

Build Your Own Agent with the GitHub Copilot SDK and Microsoft Foundry



Kurz zu mir: Matthias Liebeck

- Senior Solution Architect in Düsseldorf, Schwerpunkt .NET, C# und Azure
- Software-Entwickler seit 2009, C# seit 2006
- 2009 – 2015: Studium Informatik & Mathematik
- 2018: Promotion in Informatik über Natural Language Processing / KI
- Organisator des Azure Düsseldorf Meetups, Speaker über Azure & agentic coding
- GitHub Copilot Newsletter: ghcp.liebeck.io



Worum es heute geht

AGENDA

01 Was ist ein Agent?

Wir einigen uns auf ein Kriterium mit zwei Fragen.

02 Das GitHub Copilot SDK

Was es ist, was Leute damit bauen, was es alles kann.

03 QueueScout

Was ich konkret damit gebaut habe.

04 Gedächtnis

Wo sich Agent und Crawler trennen, mit Blick in die Forschung.

05 Hosting

Microsoft Foundry (BYOK), Feedback, Nachtbetrieb in Azure.

06 Fazit

Agent oder Crawler? Und wann SDK, wann Microsoft Agent Framework?

1

Mit dem GitHub Copilot SDK kann man Agenten bauen. Nicht nur Coding-Agenten.

2

Ich habe einen gebaut. Das Repo ist offen. Ihr könnt ihn heute Abend nachbauen.

3

Ich zeige, wie ich das Ganze in Azure gehostet habe. Seitdem läuft er jede Nacht, ohne mich.

◆ K A P I T E L

01

Was ist ein Agent?

Wir einigen uns auf ein Kriterium mit zwei Fragen.

Ich habe ChatGPT gefragt, was ein KI-Agent ist.

Mehrere Nachfragen, mehrere verschiedene Antworten. Drei davon wörtlich:

„...nimmt Informationen aus seiner Umgebung auf, leitet daraus Entscheidungen ab und führt anschließend Aktionen aus.“

„...kombiniert ein KI-Modell mit Werkzeugen, Datenquellen und einer Entscheidungslogik, um Aufgaben autonom oder teilautonom zu bearbeiten.“

„...arbeitet in einer Schleife aus Wahrnehmung, Schlussfolgerung und Aktion, bis ein gewünschter Zustand erreicht ist.“

(Und irgendwo da draußen gilt natürlich auch: „Alles, wo KI drinsteckt.“)

Was in allen Antworten steht

Immer: ein Ziel, Entscheidungen, Aktionen.

Meistens: die Umgebung · manchmal: die Schleife · selten: Werkzeuge.

In keiner einzigen:

Gedächtnis

Auf Ziel, Entscheidung und Aktion einigen sich alle. Ob das System sich erinnert, fragt niemand. Genau da wird es interessant.

Unser Arbeitskriterium für heute

1

Wer bestimmt zur Laufzeit den nächsten Schritt: mein Code oder das Modell?

Ein Workflow ist vorher entschieden. Ein Agent entscheidet unterwegs.

2

Verändert der heutige Lauf den morgigen?

Ohne Gedächtnis startet jeder Lauf identisch, egal wie klug er ist.

Wir bauen jetzt ein System und stellen diese zwei Fragen dreimal im Laufe des Vortrags.

◆ K A P I T E L

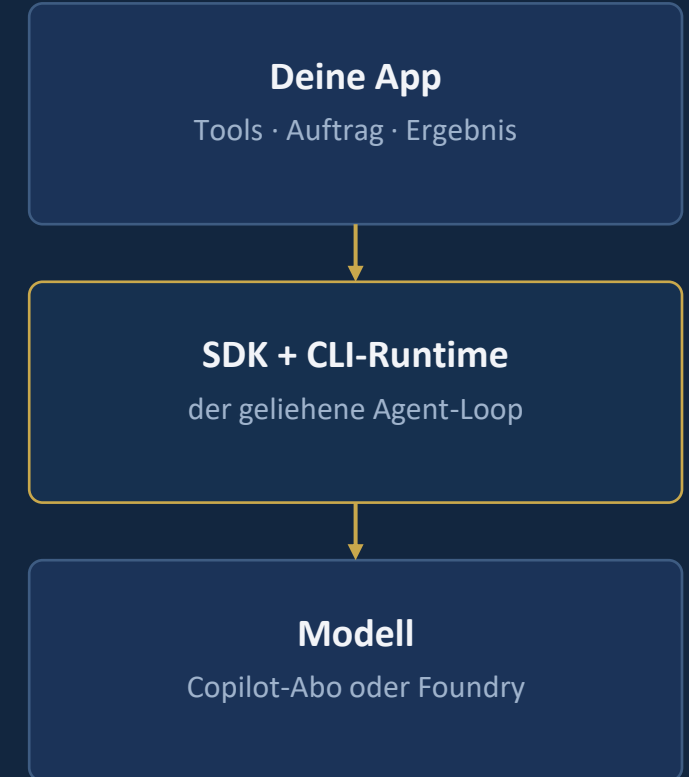
02

Das GitHub Copilot SDK

Die Engine hinter Copilot als Bibliothek.

Das SDK

- **Die Engine hinter Copilot CLI und Copilot App, als Bibliothek:** Agent-Loop, Tool-Dispatch, Retries, Streaming und Session-Verwaltung sind schon drin.
- **Sechs Sprachen:** .NET, Python, TypeScript, Go, Java, Rust.
Heute zeige ich .NET, die Konzepte sind identisch.
- **Die CLI wird mitgebündelt** und läuft unsichtbar im Server-Modus; euer Code spricht per JSON-RPC mit ihr.
- **Für Einsteiger:** Copilot CLI = GitHub Copilot fürs Terminal, gleiche Engine wie in IDE und App.



