Claude-Flow Swarm-Preset Konfigurationshandbuch

Übersicht

Dieses Handbuch erklärt detailliert, wie Swarm-Presets in Claude-Flow v2.0.0-alpha.86 erstellt, konfiguriert und verwendet werden. Swarm-Presets sind vordefinierte Konfigurationen für Agenten-Teams, die für spezifische Entwicklungsaufgaben optimiert sind.

Inhaltsverzeichnis

Grundkonzepte

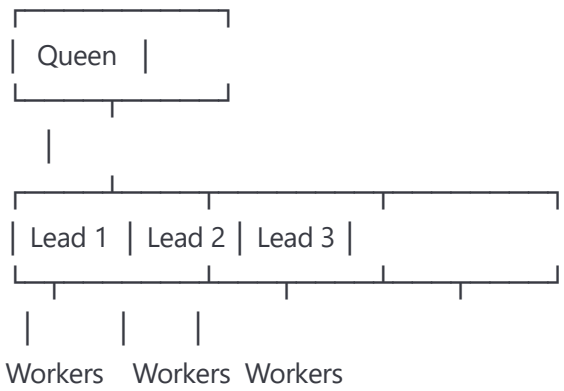
Was sind Swarm-Presets?

Swarm-Presets sind JSON-Konfigurationsdateien, die:

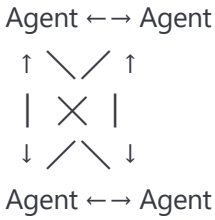
- **Agenten-Teams** definieren mit spezifischen Rollen und Fähigkeiten
- **Workflows** orchestrieren für komplexe Entwicklungsaufgaben
- **Werkzeuge und Ressourcen** zuweisen an einzelne Agenten
- **Memory und Koordination** zwischen Agenten ermöglichen

Hierarchie-Modelle

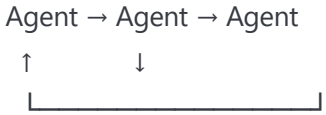
1. Hierarchical (Empfohlen für Teams)



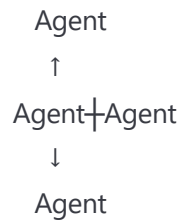
2. Mesh (Kollaboration)



3. Ring (Sequenziell)



4. Star (Zentralisiert)



Dateistruktur

Projekt-Layout nach Initialisierung

bash

```
your-project/
├── .claude/
│   ├── settings.json          # Haupt-Claude-Flow-Konfiguration
│   ├── commands/              # Custom Slash-Commands
│   │   └── python-dev.md      # Python-spezifische Commands
│   └── hooks/                 # Hook-Scripts
│       ├── pre-tool.sh
│       └── post-tool.sh
├── .hive-mind/
│   ├── config.json            # Hive-Mind-Konfiguration
│   └── sessions.db            # SQLite Session-Datenbank
├── .swarm/
│   ├── memory.db              # Persistente Memory-Datenbank
│   └── presets/               # Lokale Presets
│       ├── python-dev.json
│       ├── js-fullstack.json
│       └── custom-team.json
├── swarms/                    # Organisierte Swarm-Configs
│   ├── development/
│   │   ├── python-development.json
│   │   ├── node-development.json
│   │   └── rust-development.json
│   ├── testing/
│   │   ├── unit-test-swarm.json
│   │   └── e2e-test-swarm.json
│   └── production/
│       ├── enterprise-team.json
│       └── microservices.json
└── workflows/
    ├── python-workflow.json
    └── deployment-pipeline.json
```

Globale Preset-Speicherorte

bash

```
# Benutzer-Home-Verzeichnis
~/.claude-flow/
├── config.json              # Globale Konfiguration
├── presets/                 # Globale Presets
│   ├── _index.json          # Preset-Registry
│   ├── python-development.json
│   ├── javascript-fullstack.json
│   ├── rust-systems.json
│   ├── go-microservices.json
│   └── enterprise-team.json
└── templates/               # Basis-Templates
    ├── base-agent.json
    └── base-workflow.json

# System-weite Presets (falls als Admin installiert)
/usr/local/share/claude-flow/presets/
└── official/                # Offizielle Presets
    ├── python-development.json
    └── ...
```

