Claude Code Architektur-Analyse

Was der Source-Leak über Anthropics KI-Entwicklungstool verrät

1.903 Dateien · 512.000+ Zeilen TypeScript · 43 Native Tools

Quellcode-Snapshot vom 31. März 2026 (npm Source-Map Exposure)
Analyse: Politech Dynamics – Research & Architecture Review

Politech Dynamics UG (haftungsbeschränkt)

www.politech-dynamics.de

Über diese Analyse

Politech Dynamics ist ein GovTech-Unternehmen aus Erlangen, das KI-gestützte Lösungen für politische Kommunikation und Verwaltungsdigitalisierung entwickelt. Unsere gesamte Softwareentwicklung und Unternehmenssteuerung basiert auf einem selbst entwickelten KI-Agenten-Framework — mit Claude Code als zentralem Werkzeug.

Als am 31. März 2026 der vollständige Quellcode von Claude Code über eine ungesicherte Source-Map im npm-Paket öffentlich wurde, haben wir die Gelegenheit genutzt: Unser Research-Team hat die 1.903 Dateien und über 512.000 Zeilen TypeScript systematisch analysiert — von der Architektur über das Tool-System bis hin zu den Sicherheitsmechanismen.

Was Sie in diesem Whitepaper erfahren

- **Wie der Leak passierte** — npm Source-Maps, ungeschützte R2-Buckets, und was das für Supply-Chain-Security bedeutet
- **Die Architektur eines 500K-Zeilen CLI-Tools** — React im Terminal, Generator-basiertes Streaming, 6-Schichten-Modell
- **43 Native Tools + unbegrenzte Erweiterungen** — wie das Tool-System mit Zod-Schemas, Permission-Kaskade und Deferred Loading funktioniert
- **Multi-Agent-Orchestrierung** — Sub-Agents, Teams, Coordinator Mode, Worktree-Isolation — die heißesten Patterns im AI-Engineering
- **MCP-Client von innen** — 7 Transport-Typen, OAuth-PKCE, Server-Verwaltung als React State
- **Sicherheitsanalyse** — 2 CVEs, Angriffsvektoren, das Hook-basierte Permission-System und seine Grenzen
- **Persistentes Memory-System** — Auto-Extraktion, Relevanz-Recall via Sonnet, Team Memory Sync

> **Für wen ist dieses Whitepaper?**
> Dieses Dokument richtet sich an Software-Architekten, AI-Engineers, CTOs und technische Entscheider, die verstehen wollen, wie ein State-of-the-Art KI-Entwicklungstool von innen aufgebaut ist — und welche Architektur-Patterns sich auf eigene Projekte übertragen lassen.

Hinweis: Diese Analyse basiert auf einem öffentlich zugänglich gewordenen Quellcode-Snapshot. Der Original-Code ist Eigentum von Anthropic. Die Analyse dient Forschungs- und Bildungszwecken.

Erstellt mithilfe von Claude Code (Anthropic) und Codex (OpenAI). Die Ergebnisse wurden durch mehrere KI-gestützte Analyse-Agents parallel erarbeitet und anschließend manuell geprüft.

Inhaltsverzeichnis

1. Executive Summary

Claude Code ist Anthropics offizielles CLI-Tool für KI-gestützte Software-Entwicklung. Es ermöglicht Entwicklern, direkt aus dem Terminal mit Claude zu interagieren — Dateien zu bearbeiten, Shell-Befehle auszuführen, Codebases zu durchsuchen und komplexe Workflows zu orchestrieren. Dieses Whitepaper dokumentiert die Architektur basierend auf einer systematischen Quellcode-Analyse des am 31.03.2026 öffentlich gewordenen Source-Snapshots.

1.903	512k+	43	89	6
Dateien	Zeilen Code	Native Tools	Slash-Commands	Schichten

Kerntechnologien:

Kategorie	Technologie
Runtime	Bun (nicht Node.js)
Sprache	TypeScript (strict mode)
Terminal-UI	React + Ink (deklaratives Terminal-Rendering)
CLI-Parsing	Commander.js (extra-typings)
Schema-Validierung	Zod v4
Code-Suche	ripgrep (eingebettet)
Protokolle	MCP SDK, LSP, OAuth 2.0, JWT
API	Anthropic SDK (Streaming, Tool-Calls)
Telemetrie	OpenTelemetry + gRPC
Feature-Flags	GrowthBook (Dead-Code-Elimination via Bun)

2. Der Leak: Was passiert ist

Am 31. März 2026 veröffentlichte Security-Researcher Chaofan Shou (@Fried_rice) auf Twitter/X einen bemerkenswerten Fund: Das npm-Paket von Claude Code enthielt Source-Maps (.map-Dateien), die auf unobfuskierten TypeScript-Quellcode in einem Amazon R2-Bucket von Anthropic verwiesen. Der gesamte Quellcode — 1.903 Dateien, über 512.000 Zeilen — war frei herunterladbar.