Ihr schreibt keinen Orchestrierungs-Code. Ihr leiht euch den Loop, den Millionen Entwickler täglich benutzen.

Was Leute damit bauen

Aus Blogs, Samples und dem offiziellen Cookbook — was andere mit dem SDK gebaut haben:

- **GitHub selbst nennt:** internes Code-Analyse-Tool, Release-Notes-Generator, Agent im Support-Workflow, alle auf derselben Grundlage statt auf einem Eigenbau-Stack.
- **Ralph Loop (offizielles .NET-Cookbook):** autonome Coding-Schleife mit frischer Session pro Iteration. Der Plan liegt als Datei auf der Platte, Tests wirken als Backpressure.
- **Interview Coach (Azure-Sample):** App auf Basis des Microsoft Agent Framework, mit Blazor-UI und MCP-Servern, in der Copilot einer von drei austauschbaren LLM-Providern ist, neben Foundry und Azure OpenAI.
- **SRE-Agent aus der Community:** reagiert auf fehlgeschlagene GitHub Actions, holt Logs per MCP und entscheidet zwischen Retry und Issue.
- **Und meins gleich im Detail:** ein Recherche-Agent für ein digitales Warteschlangen-Produkt.

Der Unterschied zwischen all diesen Agenten ist nie das SDK. Es sind immer nur: welche Tools, welcher Trigger, wohin mit dem Ergebnis.

Was das SDK sonst noch kann

Ich nutze davon fast nichts. Damit ihr wisst, was da noch liegt:

Hooks

Tool-Aufrufe abfangen, verändern oder ablehnen. Pre- und Post-Tool-Use.

Fleet Mode

Parallele Sub-Agents für Arbeit, die sich aufteilen lässt.

Cloud Sessions

Läufe auf GitHub-gehostetem Compute statt im eigenen Container.

Remote Sessions

Dieselbe Session auf GitHub Web und Mobile weiterverfolgen. Mission Control.

Entra / Managed Identity

Für BYOK offiziell dokumentiert. Von mir ungetestet.

Weitere Features

MCP-Server, Custom Agents, Plugin Directories, Image Input, OpenTelemetry.

◆ K A P I T E L

03

QueueScout

Was ich konkret damit gebaut habe.

Kurz zum Produkt: SmartWarten

- **Mein eigenes SaaS.** Gäste ziehen ihr Ticket per QR-Code und sehen auf dem Handy, wie viele noch vor ihnen dran sind.
- **Der Sinn:** niemand steht mehr in der Schlange. Man wartet, wo man will, und bekommt Bescheid.
- **Im Einsatz dort, wo Menschen vor Ort anstehen:** Ämter, Praxen, Läden, Veranstaltungen.
- **Daraus folgt eine Dauerfrage:** Wo ist Warten gerade ein Problem? Welche Branchen, welche Anlässe, welche Regionen?



Wer ein Produkt für digitale Warteschlangen baut, muss wissen, wo gerade gewartet wird.

Die Aufgabe: QueueScout

- **Der Auftrag:** täglich deutsche Nachrichten danach absuchen, wo Menschen vor Ort warten oder sich darüber beschweren.
- **Zielquellen:** Berichte über Behörden, Notaufnahmen, Neueröffnungen und Veranstaltungen.
- **Bisher hieß das:** selbst suchen, unregelmäßig, in Wellen. Und genau dann nicht, wenn gerade viel zu tun ist.
- **Nicht relevant, aber sprachlich zum Verwechseln ähnlich:** Online-Ticket-Warteschlangen, virtuelle Warteräume von Shops, Message Queues in Software-Architektur
- **Anforderung:** täglich laufen, nur Neues melden, per Mail, ohne dass jemand zuschaut.

Keyword-Suche scheitert hier zwangsläufig: „Warteschlange“ steht in relevanten und irrelevanten Artikeln. Die Bedeutung entscheidet.

So sieht das Ergebnis aus

Paderborn: Das Warten im Regen hat ein Ende

Westfalen-Blatt | 10.09.2026

Besucher der Ausländerabteilung am Hoppenhof in Paderborn müssen wegen einer Zugangsbeschränkung am Eingang vor dem Gebäude warten, bevor sie abgeholt werden. Der Außenwartebereich bietet laut Kritikern keinen ausreichenden Wetterschutz; der Schalter ist nur an zwei Tagen pro Woche geöffnet.

Relevant, weil: Menschen bei einer Ausländerbehörde im Freien warten müssen und ein begrenzter Zugang beziehungsweise wenige Öffnungstage die Besuchersteuerung zu einem konkreten Vor-Ort-Problem machen.

Gefunden über: Ausländerbehörde Wartezeit

Relevant?  

- **Jeder Fund:** Titel, Quelle, Datum, Link und eine Begründung: „Relevant, weil: Menschen warten vor Ort, spontane Besuche kaum möglich.“
- **Die Begründung ist Pflicht im Auftrag.** Sie macht Fehlentscheidungen sichtbar, statt sie im Ranking zu verstecken.
- **Die „Gefunden über“-Zeile** zeigt, welche Suche den Treffer lieferte.
- **Die zwei Daumen** sind der Anfang des Feedback-Kreises. Dazu kommen wir in Kapitel 05.