JSON-Schema

Vollständiges Schema mit allen Optionen

json

```
{
  "$schema": "https://claude-flow.org/schemas/v2/swarm-preset.json",
  "version": "2.0.0",
  "name": "string (required)",
  "description": "string (optional)",
  "metadata": {
    "author": "string",
    "created": "ISO-8601 date",
    "modified": "ISO-8601 date",
    "version": "semver string",
    "tags": ["array", "of", "strings"],
    "difficulty": "beginner|intermediate|advanced|expert",
    "estimatedTime": "duration string (e.g., '2h', '30m')",
    "requirements": {
      "minAgents": "number",
      "memory": "string (e.g., '4GB')",
      "tools": ["required", "tool", "list"]
    }
  },
  "orchestrator": {
    "topology": "hierarchical|mesh|ring|star|simple",
    "maxConcurrentAgents": "number (1-20)",
    "coordinationLevel": "low|medium|high|enterprise",
    "memoryNamespace": "string",
    "strategy": "parallel|sequential|adaptive|research|development",
    "defaultTimeout": "number (seconds)",
    "retryAttempts": "number",
    "fallbackStrategy": "string"
  },
  "agents": [
    {
      "id": "string (unique)",
      "type": "coordinator|architect|coder|tester|security|devops|specialist|researcher|analyst|writer",
      "name": "string",
      "role": "queen|lead|worker",
      "priority": "number (1-10)",
      "model": "opus|sonnet|haiku",
      "temperature": "number (0-1)",
      "capabilities": ["array", "of", "capabilities"],
      "prompt": "string (system prompt)",
      "tools": ["allowed", "tool", "names"],
      "directory": "string (working directory path)",
      "connections": ["array", "of", "agent", "ids"],
      "memory": {
        "size": "string (e.g., '500MB')",
        "persistent": "boolean",
```

```
      "shared": "boolean"
    },
    "specialization": {
      "expertise": ["array", "of", "expertise"],
      "frameworks": ["supported", "frameworks"],
      "languages": ["programming", "languages"],
      "tools": ["specialized", "tools"],
      "patterns": ["design", "patterns"],
      "focus": ["main", "focus", "areas"]
    },
    "config": {
      "custom": "object with any configuration"
    },
    "hooks": {
      "onStart": "string (script path)",
      "onComplete": "string (script path)",
      "onError": "string (script path)"
    }
  }
],
"workflow": {
  "autoStart": "boolean",
  "phases": [
    {
      "name": "string",
      "description": "string",
      "agents": ["agent", "ids"],
      "parallel": "boolean",
      "duration": "string (e.g., '10m')",
      "requires": ["prerequisite", "phases"],
      "outputs": ["expected", "output", "files"],
      "validation": {
        "type": "test|review|approval",
        "criteria": "object"
      }
    }
  ],
  "dependencies": {
    "graph": "object (DAG representation)"
  },
  "triggers": {
    "onSuccess": "string (action)",
    "onFailure": "string (action)",
    "onTimeout": "string (action)"
  }
},
"memory": {
```

```
    "persistent": "boolean",
    "type": "sqlite|redis|inmemory",
    "sharedNamespaces": ["array", "of", "namespaces"],
    "compression": "none|low|medium|high",
    "maxSize": "string (e.g., '1GB')",
    "ttl": "number (seconds)",
    "syncInterval": "number (seconds)"
  },
  "hooks": {
    "global": {
      "preInit": "string (script path)",
      "postInit": "string (script path)",
      "preToolUse": "string (script path)",
      "postToolUse": "string (script path)",
      "onError": "string (script path)",
      "onComplete": "string (script path)",
      "onTimeout": "string (script path)"
    },
    "perAgent": {
      "agentId": {
        "preStart": "string",
        "postComplete": "string"
      }
    }
  },
  "environment": {
    "KEY": "value",
    "ANOTHER_KEY": "another value"
  },
  "tools": {
    "allowed": ["Bash", "Edit", "Write", "Read", "WebSearch", "WebFetch"],
    "disallowed": ["dangerous", "tools"],
    "custom": [
      {
        "name": "string",
        "command": "string",
        "args": ["array", "of", "arguments"],
        "description": "string",
        "allowedAgents": ["agent", "ids"],
        "timeout": "number"
      }
    ],
    "permissions": {
      "filesystem": {
        "read": ["allowed", "paths"],
        "write": ["allowed", "paths"],
        "execute": ["allowed", "commands"]
```

```
    },
    "network": {
      "allowed": ["domains"],
      "blocked": ["domains"]
    }
  }
},
"mcp": {
  "enabled": "boolean",
  "servers": {
    "serverName": {
      "command": "string",
      "args": ["array", "of", "args"],
      "env": {
        "KEY": "value"
      },
      "type": "stdio|http|websocket"
    }
  },
  "autoConnect": "boolean",
  "retryPolicy": {
    "maxAttempts": "number",
    "backoff": "exponential|linear"
  }
},
"monitoring": {
  "enabled": "boolean",
  "level": "basic|detailed|debug",
  "metrics": ["cpu", "memory", "io", "network", "tasks"],
  "dashboard": "boolean",
  "dashboardPort": "number",
  "export": {
    "format": "json|csv|prometheus",
    "destination": "string (file path or URL)",
    "interval": "number (seconds)"
  },
  "alerts": {
    "errorThreshold": "number",
    "timeoutThreshold": "number (seconds)",
    "memoryThreshold": "string (percentage)",
    "cpuThreshold": "string (percentage)",
    "notifications": {
      "type": "email|slack|webhook",
      "destination": "string"
    }
  }
},
```

```
  "optimization": {
    "parallelExecution": "boolean",
    "maxParallelTasks": "number",
    "batchSize": "number",
    "caching": {
      "enabled": "boolean",
      "strategy": "lru|lfu|ttl",
      "maxSize": "string"
    },
    "autoScale": {
      "enabled": "boolean",
      "minAgents": "number",
      "maxAgents": "number",
      "scaleUpThreshold": "number (0-1)",
      "scaleDownThreshold": "number (0-1)",
      "cooldownPeriod": "number (seconds)"
    },
    "resourceLimits": {
      "cpu": "string (percentage)",
      "memory": "string (e.g., '2GB')",
      "diskIO": "string (MB/s)"
    }
  },
  "security": {
    "authentication": {
      "required": "boolean",
      "method": "token|oauth|basic"
    },
    "encryption": {
      "atRest": "boolean",
      "inTransit": "boolean"
    },
    "audit": {
      "enabled": "boolean",
      "logLevel": "all|errors|warnings",
      "destination": "string (file path)"
    }
  },
  "integration": {
    "github": {
      "enabled": "boolean",
      "token": "${GITHUB_TOKEN}",
      "repo": "string",
      "branch": "string",
      "workflows": ["workflow", "names"]
    },
    "slack": {
```

```
      "enabled": "boolean",
      "webhook": "string",
      "channel": "string"
    },
    "ci": {
      "provider": "github-actions|jenkins|gitlab|circleci",
      "config": "object"
    }
  },
  "testing": {
    "framework": "pytest|jest|mocha|junit",
    "coverage": {
      "enabled": "boolean",
      "threshold": "number (percentage)",
      "failOnLowCoverage": "boolean"
    },
    "patterns": ["unit", "integration", "e2e", "performance"],
    "autoRun": "boolean"
  },
  "deployment": {
    "strategy": "blue-green|canary|rolling|direct",
    "targets": {
      "development": "object",
      "staging": "object",
      "production": "object"
    },
    "rollback": {
      "automatic": "boolean",
      "threshold": "object"
    }
  }
}
```