Was exponiert wurde

- Vollständiger TypeScript-Quellcode des Claude Code CLI
- Interne Architektur: Tool-System, Permission-Modell, Multi-Agent-Framework
- Feature-Flags und internes Scheduling-System (Kairos)
- System-Prompt-Templates und Context-Assembly-Logik
- MCP-Client-Implementierung mit allen 7 Transport-Typen
- Sicherheitsmechanismen inklusive bekannter Schwachstellen

2.1 Technischer Hintergrund

Source-Maps sind ein Standard-Mechanismus in JavaScript/TypeScript-Toolchains. Sie ermöglichen Debugging, indem sie kompilierten Code auf den Quellcode zurückführen. Normalerweise werden sie vor der Veröffentlichung entfernt. In diesem Fall blieben die .map-Dateien im npm-Paket und verwiesen auf eine öffentlich zugrängliche URL, unter der Anthropic die Source-Dateien hostete.

2.2 Supply-Chain-Implikationen

Der Vorfall illustriert ein wiederkehrendes Problem in der JavaScript-Ökosystem-Sicherheit: Build-Artefakte, die sensible Informationen enthalten, werden versehentlich veröffentlicht. Dies betrifft nicht nur Source-Maps, sondern auch .env-Dateien, interne Konfigurationen und Debug-Symbole. Für Unternehmen, die eigene npm-Pakete veröffentlichen, ist eine systematische Pre-Publish-Prüfung unverzichtbar.

2.3 Was die Analyse zeigt

Die Offenlegung ermöglicht erstmals einen tiefen Blick in die Architektur eines kommerziellen KI-Entwicklungswerkzeugs. Während Open-Source-Alternativen (Aider, Continue, Cursor) ihren Code öffentlich machen, war Claude Code bisher eine Black Box. Die folgenden Kapitel dokumentieren, was wir gefunden haben.

3. Technologie-Stack & Kennzahlen

Claude Code nutzt eine moderne, auf Bun optimierte TypeScript-Architektur. Bemerkenswert ist der Einsatz von React/Ink für Terminal-UIs — ein Paradigma, das deklaratives UI-Rendering in die Kommandozeile bringt.

3.1 Dateistatistik

Dateityp	Anzahl	Anteil/Größe
TypeScript (.ts)	1.332	69,9%
React/TSX (.tsx)	552	29,0%
JavaScript Stubs (.js)	18	0,9%
Dokumentation (.md)	1	0,1%
Gesamt	1.903	~34 MB

3.2 Verzeichnisstruktur

Verzeichnis	Inhalt	Zweck
src/tools/	~40 Tool-Implementierungen	Kern des Agenten-Systems
src/commands/	~50 Slash-Commands	User-facing Befehle
src/components/	Über 340 UI-Komponenten	React/Ink Terminal-Rendering
src/services/	API, MCP, OAuth, Analytics	Externe Integrationen
src/hooks/	Permission, Lifecycle	Event-System
src/coordinator/	Multi-Agent	Agent-Orchestrierung
src/skills/	Skill-System	Wiederverwendbare Workflows
src/memdir/	Persistent Memory	MEMORY.md-System
src/bridge/	IDE-Integration	VS Code, JetBrains
src/query/	Query Pipeline	LLM-Interaktion
src/tasks/	Task Management	Agent-Aufgabenverwaltung
src/state/	State Management	Globaler Zustand
src/remote/	Remote Sessions	Entfernte Ausführung
src/server/	Server Mode	Headless-Betrieb
src/vim/	Vim Mode	Vi-Keybindings
src/voice/	Voice Input	Spracheingabe

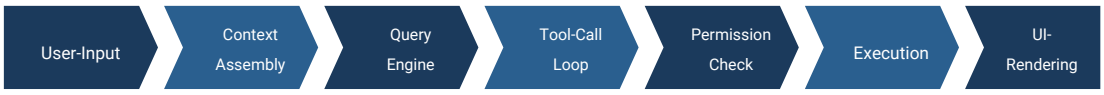
3.3 Build-System & Feature-Flags

Claude Code nutzt Buns `bun:bundle` Feature-Flag-System für Dead-Code-Elimination zur Build-Zeit. Inaktive Features werden komplett aus dem Bundle entfernt — kein Runtime-Overhead.

Flag	Beschreibung
PROACTIVE	Proaktiver Modus (Agent handelt selbstständig)
KAIROS	Internes Scheduling-System
BRIDGE_MODE	IDE-Integration aktiv
DAEMON	Hintergrund-Daemon-Betrieb
VOICE_MODE	Spracheingabe-Unterstützung
AGENT_TRIGGERS	Cron-/Remote-Trigger für Agents
MONITOR_TOOL	System-Monitoring-Tool

4. Architektur-Übersicht

Die Architektur folgt einem Schichtenmodell mit klarer Trennung zwischen CLI-Parsing, LLM-Interaktion, Tool-Ausführung und UI-Rendering.