Der ganze Agent ist eine Session

```
// CopilotSetup.cs
return new CopilotClient(new CopilotClientOptions
{
    Mode = CopilotClientMode.Empty,
    BaseDirectory = config.CopilotHome,
    GitHubToken = config.UsesFoundry ? null : config.CopilotToken,
});
// AgentRunner.cs
await using var session = await client.CreateSessionAsync(new SessionConfig
{
    Model = config.Model,
    Provider = CopilotSetup.Provider(config),
    SessionLimits = CopilotSetup.Limits(config),
    SystemMessage = new SystemMessageConfig { Mode = SystemMessageMode.Append, Content = instructions },
    Tools = BuildTools(telemetry),
    AvailableTools =[.. ToolNames],
    OnPermissionRequest = CopilotSetup.DenyAll(log),
    Streaming = false,
});
await session.SendAsync(new MessageOptions { Prompt = task });
```

Der SDK-Anteil eines Agenten ist klein. Die Arbeit steckt im Drumherum, und das ist normale Software.

Ein Tool ist ein Lambda

```
CopilotTool.DefineTool(  
    async (string url) =>  
    {  
        telemetry.RecordToolCall();  
        if (Refusal(telemetry) is { } refusal) return refusal;  
  
        log.WriteLine($"[tool] fetch_page: {url}");  
        return await fetcher.FetchAsync(url);  
    },  
    toolOptions: new CopilotToolOptions { SkipPermission = true, Defer = CopilotToolDefer.Never },  
    factoryOptions: new AIFunctionFactoryOptions  
    {  
        Name = "fetch_page",  
        Description = "Laedt eine Webseite und gibt ihren Text zurueck. "  
            + "Fehler beginnen mit FEHLER:, zu wenig Text (Paywall) mit HINWEIS:.",  
    },  
),
```

- **Name und Description sind das komplette Interface zum Modell.** Die Beschreibung entscheidet, wann das Modell das Tool ruft.
- **Fehler kommen als Text zurück, nicht als Exception.** Der Agent soll damit umgehen, statt abzustürzen.

Der Agent liest fremde Webseiten

Ein Agent, der beliebige Webinhalte liest, ist eine Prompt-Injection-Angriffsfläche. Deshalb:

- 1 **Empty-Modus:** keine Shell, kein Dateizugriff. Der Standardmodus hätte beides gehabt, automatisch genehmigt
- 2 **Tool-Allowlist:** exakt `search_web` und `fetch_page`, sonst nichts
- 3 **Permissions:** jede Anfrage der Runtime wird pauschal abgelehnt
- 4 **Garantien im Code statt im Prompt:** Dedup, Kostendeckel, Suchbudget

Nebeneffekt, gemessen: Die eingebauten Tools kosten auch Kontext: 12.855 statt 3.757 Input-Tokens pro Modellaufruf (Standard vs. Empty).

Gemessen: identischer Auftrag, gpt-5.6-luna, 13.09.2026

Der Loop ist geliehen, die Kontrollpunkte gehören mir.

Was ich vorgegeben habe, und warum

- **Feste Suchliste statt freier Formulierung:** queries.md hält getestete Suchbegriffe je Kategorie, pro Lauf rotiert eine Auswahl durch. Im Lauf vom 3. Oktober waren das fünf Suchen, drei lead und zwei trend.
- **Der Grund steht als Kommentar im Code:** Sich selbst überlassen schreibt ein Sprachmodell lange Suchbegriffe mit Jahreszahlen und Füllwörtern, auf die eine Nachrichtensuche nichts zurückgibt.
- **Zwei eigene Suchen pro Lauf bleiben erlaubt (`MAX_OWN_QUERIES=2`).** Alles, was nicht in queries.md steht, zählt als eigene Suche. Die dritte lehnt mein Code ab, nicht der Prompt:

FEHLER: Du hast bereits 2 eigene Suchen genutzt. Führe die vorgegebenen Suchbegriffe wortgleich aus.

- **Ebenfalls vorgegeben:** Ausgabeformat (reines JSON), Pflicht-Begründung je Fund, Relevanz-Definition (nächste Folie).

Entscheidungen dorthin verschieben, wo sie gut getroffen werden. Autonomie ist kein Wert an sich. Sie ist ein Budget.

Der Auftrag ist das Spec

- **Relevanz-Definition in beide Richtungen:**
 - **physisches Warten → ja** Behörden, Praxen, Gastronomie, Events
 - **digitale Queues → nein** Ticket-Shops, virtuelle Warteräume, Message Queues
- **Grenzfälle explizit entschieden, nicht dem Modell überlassen:**
 - **virtueller Warteraum vor physischem Event → relevant**
 - **Callcenter-Wartezeiten → nicht relevant**
- **Je zwei echte Beispiele pro Kategorie.** Beispiele stabilisieren die Entscheidung stärker als jede abstrakte Regel.
- **Pflicht-Begründung je Fund** („Relevant, weil ...“) kostet fast nichts und macht jede Fehlentscheidung sichtbar.

Aus meiner NLP-Zeit: 2016 wäre das ein Klassifikator mit gelabelten Trainingsdaten gewesen. Heute: vier Absätze Prompt plus Daumen-Feedback. Und ich bin froh darüber.