Python Development Preset - Vollständiges Beispiel

Datei: `python-development.json`

```
json
```

```
{
  "version": "2.0.0",
  "name": "Python Full-Stack Development Team",
  "description": "Comprehensive Python development swarm with FastAPI, SQLAlchemy, pytest, and modern DevOps pr
  "metadata": {
    "author": "Claude-Flow Community",
    "created": "2025-08-18T10:00:00Z",
    "version": "1.2.0",
    "tags": ["python", "fastapi", "sqlalchemy", "pytest", "docker", "kubernetes"],
    "difficulty": "intermediate",
    "estimatedTime": "2h",
    "requirements": {
      "minAgents": 5,
      "memory": "4GB",
      "tools": ["uv", "docker", "git", "pytest"]
    }
  },
  "orchestrator": {
    "topology": "hierarchical",
    "maxConcurrentAgents": 10,
    "coordinationLevel": "high",
    "memoryNamespace": "python-fullstack",
    "strategy": "development",
    "defaultTimeout": 900,
    "retryAttempts": 3,
    "fallbackStrategy": "sequential"
  },
  "agents": [
    {
      "id": "project-manager",
      "type": "coordinator",
      "name": "Python Project Manager",
      "role": "queen",
      "priority": 10,
      "model": "opus",
      "capabilities": [
        "project-planning",
        "task-delegation",
        "progress-monitoring",
        "resource-allocation",
        "risk-management"
      ],
      "prompt": "You are an experienced Python project manager coordinating a full-stack development team. You use Fa
      "memory": {
        "size": "1GB",
        "persistent": true,
```

```
    "shared": true
  },
  "connections": ["architect", "backend-lead", "frontend-lead", "devops-lead", "qa-lead"]
},
{
  "id": "architect",
  "type": "architect",
  "name": "Python Solutions Architect",
  "role": "lead",
  "priority": 9,
  "model": "opus",
  "capabilities": [
    "system-design",
    "architecture-patterns",
    "database-design",
    "api-design",
    "scalability-planning",
    "technology-selection"
  ],
  "prompt": "Design scalable Python architectures using Domain-Driven Design principles. Create comprehensive syst
  "tools": ["Read", "Write", "WebSearch"],
  "directory": "./architecture",
  "specialization": {
    "expertise": ["microservices", "ddd", "event-sourcing", "cqrs"],
    "frameworks": ["FastAPI", "Django", "Flask", "Starlette"],
    "databases": ["PostgreSQL", "MongoDB", "Redis", "Elasticsearch"],
    "patterns": ["Repository", "Unit of Work", "CQRS", "Saga", "Event Sourcing"],
    "cloud": ["AWS", "GCP", "Azure", "Kubernetes"]
  },
  "hooks": {
    "onComplete": "./hooks/generate-architecture-diagram.sh"
  }
},
{
  "id": "backend-lead",
  "type": "coordinator",
  "name": "Backend Team Lead",
  "role": "lead",
  "priority": 8,
  "model": "opus",
  "connections": ["api-developer", "data-engineer", "integration-specialist"],
  "prompt": "Lead the backend development team. Coordinate API development, database design, and service integr
  "directory": "./backend"
},
{
  "id": "api-developer",
  "type": "coder",
```

```
    "name": "Senior API Developer",
    "role": "worker",
    "priority": 8,
    "model": "sonnet",
    "capabilities": [
      "rest-api",
      "graphql",
      "websockets",
      "authentication",
      "rate-limiting",
      "api-versioning"
    ],
    "prompt": "Implement RESTful APIs using FastAPI with async/await patterns. Create OpenAPI documentation, imple
    "tools": ["Edit", "Write", "Read", "Bash"],
    "directory": "./backend/api",
    "specialization": {
      "expertise": ["fastapi", "async-python", "openapi", "graphql"],
      "libraries": [
        "fastapi",
        "pydantic",
        "httpx",
        "strawberry-graphql",
        "python-jose",
        "python-multipart",
        "email-validator",
        "uvicorn"
      ],
      "patterns": ["REST", "GraphQL", "WebSocket", "gRPC"]
    },
    "config": {
      "linting": "ruff",
      "formatting": "black",
      "typeChecking": "mypy"
    }
  },
  {
    "id": "data-engineer",
    "type": "specialist",
    "name": "Python Data Engineer",
    "role": "worker",
    "priority": 7,
    "model": "sonnet",
    "capabilities": [
      "database-design",
      "orm",
      "migrations",
      "query-optimization",
```

```
      "data-pipelines",
      "etl"
    ],
    "prompt": "Design and implement database schemas using SQLAlchemy ORM. Create Alembic migrations, optimize
    "tools": ["Edit", "Write", "Bash"],
    "directory": "./backend/data",
    "specialization": {
      "expertise": ["sqlalchemy", "alembic", "query-optimization", "etl"],
      "libraries": [
        "sqlalchemy",
        "alembic",
        "psycopg2",
        "pymongo",
        "redis-py",
        "pandas",
        "numpy",
        "dask"
      ],
      "databases": ["PostgreSQL", "MySQL", "MongoDB", "Redis", "TimescaleDB"],
      "patterns": ["Repository", "Unit of Work", "Active Record", "Data Mapper"]
    },
    "hooks": {