4.1 Schichtenmodell

Schicht	Kern-Module	Verantwortung
1. CLI Layer	main.tsx, Commander.js	Argument-Parsing, Entrypoint, React/Ink Bootstrap
2. Context Layer	context.ts, CLAUDE.md, System-Prompt	Sammelt Systeminfo, Git-Status, Projekt-Kontext, User-Preferences
3. Query Layer	QueryEngine.ts (~rund 2.000 Zeilen)	LLM-API-Calls, Streaming, Tool-Call-Loops, Retry, Token-Counting
4. Tool Layer	Tool.ts, src/tools/	~40 Tools mit Zod-Schemas, Permission-Modell, Execution-Logic
5. Service Layer	src/services/	API-Client, MCP, OAuth, Analytics, Memory-Extraction, Compact
6. UI Layer	src/components/, React/Ink	Über 340 Komponenten für Terminal-Rendering, Markdown, Diff-Views

4.2 Startup-Sequenz

Claude Code optimiert die Startzeit durch paralleles Prefetching:

#	Phase	Details
1	Side-Effect Imports	MDM-Settings-Read + Keychain-Prefetch + API-Preconnect starten parallel
2	Commander.js Parse	CLI-Argumente werden geparst, Subcommand identifiziert
3	React/Ink Mount	Terminal-UI wird initialisiert, Input-Handler registriert
4	Context Assembly	Git-Status, CLAUDE.md, Projekt-Infos werden gesammelt
5	QueryEngine Init	LLM-Verbindung steht, Tool-Registry geladen
6	Ready	User-Prompt wird akzeptiert

> **Schneller Start trotz 34 MB Code**
> Claude Code startet zwei Optimierungen gleichzeitig: Parallel Prefetch — Einstellungen, Keychain-Zugriff und API-Verbindung werden noch vor dem eigentlichen Programmstart parallel angefragt. Lazy Loading — Große Bibliotheken (Telemetrie, Analytics, optionale Features) werden erst geladen, wenn sie tatsächlich gebraucht werden. Beides zusammen sorgt dafür, dass das CLI trotz seiner Größe in Sekunden bereit ist.

5. Tool-System

Das Tool-System ist das Herzstück von Claude Code. 43 native Tools plus unbegrenzte MCP-Erweiterungen bilden ein typsicheres Framework mit Zod-Schemas, mehrstufigem Permission-Modell und Hook-Unterstützung. Die Architektur trennt sauber was Tools tun (Interface), wie sie aufgerufen werden (Lifecycle) und wer sie nutzen darf (Permissions).

5.1 Tool-Katalog (43 Native Tools)

Kategorie	Tools	Beschreibung
Dateisystem (5)	FileRead, FileEdit, FileWrite, Glob, Grep	Datei-I/O und Suche
Shell (3)	Bash, PowerShell, REPL	Befehlsausführung mit Sandbox
Agentur (7)	Agent, Skill, TeamCreate/Delete, SendMessage, TaskCreate, RemoteTrigger	Multi-Agent-Orchestrierung
Web (2)	WebFetch, WebSearch	Internet-Zugriff
Protokoll (3)	MCPTool, LSPTool, ReadMcpResource	MCP + Language Server
Plan/Context (5)	EnterPlanMode, ExitPlanMode, EnterWorktree, ExitWorktree, ToolSearch	Modus-Wechsel
Scheduling (4)	CronCreate, CronDelete, CronList, Sleep	Timer und Cron-Jobs
Sonstige (14)	AskUser, Config, Brief, Snip, Monitor, Workflow, Notebook, ...	Spezialfunktionen

5.2 Tool-Lifecycle (6 Phasen)

Jeder Tool-Aufruf durchläuft sechs Phasen:

Phase	Was passiert
1. Registrierung	Das Tool wird mit seinem Schema und Standardverhalten im System angemeldet
2. Eingabe-Prüfung	Die Parameter werden gegen das definierte Schema validiert
3. Berechtigungsprüfung	8-stufige Regelkaskade entscheidet: erlauben, ablehnen oder nachfragen
4. Ausführung	Die eigentliche Tool-Logik läuft asynchron mit Fortschrittsanzeige
5. Ergebnis-Verarbeitung	Große Ausgaben werden ausgelagert, das Modell erhält eine Vorschau
6. Darstellung	Das Ergebnis wird im Terminal gerendert (7 verschiedene Anzeige-Varianten)

5.3 Permission-Modell (5 Modi, 8 Rule Sources)

Jeder Tool-Aufruf durchläuft das Permission-System (src/hooks/toolPermission/) mit einer Kaskade aus 8 Quellen:

Modus	Verhalten
default	User wird bei unbekannten Actions gefragt (Standard)
plan	Nur Lese-Operationen; Schreiben erfordert Plan-Approval
auto	ML-Classifier entscheidet — bei 'high confidence' auto-allow
acceptEdits	Datei-Edits automatisch erlaubt, Rest nach Bedarf

Modus	Verhalten
bypassPermissions	Alle Checks deaktiviert, nur für interne Tests und CI/CD