💡 *Grenzfälle, die ihr nicht entscheidet, entscheidet das Modell. Jede Woche anders.*

Eine Nacht, nacherzählt

>	10/3/2026, 3:00:46.573 AM	1790996445.26678	Agent reported 4 candidate(s), 2 new.
>	10/3/2026, 3:00:46.573 AM	1790996445.26678	lead: 2
>	10/3/2026, 3:00:46.573 AM	1790996445.26678	[notes] notes.md replaced (19 line(s)).
>	10/3/2026, 3:00:34.455 AM	1790996433.25654	[tool] fetch_page: https://www.mz.de/lokal/koethen/wartezeiten-und-argen-welche-verbesserungen-es-bei
>	10/3/2026, 3:00:34.455 AM	1790996433.25654	[tool] fetch_page: https://www.suedkurier.de/schwarzwald/zulassungsstellen-schwarzwald-baar-lange-wart
>	10/3/2026, 3:00:34.455 AM	1790996433.25655	[tool] fetch_page: https://www.augsburger-allgemeine.de/neu-ulm/nach-aerger-ueber-lange-wartezeit-so-
>	10/3/2026, 3:00:34.455 AM	1790996433.25655	[tool] fetch_page: https://osthessen-news.de/n11800765/lange-schlange-vor-eroeffnung-riesiger-andrang-
>	10/3/2026, 3:00:34.455 AM	1790996433.25655	[tool] fetch_page: https://www.thueringer-allgemeine.de/lokales/weimar-und-weimarer-land/article413171
>	10/3/2026, 3:00:34.455 AM	1790996433.75607	[fetch] https://www.suedkurier.de/schwarzwald/zulassungsstellen-schwarzwald-baar-lange-wartezeiten-dur
>	10/3/2026, 3:00:34.455 AM	1790996433.75607	[fetch] https://www.thueringer-allgemeine.de/lokales/weimar-und-weimarer-land/article413171428/lange-s
>	10/3/2026, 3:00:34.455 AM	1790996433.75608	[fetch] https://www.augsburger-allgemeine.de/neu-ulm/nach-aerger-ueber-lange-wartezeit-so-sollen-buer
>	10/3/2026, 3:00:34.455 AM	1790996433.75608	[tool] search_web [own] Köthen Zulassungsstelle Wartezeit -> 1 hit(s)
>	10/3/2026, 3:00:34.455 AM	1790996433.75608	[tool] search_web [own] digitale Wartemarke Behörde -> 0 hit(s)
>	10/3/2026, 3:00:30.999 AM	1790996429.25735	[tool] search_web [lead] Neueröffnung Ansturm Schlange -> 10 hit(s)
>	10/3/2026, 3:00:29.811 AM	1790996428.7563	[tool] search_web [trend] Wartenummer digital -> 3 hit(s)
>	10/3/2026, 3:00:29.811 AM	1790996428.7563	[tool] search_web [lead] Zulassungsstelle Wartezeit -> 10 hit(s)
>	10/3/2026, 3:00:29.811 AM	1790996428.7563	[tool] search_web [trend] virtuelle Warteschlange -> 7 hit(s)
>	10/3/2026, 3:00:29.811 AM	1790996428.7563	[tool] search_web [lead] Bürgeramt Wartezeit -> 0 hit(s)
>	10/3/2026, 3:00:25.511 AM	1790996424.75642	Memory: 35 known finding(s), notes.md 20 line(s).
>	10/3/2026, 3:00:25.511 AM	1790996424.75643	Feedback: 8 vote(s), 8 used as examples.
>	10/3/2026, 3:00:25.511 AM	1790996424.75643	Search plan: 5 queries (lead 3, trend 2), up to 2 own, rotation step 23.
>	10/3/2026, 3:00:24.814 AM	1790996423.76464	[schedule] 05:00 in Europe/Berlin - running.
>	10/3/2026, 3:00:24.814 AM	1790996423.76464	Model: gpt-5.6-luna via Microsoft Foundry deployment (BYOK)
>	10/3/2026, 3:00:24.814 AM	1790996423.76464	Storage: Azure blob container queuescout

- **03:00 UTC:** Cron startet den Container, der Agent prüft die Zeitzone und läuft, weil es in Berlin 05:00 ist.
- **Suchplan:** 5 vorgegebene Suchen (3 lead, 2 trend), Rotationsschritt 23, dazu bis zu 2 eigene.
- **Gedächtnis beim Start:** 35 bekannte Funde, notes.md mit 20 Zeilen, 8 meiner Bewertungen als Beispiele.
- **Der Lauf:** 3 Modellaufrufe, 12 Tool-Aufrufe. 7 Suchen, davon 2 eigene. 5 Seiten geladen, 3 davon dünn wegen Paywall oder Zustimmungsdialo.
- **Das Ergebnis:** 4 Kandidaten, davon 2 neu. Bericht ins Blob, Mail raus, notes.md mit 19 Zeilen neu geschrieben.
- **Kosten:** 0,0387€, Obergrenze 5,00€ pro Lauf. AI Credits 0,000, genau wie unter BYOK erwartet.

Echter Lauf vom 03.10.2026, Log aus Application Insights

Stufe 1 • Ein Lauf, von Start bis Mail

Open Source (MIT) • läuft lokal im Docker-Container • Modell: Copilot-Abo oder Foundry



Scoreboard, erster Blick

1

Wer bestimmt zur Laufzeit den nächsten Schritt?

geteilt

Das Modell: welche Treffer es liest, was relevant ist, wann es nachsucht, plus zwei eigene Suchen.
Mein Code: Suchliste, Rotation, Dedup, Mail-Entscheidung, Kostendeckel.

2

Verändert der heutige Lauf den morgigen?

Nein

Jeder Lauf startet mit identischem Wissen. Was er gestern gelernt hat, ist heute weg.

Das ehrliche Zwischenfazit, und der Einwand, den ihr gerade denkt:

„Das ist ein sehr guter Crawler mit Urteilsvermögen.“ Stimmt. Noch.