      "postToolUse": "./hooks/run-migrations.sh"
    }
  },
  {
    "id": "integration-specialist",
    "type": "specialist",
    "name": "Integration Specialist",
    "role": "worker",
    "priority": 7,
    "model": "sonnet",
    "capabilities": [
      "third-party-apis",
      "message-queues",
      "event-streaming",
      "webhooks",
      "service-mesh"
    ],
    "prompt": "Integrate with external services and APIs. Implement message queuing with RabbitMQ/Kafka, webhook
    "tools": ["Edit", "Write", "WebFetch"],
    "directory": "./backend/integrations",
    "specialization": {
      "expertise": ["rabbitmq", "kafka", "celery", "redis-streams"],
      "libraries": ["celery", "kombu", "aiokafka", "aio-pika", "httpx", "tenacity"],
      "services": ["Stripe", "SendGrid", "Twilio", "AWS", "OAuth providers"]
    }
```

```
},
{
  "id": "frontend-lead",
  "type": "coordinator",
  "name": "Frontend Team Lead",
  "role": "lead",
  "priority": 8,
  "model": "opus",
  "connections": ["ui-developer", "frontend-tester"],
  "prompt": "Coordinate frontend development for Python web applications. Manage UI/UX implementation and fron
  "directory": "./frontend"
},
{
  "id": "ui-developer",
  "type": "coder",
  "name": "UI Developer",
  "role": "worker",
  "priority": 7,
  "model": "sonnet",
  "capabilities": ["jinja2", "htmx", "alpine-js", "tailwind"],
  "prompt": "Create responsive web interfaces using Jinja2 templates, HTMX for dynamic interactions, Alpine.js for clie
  "tools": ["Edit", "Write"],
  "directory": "./frontend/templates"
},
{
  "id": "qa-lead",
  "type": "tester",
  "name": "QA Team Lead",
  "role": "lead",
  "priority": 8,
  "model": "opus",
  "connections": ["test-engineer", "performance-tester"],
  "prompt": "Lead quality assurance efforts. Coordinate testing strategies, maintain test coverage above 90%, and ens
  "directory": "./tests"
},
{
  "id": "test-engineer",
  "type": "tester",
  "name": "Python Test Engineer",
  "role": "worker",
  "priority": 8,
  "model": "sonnet",
  "capabilities": [
    "unit-testing",
    "integration-testing",
    "e2e-testing",
    "tdd",
```

```
      "bdd",
      "test-automation"
    ],
    "prompt": "Write comprehensive tests using pytest following TDD principles. Create unit tests, integration tests, and
    "tools": ["Write", "Edit", "Bash"],
    "directory": "./tests",
    "specialization": {
      "frameworks": ["pytest", "unittest", "behave", "hypothesis"],
      "libraries": [
        "pytest",
        "pytest-asyncio",
        "pytest-cov",
        "pytest-mock",
        "pytest-xdist",
        "factory-boy",
        "faker",
        "hypothesis",
        "responses",
        "httpx"
      ],
      "patterns": ["AAA", "Given-When-Then", "Page Object", "Test Data Builder"]
    },
    "config": {
      "coverageThreshold": 90,
      "parallel": true,
      "markers": ["unit", "integration", "e2e", "slow"]
    }
  },
  {
    "id": "performance-tester",
    "type": "tester",
    "name": "Performance Test Engineer",
    "role": "worker",
    "priority": 6,
    "model": "sonnet",
    "capabilities": ["load-testing", "stress-testing", "profiling"],
    "prompt": "Conduct performance testing using Locust. Profile code with py-spy and memory_profiler. Identify bottle
    "tools": ["Write", "Bash"],
    "directory": "./tests/performance",
    "specialization": {
      "tools": ["locust", "py-spy", "memory_profiler", "line_profiler"]
    }
  },
  {
    "id": "devops-lead",
    "type": "devops",
    "name": "DevOps Team Lead",
```

```
    "role": "lead",
    "priority": 8,
    "model": "opus",
    "connections": ["infrastructure-engineer", "security-engineer"],
    "prompt": "Lead DevOps initiatives. Coordinate CI/CD, infrastructure as code, and security practices.",
    "directory": "./devops"
  },
  {
    "id": "infrastructure-engineer",
    "type": "devops",
    "name": "Infrastructure Engineer",
    "role": "worker",
    "priority": 7,
    "model": "sonnet",
    "capabilities": [
      "docker",
      "kubernetes",
      "terraform",
      "ci-cd",
      "monitoring",
      "logging"
    ],
    "prompt": "Create Docker containers, Kubernetes manifests, and Terraform configurations. Setup CI/CD with GitHub
    "tools": ["Write", "Bash"],
    "directory": "./devops/infrastructure",
    "specialization": {
      "tools": ["docker", "kubernetes", "terraform", "ansible", "helm"],
      "ci": ["github-actions", "gitlab-ci", "jenkins"],
      "monitoring": ["prometheus", "grafana", "datadog", "new-relic"],
      "cloud": ["AWS", "GCP", "Azure"]
    },
    "config": {
      "containerRegistry": "ghcr.io",
      "kubernetesNamespace": "production",
      "terraformBackend": "s3"
    }
  },
  {
    "id": "security-engineer",
    "type": "security",
    "name": "Security Engineer",
    "role": "worker",
    "priority": 8,
    "model": "opus",
    "capabilities": [