Rule Sources (Kaskade — spätere Quellen überschreiben frühere):

5.4 Design-Patterns

Das Tool-System nutzt mehrere Architektur-Patterns, die Startup-Zeit, Token-Verbrauch und Sicherheit optimieren:

Pattern	Was es bewirkt
Minimale Tool-Definition	Jedes Tool definiert nur seine Besonderheiten. Standardverhalten (Berechtigungen, UI, Fehlerbehandlung) wird automatisch ergänzt.
Verzögertes Laden	Tool-Schemas werden erst geladen, wenn sie tatsächlich gebraucht werden — das spart Startzeit und Token im KI-Context-Window.
Automatische Sicherheitsprüfung	Ein ML-Modell bewertet Tool-Aufrufe vorab als sicher oder unsicher. Nur bei Unsicherheit wird der Nutzer gefragt.
Parallelitäts-Deklaration	Jedes Tool gibt an, ob es parallel mit anderen laufen kann. Im Zweifel wird serialisiert (sicherer Standard).
Große Ergebnisse auslagern	Ausgaben über 20 KB werden auf die Festplatte geschrieben. Das KI-Modell erhält nur eine Vorschau und kann bei Bedarf nachlesen.

> **Verzögertes Tool-Loading spart Token**
> Tools werden zunächst nur mit Name und Kurzbeschreibung registriert. Die vollständige Schema-Definition wird erst nachgeladen, wenn das Tool tatsächlich aufgerufen wird. Bei über 40 nativen und zahlreichen externen MCP-Tools reduziert das den Token-Verbrauch erheblich.

6. Skill-System

Skills sind wiederverwendbare, konfigurierbare Workflows, die über den SkillTool aufgerufen werden. Sie ermöglichen es, häufige Aufgaben (Commit, Review, Config) in standardisierte Abläufe zu kapseln.

6.1 Skill-Architektur

Ein Skill besteht aus einer Markdown-Datei (SKILL.md) mit optionalem Frontmatter, Trigger-Beschreibung und Prompt-Template. Skills können project-lokal oder global definiert werden:

Komponente	Beschreibung
SKILL.md	Prompt-Template mit Anweisungen für Claude
config.json	Optional: Trigger-Regeln, Parameter-Schema
scripts/	Optional: Helper-Scripts die der Skill nutzt
Frontmatter	Name, Beschreibung, Trigger-Condition im YAML-Header

6.2 Built-in Skills (Auszug)

Skill	Beschreibung
/commit	Git-Commit mit konventioneller Message erstellen
/review	Code-Review der aktuellen Änderungen
/compact	Konversation komprimieren (Context sparen)
/config	Settings verwalten
/doctor	Umgebungs-Diagnose
/memory	Persistenten Speicher verwalten
/skills	Skill-Übersicht und -Management
/tasks	Aufgabenverwaltung

7. Memory-System (memdir)

Das Memory-System (memdir) ermöglicht persistente Erinnerungen über Konversationsgrenzen hinweg. Es speichert strukturierte Markdown-Dateien mit YAML-Frontmatter in einem dedizierten Verzeichnis.

7.1 Speicherstruktur

Memories werden als einzelne .md-Dateien in ~/.claude/projects/<project>/memory/ gespeichert. Eine MEMORY.md-Indexdatei verweist auf alle Memory-Dateien.

Typ	Inhalt	Zweck
user	Rolle, Ziele, Präferenzen des Nutzers	Antworten personalisieren
feedback	Korrekturen und bestätigte Ansätze	Gleiche Fehler vermeiden
project	Laufende Arbeit, Entscheidungen, Deadlines	Kontext verstehen
reference	Externe Ressourcen, Links, Systeme	Richtige Quellen finden

7.2 Memory-Format

Jede Memory-Datei folgt einem einheitlichen Format:

```
---
name: memory_name
description: Einzeiler für Relevanz-Matching
type: user|feedback|project|reference
---

Memory-Inhalt (Markdown)
```

7.3 Automatische Wissensextraktion

Nach jeder abgeschlossenen Konversation startet im Hintergrund ein spezialisierter Agent, der die letzten Nachrichten auf speichernswertes Wissen durchsucht. Dieser Agent arbeitet mit eingeschränkten Rechten: Er darf Dateien lesen und durchsuchen, aber nur im Memory-Verzeichnis schreiben. Falls der Hauptagent bereits selbst Erinnerungen gespeichert hat, wird die automatische Extraktion übersprungen, damit keine Duplikate entstehen.

7.4 Intelligenter Abruf

Das Inhaltsverzeichnis aller Erinnerungen wird bei jedem Start geladen. Zusätzlich wählt ein KI-Modell automatisch bis zu fünf thematisch passende Erinnerungsdateien aus — basierend auf der aktuellen Frage des Nutzers. So erhält Claude immer den relevantesten Kontext, ohne dass alle Erinnerungen gleichzeitig geladen werden müssen.

