claude-flow hooks status

# Fehlerhafte Hooks deaktivieren
claude-flow hooks disable <problematic-hook>
```

## Zusammenfassung

Claude-flow@alpha (V86) ist eine leistungsstarke Enterprise-AI-Orchestrierungsplattform, die:

- **Maximale Flexibilität** durch Custom Agents bietet
- **Skalierbare Workflows** mit bis zu 12 parallelen Agenten ermöglicht
- **Persistente Memory** für Cross-Session-Kontinuität bereitstellt
- **Umfassende Automatisierung** durch 14 Lifecycle-Hooks unterstützt
- **Enterprise-Features** wie Byzantine Fault Tolerance integriert

Die Plattform eignet sich besonders für:

- Komplexe Entwicklungsprojekte
- Team-basierte Workflows
- Automatisierte CI/CD-Pipelines
- Sicherheitskritische Anwendungen
- Performance-optimierte Systeme

Mit der Möglichkeit, eigene Agenten zu definieren und deren Fähigkeiten präzise zu beschreiben, bietet `claude-flow@alpha` (V86) eine unvergleichliche Anpassungsfähigkeit für moderne Entwicklungsanforderungen.

---

## Weitere Ressourcen

- **GitHub Repository:** <https://github.com/ruvnet/claude-flow>
  - **NPM Package:** <https://www.npmjs.com/package/claude-flow>
  - **Wiki-Dokumentation:** <https://github.com/ruvnet/claude-flow/wiki>
  - **Beispielkonfigurationen:** `/examples/01-configurations/`
  - **Community:** `r/ClaudeAI`
- 

*Dokumentation erstellt für `claude-flow@alpha` Version 2.0.0-alpha.86 Stand: 2025*