      "security-audit",
      "vulnerability-scanning",
```

```
      "authentication",
      "encryption",
      "compliance"
    ],
    "prompt": "Implement security best practices. Conduct security audits with Bandit and Safety, implement OAuth2/JW
    "tools": ["Read", "Edit", "WebSearch"],
    "directory": "./security",
    "specialization": {
      "tools": ["bandit", "safety", "pip-audit", "trivy", "owasp-zap"],
      "standards": ["OWASP", "PCI-DSS", "GDPR", "SOC2"],
      "focus": ["authentication", "authorization", "encryption", "secrets-management"]
    }
  },
  {
    "id": "documentation-specialist",
    "type": "writer",
    "name": "Documentation Specialist",
    "role": "worker",
    "priority": 5,
    "model": "sonnet",
    "capabilities": [
      "technical-writing",
      "api-documentation",
      "user-guides",
      "architecture-docs"
    ],
    "prompt": "Create comprehensive documentation using Sphinx and MkDocs. Write clear docstrings following Goog
    "tools": ["Write", "Read"],
    "directory": "./docs",
    "specialization": {
      "tools": ["sphinx", "mkdocs", "swagger", "redoc"],
      "formats": ["markdown", "restructuredtext", "openapi"],
      "types": ["api-docs", "user-guides", "developer-docs", "adrs"]
    }
  }
],
"workflow": {
  "autoStart": true,
  "phases": [
    {
      "name": "planning",
      "description": "Architecture design and project planning",
      "agents": ["project-manager", "architect"],
      "duration": "15m",
      "outputs": ["architecture/design.md", "architecture/api-spec.yaml", "requirements.txt"]
    },
    {
```

```
    "name": "setup",
    "description": "Project setup and configuration",
    "agents": ["infrastructure-engineer"],
    "duration": "10m",
    "outputs": ["Dockerfile", "docker-compose.yml", ".github/workflows/ci.yml"]
  },
  {
    "name": "database-design",
    "description": "Database schema and migrations",
    "agents": ["data-engineer"],
    "duration": "15m",
    "requires": ["planning"],
    "outputs": ["backend/data/models.py", "alembic/versions/"]
  },
  {
    "name": "api-development",
    "description": "API implementation",
    "agents": ["api-developer", "integration-specialist"],
    "parallel": true,
    "duration": "30m",
    "requires": ["database-design"],
    "outputs": ["backend/api/", "backend/integrations/"]
  },
  {
    "name": "frontend-development",
    "description": "UI implementation",
    "agents": ["ui-developer"],
    "duration": "20m",
    "requires": ["api-development"],
    "outputs": ["frontend/templates/", "frontend/static/"]
  },
  {
    "name": "testing",
    "description": "Comprehensive testing",
    "agents": ["test-engineer", "performance-tester"],
    "parallel": true,
    "duration": "20m",
    "requires": ["api-development", "frontend-development"],
    "outputs": ["tests/", "coverage.xml"],
    "validation": {
      "type": "test",
      "criteria": {
        "coverage": 90,
        "passing": true
      }
    }
  },
```

```
    {
      "name": "security-review",
      "description": "Security audit and hardening",
      "agents": ["security-engineer"],
      "duration": "15m",
      "requires": ["testing"],
      "outputs": ["security/audit-report.md", "security/recommendations.md"]
    },
    {
      "name": "deployment-prep",
      "description": "Prepare for deployment",
      "agents": ["infrastructure-engineer"],
      "duration": "10m",
      "requires": ["security-review"],
      "outputs": ["k8s/", "terraform/"]
    },
    {
      "name": "documentation",
      "description": "Generate documentation",
      "agents": ["documentation-specialist"],
      "parallel": true,
      "duration": "15m",
      "outputs": ["docs/"]
    }
  ],
  "triggers": {
    "onSuccess": "notify-slack",
    "onFailure": "rollback-and-alert",
    "onTimeout": "escalate-to-human"
  }
},
"memory": {
  "persistent": true,
  "type": "sqlite",
  "sharedNamespaces": [
    "project-context",
    "api-design",
    "database-schema",
    "test-results",
    "security-findings"
  ],
  "compression": "medium",
  "maxSize": "2GB",
  "syncInterval": 30
},
"hooks": {
  "global": {
```

```
    "preInit": "./hooks/setup-environment.sh",
    "postInit": "./hooks/verify-setup.sh",
    "preToolUse": "./hooks/log-tool-use.py",
    "postToolUse": "./hooks/validate-output.py",
    "onError": "./hooks/error-handler.py",
    "onComplete": "./hooks/generate-report.py"
  },
  "perAgent": {
    "test-engineer": {
      "postComplete": "./hooks/run-coverage-report.sh"
    },
    "infrastructure-engineer": {
      "postComplete": "./hooks/validate-infrastructure.sh"
    }
  }
},
"environment": {
  "PYTHON_VERSION": "3.11",
  "UV_SYSTEM_PYTHON": "true",
  "FASTAPI_ENV": "development",
  "DATABASE_URL": "postgresql://user:pass@localhost/dbname",
  "REDIS_URL": "redis://localhost:6379",