7.5 Geteiltes Wissen im Team

Wenn mehrere Agents zusammenarbeiten, können sie Erinnerungen über ein gemeinsames Verzeichnis teilen. Die Synchronisation funktioniert in drei Schritten:

- **Hochladen:** Nur geänderte Dateien werden übertragen. Vor jedem Upload prüft ein Scanner, ob versehentlich Passwörter oder API-Schlüssel in den Erinnerungen stehen.
- **Herunterladen:** Der Client fragt zuerst, ob sich überhaupt etwas geändert hat. Nur bei tatsächlichen Änderungen werden Daten übertragen.
- **Automatisch:** Lokale Änderungen lösen nach kurzer Verzögerung automatisch einen Upload aus.

> **Warum das Memory-System so relevant ist**
>
> Das Memory-System ist eines der am meisten unterschätzten Features moderner KI-Entwicklungswerkzeuge. Claude Code zeigt, wie persistente Erinnerungen über Sessions hinweg die Entwicklerproduktivität dramatisch steigern können — insbesondere die Auto-Extraktion und der Relevanz-basierte Recall.

8. Multi-Agent & Coordinator

Claude Code implementiert ein ausgereiftes Multi-Agent-System, das es ermöglicht, Sub-Agents für parallele Aufgaben zu spawnen, in Teams zu organisieren und über ein Messaging-System zu koordinieren.

8.1 Agent-Spawning (AgentTool)

Der AgentTool startet eigenständige Sub-Agents mit eigener Konversation, eigenem Context-Window und optionaler Worktree-Isolation. Jeder Agent erhält eine Task-Beschreibung und arbeitet autonom.

Parameter	Beschreibung
prompt	Aufgabenbeschreibung für den Agent
description	Kurze Zusammenfassung (3-5 Wörter)
subagent_type	Spezialisierung (Explore, Plan, general-purpose, ...)
run_in_background	Agent läuft asynchron im Hintergrund
isolation	worktree = isolierte Repo-Kopie
model	Optional: Model-Override (sonnet, opus, haiku)

8.2 Agent-Typen

Typ	Zweck	Verfügbare Tools
general-purpose	Standard-Agent für komplexe Aufgaben	Alle Tools
Explore	Schnelle Codebase-Exploration	Read-only Tools
Plan	Architektur-Planung	Read-only + Plan-Output
project-reviewer	Code-Review im Projektkontext	Alle Tools

8.3 Team-System (Swarm-Modell)

Ein Team besteht aus einem Lead-Agent und mehreren Teammates. TeamFile speichert Mitglieder mit agentId, tmuxPaneId, cwd, worktreePath und Subscriptions. Teammates können in-process (AsyncLocalStorage-isoliert) oder als separate Prozesse (Tmux) laufen.

8.4 Inter-Agent-Kommunikation

Mechanismus	Beschreibung
SendMessage (direkt)	Running Agent: Message in pendingMessages-Queue
SendMessage (Mailbox)	File-basierte Inbox (JSON-Dateien pro Agent)
Broadcast (to: '*')	Nachricht an alle Non-Lead Teammates
Structured Messages	shutdown_request/response, plan_approval_response
Task-Notifications	XML-Format bei Agent-Completion an Koordinator

8.5 Coordinator Mode

Im Coordinator-Mode orchestriert ein Master-Agent Worker über 4 Phasen: Research (parallel Explore) Synthesis (Koordinator versteht Problem) Implementation (Worker ändern Code) Verification (Worker testen). Workers haben keinen Zugriff auf Team/Message-Tools — nur der Koordinator.

> **Multi-Agent: Der heißeste Trend im AI-Engineering**
> Die Multi-Agent-Architektur von Claude Code ist eine der fortschrittlichsten Implementierungen in einem kommerziellen Produkt. Das 4-Phasen-Coordinator-Modell (Research Synthesis Implementation Verification) ist ein Pattern, das sich auf nahezu jedes AI-Agent-Framework übertragen lässt.

9. MCP-Client-Architektur

Claude Code implementiert einen vollständigen MCP-Client (Model Context Protocol) mit 7 Transport-Typen, OAuth-PKCE-Auth, und React-Context-basiertem Server-Management. Dies ermöglicht die Erweiterung um beliebige domänenspezifische Tools.

9.1 Transport-Layer (7 Typen)

Transport	Beschreibung	Anwendung
stdio	Child-Prozess mit stdin/stdout	Standard, .mcp.json
SSE	Server-Sent Events (HTTP)	Cloud-basierte Server
SSE-IDE	IDE-spezifische SSE-Variante	VS Code Extension
HTTP	REST-basiert (StreamableHTTP)	Neuere Server
WebSocket	Bidirektional, persistent	Real-time Server
SDK	In-Process (kein externer Prozess)	Embedded Server
claudeai-proxy	Claude.ai Proxy-Server	Remote Claude.ai

9.2 Tool Discovery & Normalisierung

Beim Start verbindet sich Claude Code zu allen konfigurierten MCP-Servern und fragt deren verfügbare Tools ab. Die Tool-Namen werden automatisch mit dem Servernamen als Präfix versehen, um Namenskonflikte zu vermeiden. Die Ergebnisse werden zwischengespeichert, sodass wiederholte Abfragen schnell sind. Jedes externe MCP-Tool durchläuft dieselbe Berechtigungsprüfung wie die eingebauten Tools.