◆ K A P I T E L

04

Gedächtnis

Wo sich Agent und Crawler trennen.

Warum ein Gedächtnis nicht optional ist

Stellt euch meinen Agenten ohne Gedächtnis vor. Klug genug ist er ja. Drei Dinge passieren:

Er meldet dieselben Funde wieder

Jede Nacht dieselben fünf Artikel. Nach drei Tagen lese ich die Mail nicht mehr. Und ein Agent, dessen Ergebnis niemand liest, ist abgeschaltet.

Er macht denselben Fehler wieder

Was gestern danebenlag, liegt heute wieder daneben. Ohne Spur gibt es nichts, woraus er oder ich lernen könnten.

Meine Korrekturen verpuffen

Ich kann noch so oft „nicht relevant“ sagen: Wenn es nirgendwo landet, ist es mit dem Ende der Session weg.

Und der Punkt, der oft untergeht: Kontext ist kein Gedächtnis. Der Kontext endet mit der Session. Alles, was länger leben soll, muss jemand bewusst wegschreiben. Das ist eine Architekturentscheidung, kein Modell-Feature.

💡 *Ein Agent ohne Gedächtnis kann klug sein. Besser werden kann er nicht.*

Drei Dateien, drei Besitzer

findings.json

schreibt mein Code

Alles je Gemeldete, Basis der Dedup. Garantie: keine Doppel-Mails. Garantien gehören in Code, nicht in den Prompt.

notes.md

schreibt der Agent

Gelernte Regeln, max. 40 Zeilen, verdichten statt anhängen, über seine Endausgabe statt per Datei-Tool.

feedback.json

schreibe ich

Ein Daumen je Fund, direkt aus der E-Mail. Schlägt alles andere. Wie genau, später in diesem Kapitel.

Dazu **runs.jsonl**: schreibt mein Code, liest der Mensch: Tokens, Kosten, Treffer je Lauf.



Wer schreiben darf, ist die wichtigste Memory-Entscheidung, wichtiger als das Speicherformat.

Was der Agent sich selbst notiert

notes.md, Stand 02.10.2026 (Auszug)

- Warten im Freien, volle Wartebereiche, Einlassstopps und limitierte Wartemarken sind direkte Leads.
- Reine Fan- oder Produkt-Hypes sowie Verkaufswarteschlangen bei Produktstarts nicht melden.
- Funde aus den Suchbegriffen "Wartezimmer überfüllt", "Wartezeit online" und "Aufrufsystem" waren bereits gemeldete Themen oder nicht ausreichend neu.

37

Funde gesamt

27 Leads · 10 Trends

35

verschiedene Quellen

von BR bis Radio K.W.

11

Suchbegriffe mit Treffern

Spitzenreiter: Zulassungsstelle
Wartezeit mit 10 Funden

20

Zeilen notes.md

jede Nacht komplett neu
geschrieben

- **Die letzte Zeile ist die interessanteste:** Der Agent merkt sich inzwischen, welche seiner eigenen Suchbegriffe nichts Neues mehr bringen.
- **Der Verlauf ist versioniert:** lokal per Git-Commit nach jedem Lauf, in der Cloud per Blob-Versionierung. Ich kann jede Nacht nachlesen, was er gelernt zu haben glaubt.

Zahlen aus findings.json und notes.md, Stand 03.10.2026

Selbstverwaltetes Gedächtnis ist ein Drift-Mechanismus ohne eingebaute Richtung. Die Historie macht sie sichtbar.

Was sagt die Forschung?

Die Frage: Ein Agent erledigt eine Aufgabe und notiert sich danach, was er gelernt hat. Reicht dieser eine Durchlauf als Beleg dafür, dass die Notiz stimmt?

Die Antwort: Fast. Die Trefferquote stimmt, aber sie schwankt stark. Erst wenn ein zweiter Agent die Notiz vor dem Speichern in der echten Umgebung nachprüft, wird das Ergebnis verlässlich.

- **Der Test:** 40 Datenbank-Fragen, bei denen nach Frage 20 das Schema umgebaut wird. Genau der Fall, in dem eine alte Notiz zur Falle wird.
- **Ohne Gedächtnis:** 39 % gelöst. Mit Notizen aus dem eigenen Lauf: 70 %, aber von Durchlauf zu Durchlauf ± 16 Punkte.
- **Mit Prüfung vor dem Speichern:** 73 % und nur noch ± 5 Punkte, bei halbierten Kosten pro Frage (3,38 $\rightarrow$ 1,68 \$).

Suresh, Mak, Bhatnagar, Methani, Gutierrez Munoz (Microsoft): „Grounding Agent Memory“ · arXiv 2609.11060 · Benchmark CLBench, 40 Fragen mit Schema-Wechsel, GPT-5.4, fünf Durchläufe je Variante · die vier Varianten sind derselbe Agent, gebaut mit dem GitHub Copilot SDK

Das Gedächtnis bringt den Sprung. Das Prüfen bringt die Verlässlichkeit.

Zurück zu QueueScout: zwei Kuratoren

Soweit die Forschung. Ich habe keinen zweiten Agenten gebaut. Ich bin der zweite Kurator.

- **Der Agent kuratiert sich selbst, jede Nacht:** liest notes.md, verdichtet, verwirft, schreibt neu. Das läuft auch ohne mich.
- **Ich bin die Korrekturinstanz:** Daumen hoch / runter je Fund, direkt aus der Mail.
- **Daumen hoch = „auf Ähnliches achten“ · Daumen runter = „nicht mehr melden“.** Aus jeder Stimme destilliert der Agent eine Regel in seine eigene notes.md. So überlebt sie das Fenster der letzten 20 Bewertungen, das der Agent überhaupt zu sehen bekommt.