  "CELERY_BROKER_URL": "redis://localhost:6379/0",
  "PYTEST_ADDOPTS": "--cov=backend --cov-report=term-missing --cov-report=xml",
  "BLACK_LINE_LENGTH": "100",
  "MYPY_FLAGS": "--strict --ignore-missing-imports"
},
"tools": {
  "allowed": ["Bash", "Edit", "Write", "Read", "WebSearch", "WebFetch"],
  "custom": [
    {
      "name": "uv-sync",
      "command": "uv sync",
      "description": "Sync Python dependencies with uv",
      "allowedAgents": ["api-developer", "data-engineer", "test-engineer"]
    },
    {
      "name": "pytest",
      "command": "pytest",
      "args": ["--verbose", "--cov=backend", "--cov-report=term"],
      "description": "Run pytest with coverage",
      "allowedAgents": ["test-engineer"],
      "timeout": 300
    },
    {
      "name": "alembic",
      "command": "alembic",
```

```
      "args": ["upgrade", "head"],
      "description": "Run database migrations",
      "allowedAgents": ["data-engineer"]
    },
    {
      "name": "ruff",
      "command": "ruff",
      "args": ["check", "--fix"],
      "description": "Run ruff linter with auto-fix",
      "allowedAgents": ["api-developer", "data-engineer"]
    },
    {
      "name": "mypy",
      "command": "mypy",
      "args": ["backend/"],
      "description": "Type checking with mypy",
      "allowedAgents": ["api-developer"]
    },
    {
      "name": "docker-build",
      "command": "docker",
      "args": ["build", "-t", "app:latest", "."],
      "description": "Build Docker image",
      "allowedAgents": ["infrastructure-engineer"]
    },
    {
      "name": "security-scan",
      "command": "bandit",
      "args": ["-r", "backend/", "-f", "json"],
      "description": "Security scan with Bandit",
      "allowedAgents": ["security-engineer"]
    }
  ],
  "permissions": {
    "filesystem": {
      "read": ["./", "/tmp"],
      "write": ["./", "/tmp"],
      "execute": ["uv", "python", "pytest", "docker", "kubectl"]
    },
    "network": {
      "allowed": ["github.com", "pypi.org", "docker.io"],
      "blocked": ["malicious-site.com"]
    }
  }
},
"mcp": {
  "enabled": true,
```

```
  "servers": {
    "filesystem": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "./"],
      "type": "stdio"
    },
    "memory": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-memory"],
      "type": "stdio"
    },
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      },
      "type": "stdio"
    },
    "postgres": {
      "command": "npx",
      "args": ["-y", "postgres-mcp-server"],
      "env": {
        "DATABASE_URL": "${DATABASE_URL}"
      },
      "type": "stdio"
    }
  },
  "autoConnect": true,
  "retryPolicy": {
    "maxAttempts": 3,
    "backoff": "exponential"
  }
},
"monitoring": {
  "enabled": true,
  "level": "detailed",
  "metrics": ["cpu", "memory", "io", "network", "tasks", "errors"],
  "dashboard": true,
  "dashboardPort": 3001,
  "export": {
    "format": "prometheus",
    "destination": "http://localhost:9090/metrics",
    "interval": 60
  },
  "alerts": {
    "errorThreshold": 10,
```

```
    "timeoutThreshold": 1800,
    "memoryThreshold": "85%",
    "cpuThreshold": "90%",
    "notifications": {
      "type": "slack",
      "destination": "${SLACK_WEBHOOK_URL}"
    }
  }
},
"optimization": {
  "parallelExecution": true,
  "maxParallelTasks": 5,
  "batchSize": 3,
  "caching": {
    "enabled": true,
    "strategy": "lru",
    "maxSize": "500MB"
  },
  "autoScale": {
    "enabled": true,
    "minAgents": 5,
    "maxAgents": 15,
    "scaleUpThreshold": 0.8,
    "scaleDownThreshold": 0.3,
    "cooldownPeriod": 60
  },
  "resourceLimits": {
    "cpu": "80%",
    "memory": "4GB",
    "diskIO": "100MB/s"
  }
},
"testing": {
  "framework": "pytest",
  "coverage": {
    "enabled": true,
    "threshold": 90,
    "failOnLowCoverage": true
  },
  "patterns": ["unit", "integration", "e2e", "performance", "security"],
  "autoRun": true
},
"deployment": {
  "strategy": "blue-green",
  "targets": {
    "development": {
      "url": "http://localhost:8000",
```

```
      "branch": "develop"
    },
    "staging": {
      "url": "https://staging.example.com",
      "branch": "staging",
      "approval": true
    },
    "production": {
      "url": "https://api.example.com",
      "branch": "main",
      "approval": true,
      "rollback": true
    }
  },
  "rollback": {
    "automatic": true,
    "threshold": {
      "errorRate": 0.05,
      "responseTime": 2000
    }
  }
 },
 "integration": {
  "github": {
    "enabled": true,
    "token": "${GITHUB_TOKEN}",
    "repo": "username/python-project",
    "branch": "main",
    "workflows": ["ci", "cd", "security-scan"]
  },
  "slack": {
    "enabled": true,
    "webhook": "${SLACK_WEBHOOK_URL}",
    "channel": "#dev-team"
  },
  "ci": {
    "provider": "github-actions",
    "config": {
      "onPush": true,
      "onPullRequest": true,
      "schedule": "0 0 * * *"
    }
  }
 }
}
```