9.3 OAuth für MCP-Server

MCP-Server können OAuth-Authentifizierung erfordern. Der ClaudeAuthProvider implementiert Authorization-Code mit PKCE, Cross-App-Access (XAA), und automatischen Token-Refresh. Auth-Status wird 15 Minuten gecachet; 401-Fehler setzen den Server in 'needs-auth'-Status.

9.4 Server-Verwaltung (React Context)

MCPConnectionContext verwaltet alle Server-Verbindungen als React State: reconnect, toggle (enable/disable), Status-Tracking (connected / pending / failed / needs-auth / disabled). Agent-spezifische MCP-Server werden additiv zu Parent-Servern geladen und beim Agent-Finish automatisch aufgeräumt.

10. Hook- & Permission-System

Das Hook-System ist der zentrale Mechanismus für Sicherheit und Erweiterbarkeit. 12+ Hook-Events fangen den gesamten Lifecycle ab — von Tool-Aufrufen über Session-Management bis zu Agent-Spawning. Hooks können Shell-Befehle ausführen, Tool-Calls blockieren oder Input/Output modifizieren.

10.1 Hook-Events (13 Typen)

Event	Zeitpunkt	Typische Nutzung
PreToolUse	Vor Tool-Ausführung	Permission, Logging, Input-Modifikation
PostToolUse	Nach Tool-Erfolg	Result-Validation, Audit
PostToolUseFailure	Nach Tool-Fehler	Error-Handling, Retry-Logik
PermissionRequest	User-Bestätigung nötig	Auto-Allow/Deny-Regeln
UserPromptSubmit	User-Eingabe verarbeitet	Input-Filtering, -Modifikation
SessionStart	Session-Initialisierung	Setup-Befehle, Context-Loading
SessionEnd	Session-Beendigung	Cleanup, State-Persist
PreCompact	Vor Context-Komprimierung	Custom Komprimierungs-Regeln
PostCompact	Nach Komprimierung	Nachbearbeitung
SubagentStart/Stop	Agent-Lifecycle	Agent-spezifischer Context
Notification	Verschiedene Events	Custom Alerting
Elicitation	MCP-Server fragt User	Accept/Decline/Cancel
Setup	Projekt-Init/Maintenance	Auto-Setup-Scripts

10.2 Wie Hooks ausgeführt werden

Hooks laufen als eigene Prozesse. Sie erhalten ihre Eingabe als JSON und signalisieren das Ergebnis über ihren Exit-Code:

- **Code 0** — Erfolg: Die Ausgabe des Hooks wird verarbeitet
- **Code 2** — Blockierend: Der Tool-Aufruf wird abgebrochen
- **Andere Codes** — Fehler wird angezeigt, die Verarbeitung läuft weiter

Hooks können aus drei Quellen stammen: Shell-Befehle in den Einstellungen, Plugin-Hooks und Skill-Hooks.

10.3 Hook-Output-Möglichkeiten

Output-Feld	Beschreibung
permissionBehavior	ask / deny / allow / passthrough
updatedInput	Modifizierte Tool-Eingabe (Input-Rewriting)
updatedMCPToolOutput	Modifizierte MCP-Tool-Ausgabe
blockingError	Blockierender Fehler-JSON
preventContinuation	Weitere Verarbeitung stoppen
elicitationResponse	Antwort auf MCP-Elicitation

> **Sicherheits-Warnung: Hook-basierte Angriffe (CVE-2025-59536 / CVE-2026-21852)**
> Bösartige .claude/settings.json in geklonten Repos können Hooks definieren, die bei Tool-Aufrufen beliebigen Code ausführen — inklusive API-Key-Exfiltration über manipulierte ANTHROPIC_BASE_URL. Claude Code erfordert Trust-Bestätigung beim ersten Öffnen, aber der Trust-Prompt konnte umgangen werden (gepatcht).

11. QueryEngine & LLM-Integration

Mit rund rund 2.000 Zeilen ist die QueryEngine das größte Modul und das Herzstück der KI-Kommunikation. Sie steuert den gesamten Dialog zwischen Nutzer und Sprachmodell: Nachrichten senden, Antworten empfangen, Tool-Aufrufe ausführen und Ergebnisse zurückspielen.