Menschliche Stimme

schlägt

Agent-Urteil

schlägt

Notizen (notes.md)

schlägt

Instructions

Die Forschung baut einen zweiten Agenten als Kurator. Meine Version kostet zwei Klicks am Tag und hat immer recht.

Stufe 2 • Der Relevance-Feedback-Kreis

läuft jede Nacht in Azure • der Mensch steuert mit zwei Klicks am Tag



Zwei Daumen, eine Regel

Was aus einem einzigen Klick wird, nachvollzogen über vier notes.md-Stände:

17.09.

Fund: „Vorab einchecken: Neues System im Klinikum Lüdenscheid“, vom Agenten als trend gemeldet.



Ich klicke Daumen runter

21.09.

„Vorab-Tickets, QR-Check-ins und Wartezeit-Tracking in Kliniken sind nach der negativen Bewertung des Lüdenscheid-Funds nur bei belastbarem Nachweis relevant.“



Der Agent macht eine Regel daraus

28.09.

„Technische Kliniklösungen ohne belastbaren konkreten Nutzen nicht als trend melden.“ Der Ortsname ist verschwunden.



Die Regel löst sich vom Anlass

02.10.

„Technische Kliniklösungen wie QR-Check-ins, digitale Tickets oder Wartezeit-Tracking nicht als Trend melden, wenn kein belastbarer konkreter Nutzen nachgewiesen ist.“



Sie gilt bis heute

Der zweite Daumen, 30 Sekunden früher: 🖱️ Pokémon-Hype (BR, 17.09.) → „Reine Fan- oder Produkt-Hypes sowie Verkaufswarteschlangen bei Produktstarts nicht melden.“ — die Regel aus dem notes.md-Auszug von vorhin.

Ein Klick am 17. September entscheidet mit, was der Agent heute Nacht meldet.

Der Aufräumtag und was er gekostet hat

29.09.: Der Agent verdichtet seine Notizen von 24 auf 14 Zeilen. Niemand hat ihn darum gebeten.

raus

Außenwartebereiche vor Ausländerbehörden sind auch ohne konkrete Wartezeitangabe relevante Leads.

raus

Regionale Medien wie WAZ, Westfalen-Blatt, Merkur und buten un binnen liefern konkrete Vor-Ort-Warteprobleme.

raus

Breite Suchbegriffe liefern häufig keine Treffer; eigene Suchen sollten Ort und Warteproblem kombinieren.

neu

Eine konkrete Online-Terminvergabe mit kurzfristig freien Terminen kann als trend gelten.

neu

Die Leserbewertung bestätigt konkrete, wiederkehrende Schlangen und fehlende Behördentermine als besonders wertvolle Leads.
(01.10., nach meinem Daumen hoch auf Ludwigshafen)

Die erste rote Zeile war genau die Regel, die den Paderborn-Fund vom Anfang erklärt hat. Sie ist weg, eine allgemeinere hat ihren Platz genommen.

Verdichten ist nicht Vergessen. Aber ohne Historie merkt man den Unterschied nicht.

Scoreboard, zweiter Blick

1

Wer bestimmt zur Laufzeit den nächsten Schritt?

unverändert

Geteilt wie vorher: Das Modell urteilt, mein Code garantiert.

2

Verändert der heutige Lauf den morgigen?

Ja, dreifach

`findings.json`

nie wieder doppelt
melden

`notes.md`

gelernte Regeln prägen
den Auftrag

`feedback.json`

mein Veto schlägt alles

Offen benannt: Das Selbstverstärkungs-Risiko bleibt: eine falsche Notiz beeinflusst Folgeäufe. Deshalb Versionierung, deshalb das Veto.

Jetzt wird es schwer, das noch Crawler zu nennen. Der Unterschied war nicht mehr Intelligenz. Es war Gedächtnis.

◆ K A P I T E L

05

Hosting

Microsoft Foundry, Feedback und Nachtbetrieb in Azure.

Wessen Modell?

Derselbe Codestand kann beide Wege. Der Lauf, dessen Log ihr vorhin gesehen habt, ging schon über Foundry — daher die null AI Credits. Was die beiden Wege unterscheidet:

	GitHub Copilot-Abo	Microsoft Foundry (BYOK)
Wer hostet das Modell	GitHub	ich, in meiner Subscription und Region
Abrechnung	AI Credits aus dem Abo	pay per token auf der Azure-Rechnung
Authentifizierung	fine-grained PAT	API-Key, Entra / Managed Identity möglich
Wohin gehen die Artikeltexte	zu GitHub	bleiben im eigenen Tenant

Gleich bleiben: Code, Agenten-Loop, Tools und Gedächtnis. Der Wechsel ist eine Umgebungsvariable. · Ehrlich dazu: Serper als US-Dienst sieht die Suchanfragen in beiden Pfaden.

Der Modellwechsel ist keine Migration, sondern nur eine Konfigurationssache.

