We Are Developers!

Frühjahr – 1/2023

Eine Themenbeilage der
Heise Medien GmbH & Co. KG

> **JAVASCRIPT-FRAMEWORK**

Ernsthafte Konkurrenz für Node.js: Bun?

> **PYTHON-TUTORIAL**

Programmieren mit dem Keyword await

> **TWITTER-ALTERNATIVE**

Mastodon: Skepsis aus technischer Sicht

> **KI-CONTENT**

Deep Dive in das Urheberrecht

> **RUST IM LINUX-KERNEL**

Interview mit Projektleiter Miguel Ojeda

> **AGILE PRAXIS**

Entlastung durch interne Developer-Plattformen

Powered by heise Developer

IN AGILEN WORKSTREAMS DIE CLOUD-LÖSUNGEN VON MORGEN ENTWICKELN.

DARUM SIND WIR BEI DATEV.

Gemeinsam sichere Cloud-Lösungen und innovative Apps realisieren: Als Cloud-Entwicklerin oder -Entwickler erwarten dich bei DATEV vielfältige Aufgaben in einer agilen Innovations-Kultur. Informiere dich über freie Stellen und spannende Projekte bei einem der führenden IT-Dienstleister in Europa.

Valeria und Dominik, Cloud-Entwicklerin und -Entwickler bei DATEV

DATEV.DE/KARRIERE

DATEV

Zukunft gestalten. Gemeinsam.

EDITORIAL

You build it, you run it

Nutzt ihr Twitter? Als der Exodus von dort einsetzte, hatte ich's mir auf Mastodon schon gemütlich gemacht mit einem Zweitsofa. Dort geht es friedlicher zu. Man bekommt wenig mit voneinander. Als hätte jemand das Licht stark gedimmt und den Ton halb gemutet. Eskalationen verpuffen geräuschlos. Aber auch inspirierender Austausch hinterlässt kaum Spuren. Auch wenn mir das Konzept dahinter grundsympathisch ist: Die kleine Datscha dort drüben – bleibt eine dezentrale, schlichte Klause. Für die Nebensaison und digitale Auszeiten.

Golo Roden hat ebenfalls **gemischte Gefühle zu Mastodon**, aus anderen Gründen. Darüber sprechen wir in dieser „We Are Developers"-Ausgabe. Der Softwarearchitekt hatte im Herbst 2022 große Pläne: Auf Mastodon eine eigene Instanz aufzubauen, möglichst sicher. Moment, wieso ist da alles in Ruby programmiert? Beginn eines Kampfs gegen Windmühlen: Golo stieß auf veraltete Module, mangelhafte Dokumentation und undurchdachte Strukturen. Wir haben über nachhaltige Finanzierung und Entwicklung gesprochen, über Server-Infrastruktur – und über die Gratismentalität bei Open Source, die schon manch guter Idee den Garaus machte. Zwischenstand: Skepsis.

Die Frühjahrsausgabe ist gesprächiger als sonst: Zwei weitere Interviews erwarten euch hier. Mit Jarred Sumner, dem kalifornischen Gründer des **JavaScript-Frameworks Bun**, haben Timo Zander und ich über dessen Node.js-Alternative gesprochen – und warum der Überflieger die Schule abgebrochen hat. Manche Abbrecher sind erfolgreicher als Mainstreamer und machen jungen Menschen Mut, gegen den Strom zu schwimmen (wie 2005 in Stanford: Steve Jobs).

Mein Kollege Moritz Förster hat Miguel Ojeda über **Rust im Linux-Kernel** befragt. Miguel leitet auf der Rust-Seite dieses Mammut-Projekt, das einen Paradigmenwechsel bei den Kernelsprachen einläutet.