11.1 Kern-Funktionen

Die QueryEngine löst acht zentrale Aufgaben:

Funktion	Was sie bewirkt
Echtzeit-Streaming	Antworten zeichenweise angezeigt, nicht erst nach Fertigstellung
Tool-Aufruf-Schleife	Claude fordert Tool an, Engine führt aus, Ergebnis fließt zurück
Adaptives Denken	Interner Denkprozess vor der Antwort, automatisch bei komplexen Aufgaben
Fehlerbehandlung	Automatische Wiederholung mit steigenden Wartezeiten, Modell-Fallback
Token-Zählung	Echtzeit-Zählung pro Modell, getrennt nach Ein- und Ausgabe
Komprimierung	Bei vollem Kontext werden ältere Nachrichten automatisch zusammengefasst
Vorausladen	Einstellungen und Erinnerungen werden parallel im Hintergrund geladen
Kostenerfassung	API-Kosten pro Modell und Sitzung, wiederherstellbar bei Fortsetzung

11.2 Context Assembly

Der System-Prompt wird dynamisch zusammengebaut aus:

Quelle	Inhalt
Basis-Instruktionen	Kern-Verhalten, Sicherheitsregeln, Tool-Nutzung
CLAUDE.md	Projekt-spezifische Instruktionen (alle CLAUDE.md im Pfad)
MEMORY.md	Persistierte Erinnerungen aus dem Memory-System
Git-Status	Aktueller Branch, Änderungen, letzte Commits
Environment	OS, Shell, Plattform, Working Directory
MCP-Tools	Verfügbare MCP-Server-Tools (Name + Schema)
System-Reminders	Dynamisch injizierte Hinweise (Skill-Kontext, etc.)

12. Command-System

Slash-Commands sind Befehle, die der Nutzer mit einem Schrägstrich aufruft (z.B. `/commit` oder `/review`). Claude Code bringt rund 89 eingebaute Commands mit, ergänzt durch Skills, Plugins und Workflow-Skripte. Es gibt drei Typen: Befehle die einen KI-Prompt auslösen, local-jsx (React/Ink UI-Komponenten).

Command	Beschreibung	Details
/commit	Git-Commit erstellen	Analyse, Message, Stage, Commit
/review	Code-Review	Diff-Analyse, Findings, Suggestions
/compact	Context komprimieren	Alte Nachrichten zusammenfassen
/config	Settings verwalten	JSON-basiert, Global + Projekt
/doctor	Umgebungs-Diagnose	Dependencies, Permissions, APIs
/login / /logout	Authentifizierung	OAuth 2.0 Flow
/memory	Memory verwalten	List, Add, Remove, Search
/skills	Skills verwalten	List, Add, Enable/Disable
/tasks	Tasks anzeigen	Status, Progress, History
/vim	Vim-Modus	Toggle Vi-Keybindings
/diff	Änderungen anzeigen	Git Diff mit Syntax-Highlighting
/cost	Kosten prüfen	Token-Usage, API-Kosten
/context	Context visualisieren	Token-Budget, Tool-Liste
/pr_comments	PR-Kommentare	GitHub PR Review
/resume	Session fortsetzen	Letzte Konversation laden

13. UI-Layer (React/Ink)

Claude Code nutzt React mit Ink für deklaratives Terminal-Rendering. Über 340 Komponenten in `src/components/` bilden die UI-Schicht — von Markdown-Rendering über Diff-Views bis zu interaktiven Permission-Dialogen. Dies ist eine der ungewöhnlichsten Architekturentscheidungen: ein vollständiges React-Komponentensystem innerhalb eines Terminal-Tools.

13.1 Kern-Komponenten (Auszug)

Komponente	Beschreibung
MessageStream	Streaming-Anzeige der LLM-Antwort
ToolCallView	Visualisierung eines Tool-Aufrufs + Ergebnis
PermissionPrompt	Interaktiver Permission-Dialog (Allow/Deny/Always)
DiffView	Syntax-highlighted Git-Diff-Anzeige
MarkdownRenderer	Markdown Terminal (Syntax-Highlighting, Tables)
TaskList	Fortschrittsanzeige der Agent-Tasks
CostDisplay	Token-Usage und Kosten-Anzeige
InputBox	User-Input mit Keybindings (Emacs/Vim)

13.2 State Management

Der globale Zustand wird über `src/state/` verwaltet. Konversationsverlauf, Tool-Ergebnisse, Berechtigungsstatus und Nutzereinstellungen werden zentral gespeichert und automatisch an alle UI-Komponenten weitergegeben, wenn sich etwas ändert.

14. Bridge & IDE-Integration

Das Bridge-System verbindet IDE-Extensions (VS Code, JetBrains) bidirektional mit dem Claude Code CLI. Die Kommunikation nutzt JWT-basierte Authentifizierung und ein Nachrichten-Protokoll.