BYOK in vier Zeilen

```
MODEL_PROVIDER=foundry           # der Schalter: copilot | foundry
FOUNDRY_ENDPOINT=https://...
FOUNDRY_API_KEY=...               # Key-basiert; Entra: BearerTokenProvider
FOUNDRY_DEPLOYMENT=gpt-5-6-luna
FOUNDRY_WIRE_API=responses        # der Stolperstein, siehe unten
```

- **Der Stolperstein:** Die GPT-5-Familie spricht die Responses-API. Wer WireApi nicht setzt, sieht kryptische Fehler und sucht am falschen Ende.
- **Auth ist bei mir Key-basiert.** Für Entra ID / Managed Identity gibt es inzwischen einen offiziellen Setup-Guide (BYOK mit Azure AI Foundry). Von mir ungetestet, deshalb nur als Verweis.
- **Im Code:** eine ProviderConfig

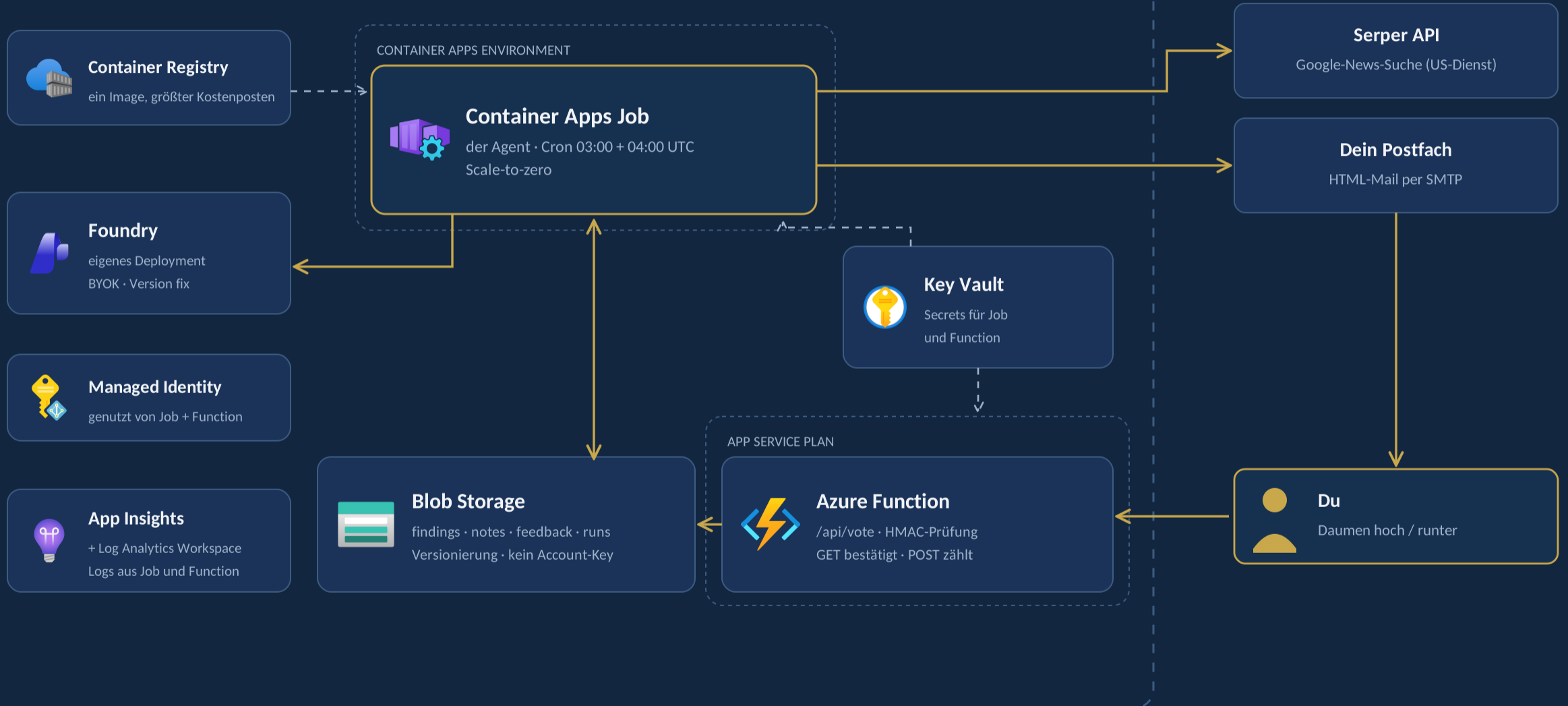
Wann lohnt der eigene Tenant?

- 1 **Datenhoheit:** Artikeltexte und Prompts bleiben in der eigenen Subscription und Region. Für manche Teams die einzige erlaubte Option.
- 2 **Version einfrieren:** Abo-Modelle können sich serverseitig ändern. Das eigene Deployment liegt fest. Wer misst und vergleicht, braucht genau das.
- 3 **Kostenmodell:** pay per token auf der Azure-Rechnung statt AI Credits: eigener Vertrag, eigene Kostenkontrolle.
- 4 **Modellwahl:** der Foundry-Katalog reicht über die Abo-Modelle hinaus.

BYOK ist kein Feature-Upgrade, sondern ein Kontroll-Upgrade. Ob ihr das braucht, ist eine Compliance-Frage, keine technische.