Zur Sache geht es bei Clemens Sielaff, der uns in einem **Python-Tutorial** in 16 Listings das asynchrone Programmieren näherbringt. Das Keyword await ist keine Magie, aber mächtig: Es lässt schwierigen, asynchronen Code wie linearen Code erscheinen. Wie Entwicklerteams sich das Leben erleichtern, da hat auch Robert Ruzitschka einen Tipp: **Interne Developerplattformen** entbinden von operativen Aufgaben. Im Sinne von: „You build it, you run it."

Und ich könnte wetten, auch ihr habt schon generative KI wie ChatGPT oder Midjourney ausprobiert? Gegen Ende des Hefts wird es erneut kämpferisch. Till Jaeger nimmt uns mit auf einen **Deep Dive ins Urheberrecht** – für KI-erstellte Bilder. Schaut selbst, wie Stable Diffusion sich den Kampf der Künstler und der KI um das Urheberrecht vorstellt.

Genug gequasselt: Let's build!

Silke Hahn

INHALT

Young Professionals schreiben für Young Professionals

Unsere Beilage zu c't und iX basiert überwiegend auf einer Online-Artikelserie, die Young Professionals eine Bühne bietet für erste Fachartikel. Die Autoren und Autorinnen erhalten von der Heise-Developer-Redaktion Unterstützung beim Konzipieren und Schreiben.

Dein erster Fachartikel bei Heise

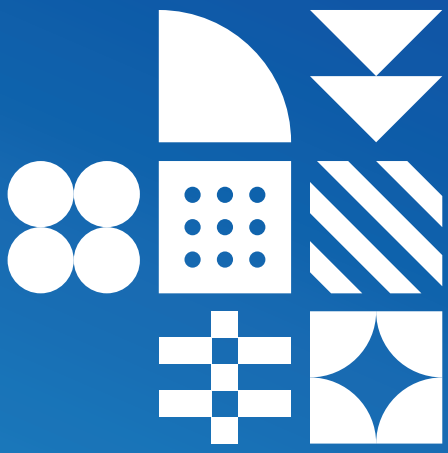
Die Serie ermutigt dazu, eigene Erfahrungen mitzuteilen, ein Projekt vorzustellen – oder wolltest du schon immer mal selbst einen Fachartikel schreiben und hast eine Idee?

Schreib uns: developer@heise.de

> Framework Bun: „Schneller als mir irgendwer glaubt“

Timo Zander

Jarred Sumner ist der Kopf hinter der Node.js-Alternative Bun. Ein Gespräch über seine Motivation, sein Unternehmen und warum er die Schule abgebrochen hat.



Seit 2021 entwickelt der kalifornische Softwareentwickler Jarred Sumner federführend die JavaScript-Runtime Bun. Das Community-getriebene Projekt soll ein All-in-One-Werkzeug sein zum Bundlen, Transpilieren, Installieren und Ausführen von JavaScript- sowie TypeScript-Code.

Bun ist an Deno und Node.js angelehnt. Es ist Open Source und die erste öffentliche Version ist seit Juli 2022 verfügbar. Der Heise-Autor und JavaScript-Entwickler Timo Zander hat die Runtime ausprobiert und mit Jarred Sumner über die Hintergründe gesprochen. Das Interview führte Zander im November 2022, im Original fand das Gespräch auf Englisch statt. Die Originalversion findet sich unter ix.de/ztcp.

Timo: *Bun erlebt derzeit einen Hype. Was ist dein 5-Minuten-Pitch für das Tool?*
Jarred: Bun ist alles in einem: JavaScript-Runtime, Bundler, Transpiler und npm-Paketmanager. Also nimmt es eine ganze Menge an Tools aus dem bestehenden Ökosystem, vereinigt sie in einem Tool und macht es richtig schnell. Der Weg, diese Geschwindigkeit zu erreichen, ist mithilfe von nativem Code und der Wahl schnellerer Systemaufrufe. Leute denken oft, dass Software I/O-gebunden sei, obwohl in Wahrheit die Systemaufrufe das Bottleneck sind. Also denkt man, dass das Lesen von der Festplatte bremst – was I/O-gebunden bedeutet. Doch in Wahrheit braucht das Einlesen und Verarbeiten oft deutlich mehr Zeit. Die Leute unterschätzen das oft, daher konzentriert Bun sich auf Performancegewinn durch das Reduzieren der Dauer von Systemaufrufen.