Modul	Beschreibung
bridgeMain.ts	Bridge-Hauptloop, Event-Handling
bridgeMessaging.ts	Nachrichten-Protokoll (JSON-basiert)
bridgePermissionCallbacks.ts	Permission-Anfragen an die IDE delegieren
jwtUtils.ts	JWT-Token-Generierung und -Validierung
sessionRunner.ts	Session-Lifecycle-Management
replBridge.ts	REPL-Session-Brücke
trustedDevice.ts	Geräte-Vertrauensstellung

15. Services & Infrastruktur

Claude Code gliedert seine Backend-Dienste in zwölf spezialisierte Module:

Dienst	Aufgabe
API-Client	Kommunikation mit der Anthropic-API
MCP-Client	Anbindung externer Tool-Server (siehe Kapitel 9)
Authentifizierung	Login und Logout über OAuth 2.0
Code-Analyse	Anbindung an Language Server für Code-Intelligence
Telemetrie	Feature Flags und Nutzungsstatistiken
Plugin-System	Laden von Erweiterungen (siehe 15.1)
Komprimierung	Zusammenfassung alter Nachrichten bei vollem Kontext
Richtlinien	Nutzungsbeschränkungen für Unternehmenskunden
Fernkonfiguration	Zentrale Verwaltung von Einstellungen (MDM)
Memory-Extraktion	Automatisches Speichern von Wissen (siehe Kapitel 7)
Team-Memory	Synchronisation von Erinnerungen zwischen Agents
Token-Schätzung	Berechnung des Kontextverbrauchs pro Nachricht

15.1 Plugin-System

Das Plugin-System (src/plugins/) ermöglicht Third-Party-Erweiterungen. Plugins werden über eine Registry geladen und können eigene Tools, Commands und UI-Komponenten registrieren.

15.2 Telemetrie & Feature Flags

Analytics basiert auf GrowthBook (Feature Flags + A/B Testing). OpenTelemetry + gRPC liefern Performance-Metriken. Feature Flags steuern Feature-Rollouts und sind zur Build-Zeit in Dead-Code-Elimination integriert.

16. Sicherheitsanalyse

Bekannte Schwachstellen (CVEs)

CVE-2025-59536: Code-Injection über untrusted project hooks (.claude/settings.json) bei erstem Öffnen eines Repositories — Bypass der Trust-Prompt-Logik.

CVE-2026-21852: API-Key-Exfiltration über manipulierte ANTHROPIC_BASE_URL in Repository-Settings — API-Requests vor Trust-Prompt.

16.1 Sicherheitsarchitektur

Mechanismus	Beschreibung	Effektivität
Permission System	Jeder Tool-Call durchläuft Permission-Checks	HOCH
Trust Prompt	Neue Projekte erfordern explizite Vertrauensbestätigung	MITTEL (umgehbar, siehe CVEs)
Bash Sandbox	Shell-Befehle in Sandbox mit Timeout und Allowlist	HOCH
Path Validation	Dateizugriffe werden gegen Projekt-Root validiert	HOCH
URL Validation	WebFetch prüft URLs gegen Blocklist	MITTEL
JWT Auth	Bridge-Kommunikation ist JWT-gesichert	HOCH
Hook Isolation	Hooks laufen in eigener Shell, kein Memory-Sharing	MITTEL
Worktree Isolation	Agents können in isolierten Repo-Kopien arbeiten	HOCH

16.2 Angriffsvektoren

Vektor	Beschreibung	Mitigation
Malicious .claude/	Bösartige Settings/Hooks in geklonten Repos	Trust-Prompt (gepatcht)
MCP Server Poisoning	Kompromittierter MCP-Server liefert bösartige Tools	Server-Validierung
Prompt Injection	Tool-Results enthalten Injection-Versuche	System-Prompt-Warnung
ANTHROPIC_BASE_URL	Env-Variable leitet API-Traffic um	Gepatcht in CVE-2026-21852
Dependency Confusion	Bösartige npm-Pakete im Build	Package-Lock, Audit

Claude Code implementiert eine mehrschichtige Sicherheitsarchitektur, die für die meisten Szenarien ausreichend ist. Die beiden CVEs zeigen jedoch, dass insbesondere das Trust-Modell für Repository-Einstellungen anfällig war — ein kritischer Bereich, da Entwickler regelmässig fremde Repositories klonen. Beide Schwachstellen wurden mittlerweile gepatcht.

Über Politech Dynamics

Politech Dynamics ist ein GovTech-Unternehmen aus Erlangen, das KI-gestützte Lösungen für politische Kommunikation, Verwaltungsdigitalisierung und demokratische Teilhabe entwickelt.

Bereich	Beschreibung
KI-Agent-Frameworks	Autonome KI-Agenten für Unternehmenssteuerung und Softwareentwicklung
Infrastructure Automation	Ansible, Docker, Monitoring — alles als Code
GovTech-Software	Livestreaming, Ratsinformationssysteme, Bürgerbeteiligung
KI-Beratung	Workshops, Seminare und Implementierungsberatung

Politech Dynamics UG (haftungsbeschränkt)

Stoke-on-Trent-Str. 1, 91058 Erlangen

Web: www.politech-dynamics.de

E-Mail: fischbach@politech-dynamics.de

Telefon: +49 9131 8875 677

Geschäftsführer: Matthias Fischbach

Stand: 01.04.2026 | Amtsgericht Fürth, HRB 21840

POLITECH — DYNAMICS —