Stufe 2 • Architektur in Azure

AZURE • EIGENER TENANT • ZEHN RESSOURCEN, ALLES ALS BICEP



Datenfluss zur Laufzeit

Image und Secrets, über die Managed Identity

ca. 5 Euro im Monat • keine Datenbank, kein Server

Nachts laufen

Name ▾	Status ▾	Duration ▾	Start time (UTC) ▾
queuescout-job-29850000	✓ Succeeded	30 seconds	2026-10-03 04:00:00
queuescout-job-29849940	✓ Succeeded	48 seconds	2026-10-03 03:00:00
queuescout-job-29848560	✓ Succeeded	32 seconds	2026-10-02 04:00:00
queuescout-job-29848500	✓ Succeeded	45 seconds	2026-10-02 03:00:00
queuescout-job-29847120	✓ Succeeded	28 seconds	2026-10-01 04:00:00
queuescout-job-29847060	✓ Succeeded	41 seconds	2026-10-01 03:00:00
queuescout-job-29845680	✓ Succeeded	31 seconds	2026-09-30 04:00:00
queuescout-job-29845620	✓ Succeeded	49 seconds	2026-09-30 03:00:00
queuescout-job-29844240	✓ Succeeded	27 seconds	2026-09-29 04:00:00
queuescout-job-29844180	✓ Succeeded	55 seconds	2026-09-29 03:00:00
queuescout-job-29842800	✓ Succeeded	29 seconds	2026-09-28 04:00:00
queuescout-job-29842740	✓ Succeeded	52 seconds	2026-09-28 03:00:00

Ausführungshistorie des Container Apps Jobs

- **Azure Container Apps Job mit Cron-Trigger:** Container startet, arbeitet, verschwindet. Beahlt wird die Laufminute, ein Lauf dauert 30 bis 55 Sekunden.
- **Zweimal nachts, einmal gearbeitet:** Cron rechnet in UTC, deshalb starten 03:00 und 04:00 UTC. Der Agent läuft nur, wenn es in Berlin 05:00 ist.
- **Azure Function nimmt die Daumen-Klicks entgegen:** GET zeigt die Bestätigungsseite, erst POST zählt, weil Mail-Scanner Links anklicken.
- **Sicherheit:** Managed Identities überall, Storage ohne Account-Key, Secrets im Key Vault. Alles als Bicep, inklusive Resource Group.
- **Kosten:** rund 5 Euro im Monat, fast alles davon Container Registry.

Läuft seit dem 16.09.2026 jede Nacht. Am Vortragstag sind das exakt drei Wochen.

Job für geplante Arbeit, Function für Ereignisse, Blob für Zustand. Kein Server, keine Datenbank, kein Babysitting.

◆ K A P I T E L

06

Fazit

Agent oder Crawler? Und wann SDK, wann Microsoft Agent Framework?

Scoreboard, Finale

Vorgegeben

mein Code

Suchliste + Rotation · Relevanz-Definition mit
Grenzfällen · Dedup-Garantie · Mail-
Entscheidung · Kostendeckel

Delegiert

das Modell

lesen · urteilen · begründen · notieren · 2
eigene Suchen pro Nacht

Erinnert

das System

jeden Fund · jede gelernte Regel · jedes Veto
von mir

Antwort auf die Eingangsfrage, nach unserem Kriterium: **Ja, ein Agent. Er entscheidet zur Laufzeit und heute verändert morgen.**
Und gleichzeitig: eine bewusst deterministische Hülle drumherum.

Agency sitzt im Kontrollfluss, nicht im Schadensradius. Zwei Lese-Tools = kleiner Wirkungskreis, kein Nicht-Agent.

Wann SDK, wann Microsoft Agent Framework?

SCHICHT	OPTIONEN AUF DIESER SCHICHT	WAS QUEUESCOUT TUT
Tools	Inline-Code oder MCP-Server	<i>zwei C#-Lambdas</i>
Agent-Loop	Einen Agenten-Loop bringen beide mit. · Copilot SDK: komplette Copilot-Runtime, Modell wahlweise übers Abo · Microsoft Agent Framework: eigener Provider, dafür Workflows, Multi-Agent-Übergaben	<i>Copilot-Runtime geliehen</i>
Modell	Abo-Modelle oder eigenes Deployment (Foundry, BYOK)	<i>umschaltbar</i>

Für QueueScout hätte beides funktioniert. Ich habe das SDK genommen, weil ich die fertige Copilot-Runtime wollte und sie über mein Abo laufen lassen konnte. Mit dem Wechsel auf Foundry fiel dieser Vorteil weg. Eindeutig wird die Wahl erst bei mehreren Agenten mit Übergaben.

Q & A *Kein Lagerkampf, sondern eine Schichtenwahl. Die Integration ist auf beiden Seiten dokumentiert: bei GitHub und im Agent Framework.*

Probiert es

github.com/Liebeck/queuescout

MIT-Lizenz · Snapshot Oktober 2026 · NuGet-Version gepinnt



1 git clone

Repo holen

2 .env füllen

Token rein, drei Zeilen

3 docker run

erster Lauf

Unter 10 Minuten bis zum ersten eigenen Lauf.

WAS IHR BRAUCHT

Copilot-Abo und einen fine-grained PAT mit der Berechtigung „Copilot Requests“. **Klassische ghp_-Token funktionieren nicht.** Optional: Serper-Key für die Suche, Foundry-Deployment für das eigene Modell.

BEWUSST NICHT IM REPO

Feedback-Loop, Azure-Infrastruktur und die Wettbewerbs-Kategorie bleiben privat. Die Stolpersteine stehen dafür ehrlich im README.

Drei Dinge zum Mitnehmen

1

Das SDK macht den Loop zur Bibliothek.

Ein Agent ist eine Session, zwei Tools und ein Auftrag. Der Rest ist normale Software. Und es muss kein Coding-Agent sein.

2

Gedächtnis macht aus dem Crawler einen Agenten.

Die Architekturentscheidung ist, wer schreiben darf: Code garantiert, der Agent notiert, der Mensch hat das Veto.

3

Probiert es heute Abend.

Repo klonen, fine-grained PAT, docker run. Stufe 1 ist ein Abend. github.com/Liebeck/queuescout

Danke!

Fragen? Jetzt – oder auf LinkedIn.



LinkedIn

Dr. Matthias Liebeck · liebeck.io



GitHub Copilot Newsletter