Weitere Preset-Beispiele

JavaScript/TypeScript Full-Stack

json

```
{
  "version": "2.0.0",
  "name": "JavaScript Full-Stack Team",
  "agents": [
    {
      "id": "lead",
      "type": "coordinator",
      "name": "Full-Stack Lead",
      "connections": ["frontend-dev", "backend-dev", "devops"]
    },
    {
      "id": "frontend-dev",
      "type": "coder",
      "name": "React Developer",
      "specialization": {
        "frameworks": ["React", "Next.js", "Vue"],
        "tools": ["Webpack", "Vite", "TypeScript"]
      }
    },
    {
      "id": "backend-dev",
      "type": "coder",
      "name": "Node.js Developer",
      "specialization": {
        "frameworks": ["Express", "NestJS", "Fastify"],
        "databases": ["MongoDB", "PostgreSQL"]
      }
    }
  ]
}
```

Microservices Team

json

```
{
  "version": "2.0.0",
  "name": "Microservices Architecture Team",
  "orchestrator": {
    "topology": "mesh",
    "strategy": "parallel"
  },
  "agents": [
    {
      "id": "api-gateway",
      "type": "specialist",
      "name": "API Gateway Specialist"
    },
    {
      "id": "service-1",
      "type": "coder",
      "name": "User Service Developer"
    },
    {
      "id": "service-2",
      "type": "coder",
      "name": "Order Service Developer"
    },
    {
      "id": "service-3",
      "type": "coder",
      "name": "Payment Service Developer"
    },
    {
      "id": "message-broker",
      "type": "specialist",
      "name": "Message Queue Specialist"
    }
  ]
}
```

Preset-Verwaltung

CLI-Befehle

bash

```bash
# Preset erstellen mit Wizard
npx claude-flow@alpha preset create --wizard

# Preset aus Template erstellen
npx claude-flow@alpha preset create my-team \
  --template python-development \
  --agents 8 \
  --output ./my-team.json

# Preset validieren
npx claude-flow@alpha preset validate ./my-team.json

# Preset testen (Dry-Run)
npx claude-flow@alpha swarm test --preset ./my-team.json --dry-run

# Preset installieren
npx claude-flow@alpha preset install ./my-team.json --name "My Team"

# Globales Preset installieren
npx claude-flow@alpha preset install ./my-team.json --global

# Preset aktualisieren
npx claude-flow@alpha preset update python-development --version 1.3.0

# Preset löschen
npx claude-flow@alpha preset remove my-team
```

Preset-Registry

```bash
# Alle verfügbaren Presets anzeigen
npx claude-flow@alpha preset list

# Nach Presets suchen
npx claude-flow@alpha preset search python

# Preset-Details anzeigen
npx claude-flow@alpha preset info python-development

# Preset herunterladen
npx claude-flow@alpha preset download python-development

# Community-Presets durchsuchen
npx claude-flow@alpha preset browse --community
```

Erweiterte Konfigurationen

Dynamic Agent Spawning

json

```
{
  "agents": [
    {
      "id": "spawner",
      "type": "coordinator",
      "capabilities": ["dynamic-spawning"],
      "config": {
        "spawnRules": [
          {
            "condition": "workload > 80%",
            "action": "spawn",
            "agentType": "coder",
            "count": 2
          },
          {
            "condition": "workload < 30%",
            "action": "terminate",
            "agentType": "coder",
            "count": 1
          }
        ]
      }
    }
  ]
}
```

Conditional Workflows

json

```
{
  "workflow": {
    "phases": [
      {
        "name": "analysis",
        "agents": ["analyzer"],
        "conditional": {
          "if": "complexity == 'high'",
          "then": {
            "agents": ["senior-dev", "architect"],
            "duration": "30m"
          },
          "else": {
            "agents": ["junior-dev"],
            "duration": "15m"
          }
        }
      }
    ]
  }
}
```

Multi-Environment Configuration

```
json
```

```
{
  "environments": {
    "development": {
      "agents": 5,
      "timeout": 600,
      "monitoring": false
    },
    "staging": {
      "agents": 8,
      "timeout": 900,
      "monitoring": true
    },
    "production": {
      "agents": 12,
      "timeout": 1800,
      "monitoring": true,
      "alerts": true
    }
  }
}
```

Integration und Automatisierung

GitHub Actions Integration

yaml

```yaml
# .github/workflows/claude-flow.yml
name: Claude-Flow Development

on:
  push:
    branches: [main, develop]
  pull_request:
    types: [opened, synchronize]

jobs:
  swarm-development:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Claude-Flow
        run: |
          npm install -g claude-flow@alpha
          claude-flow --version

      - name: Load Python Preset
        run: |
          claude-flow preset validate ./swarms/python-development.json

      - name: Run Development Swarm
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          claude-flow swarm create \
            --preset python-development \
            --task "${{ github.event.pull_request.title }}" \
            --max-agents 8 \
            --timeout 30 \
            --output ./swarm-results.json

      - name: Upload Results
        uses: actions/upload-artifact@v3
        with:
          name: swarm-results
          path: ./swarm-results.json
```

Docker Compose Integration

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  claude-flow:
    image: claude-flow:alpha
    environment:
      - ANTHROPIC_API_KEY=${ANTHROPIC_API_KEY}
    volumes:
      - ./swarms:/app/swarms
      - ./src:/app/src
      - ./.swarm:/app/.swarm
    command: >
      swarm create
      --preset /app/swarms/python-development.json
      --task "Develop API endpoints"
      --monitor
    ports:
      - "3001:3001"  # Dashboard
    networks:
      - development

  postgres:
    image: postgres:15
    environment:
      - POSTGRES_DB=appdb
      - POSTGRES_USER=appuser
      - POSTGRES_PASSWORD=secret
    networks:
      - development

  redis:
    image: redis:7-alpine
    networks:
      - development

networks:
  development:
    driver: bridge
```

Makefile Integration

makefile

```
# Makefile
.PHONY: swarm-init swarm-dev swarm-test swarm-deploy

PRESET = ./swarms/python-development.json
TASK = "Develop new feature"

swarm-init:
  npx claude-flow@alpha init --sparc --force
  npx claude-flow@alpha preset install $(PRESET)

swarm-dev:
  npx claude-flow@alpha swarm create \
    --preset python-development \
    --task $(TASK) \
    --max-agents 8 \
    --monitor

swarm-test:
  npx claude-flow@alpha swarm create \
    --preset python-development \
    --task "Write comprehensive tests" \
    --agents test-engineer,performance-tester \
    --parallel

swarm-deploy:
  npx claude-flow@alpha swarm create \
    --preset python-development \
    --task "Prepare production deployment" \
    --agents infrastructure-engineer,security-engineer,devops-lead \
    --strategy sequential

swarm-clean:
  rm -rf .swarm/ .hive-mind/
  npx claude-flow@alpha memory clear --all
```

Best Practices

1. Preset-Design

- **Modular**: Erstelle kleine, wiederverwendbare Presets
- **Spezifisch**: Optimiere für konkrete Use Cases
- **Dokumentiert**: Füge klare Beschreibungen und Metadaten hinzu

2. Agent-Konfiguration

- **Klare Rollen**: Definiere eindeutige Verantwortlichkeiten
- **Richtige Größe**: 5-8 Agenten für die meisten Projekte
- **Hierarchie**: Nutze Lead-Agenten für Koordination

3. Memory-Management

- **Namespaces**: Trenne Kontexte logisch
- **Kompression**: Nutze für große Projekte
- **Persistenz**: Aktiviere für langlebige Projekte

4. Performance-Optimierung

- **Parallelisierung**: Wo möglich parallel arbeiten
- **Caching**: Wiederholte Operationen cachen
- **Auto-Scaling**: Für variable Workloads

5. Sicherheit

- **Secrets**: Niemals in Presets hardcoden
- **Permissions**: Minimale notwendige Rechte
- **Audit**: Logging für alle kritischen Operationen

Zusammenfassung

Swarm-Presets in Claude-Flow ermöglichen es, komplexe Entwicklungsteams mit einem einzigen Befehl zu orchestrieren. Durch die JSON-basierte Konfiguration können Teams:

1. **Wiederverwendbare Team-Strukturen** definieren
2. **Spezialisierte Agenten** für verschiedene Aufgaben einsetzen
3. **Komplexe Workflows** automatisieren
4. **Best Practices** standardisieren
5. **Entwicklungszeit** drastisch reduzieren

Die Flexibilität des Systems erlaubt es, von einfachen 3-Agenten-Teams bis zu komplexen Enterprise-Strukturen mit 20+ Agenten zu skalieren, wobei jeder Agent seine spezifischen Tools, Prompts und Arbeitsverzeichnisse hat.

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