Timo: *Also geht es hauptsächlich um das Parsen und Auswerten von Code und weniger um Lesen und Schreiben?*

Jarred: Nicht ganz. Bun ist zwanzig- bis hundertmal so schnell wie bestehende npm-Klienten auf Linux. Festplatten sind heutzutage sehr schnell, solange man keine rotierenden Platten benutzt – und das tun die wenigsten. Das Bottleneck ist daher nicht mehr „Zeit zum Schreiben“. Aber wenn man zum Beispiel tausend npm-Pakete installiert, von denen die meisten schon heruntergeladen wurden, benötigt die Masse an Dingen, die dem Computer mitzuteilen sind, die meiste Zeit. Verkleinert man den Overhead dieser Systemaufrufe, erzielt man eine deutliche Beschleunigung.

Bun erreicht das mit einer Mischung aus nativem Code – statt JavaScript – und durch eine genaue Auswahl der Algorithmen. Das ist häufig ganz banal. Ich habe eine Woche damit verbracht, herauszufinden, was der schnellste Weg ist, Dateien zu kopieren. Erst auf Linux, dann auf macOS, denn die Lösungen sind ziemlich unterschiedlich.

Timo: *So viel mal zu macOS and Linux. Was für Pläne gibt es für Windows?*
Jarred: Die Windows-Unterstützung soll binnen eines halben Jahres kommen. Windows Subsystem for Linux 2 lässt sich verwenden. Das funktioniert bereits heute, sofern man den neuesten Kernel hat.

In a Nutshell

- Bun ist ein JavaScript-Tool, das gleichzeitig Runtime, Bundler, Transpiler und Paketmananger vereint. Es ist in der systemnahen Sprache Zig programmiert – dank ihr wirbt Bun insbesondere mit Geschwindigkeit.
- Bun ist ein neuer Konkurrent zu Deno und Node.js, setzt aber auf Kompatibilität mit Bestehendem.
- Jarred Sumner entwickelt das Open-Source-Projekt mit seinem Start-up Oven – im Gespräch mit Heise-Autor Timo Zander gibt er Einblicke in seine Pläne für die Zukunft.

Timo: *Wieso liegt der Fokus so stark auf Performance?*
Jarred: Aus persönlichem Frust. Jahrelang war ich schwer frustriert, wie langsam JavaScript ist. Ich erinnere mich gut, wie ich zum ersten Mal Objective C nutzte und sah, wie viel schneller etwa NSLog gegenüber console.log ist. Mein Gedanke war, dass es keinen Grund gibt, wieso JavaScript nicht so schnell wie nativer Code sein kann bei solchen Aufgaben. Wenn alles von Systemaufrufen gebremst wird, dann geht es ausschließlich um den Overhead zwischen einer Nachricht an die Hardware und ihrer Verarbeitung.

Timo: *Hattest du bereits Erfahrung mit systemnaher Programmierung?*
Jarred: Ehrlich gesagt, nicht wirklich. Wobei, das stimmt nicht ganz. Ich hatte ein Jahr lang ein performancesensibles Voxel-Multiplayer-Spiel mit JavaScript programmiert – Minecraft im Browser. Dort habe ich viel Erfahrung gesammelt, denn es war ziemlich schwierig, das umzusetzen. Irgendwann war der Code dann so groß, dass die Build-Zeiten und die Iterationszeit – also die Zeitspanne, bis eine gespeicherte Änderung sichtbar wird – wirklich extrem schlecht wurden. Ich habe alles probiert: esbuild, meine Tools upgraden, aber es war immer noch nicht schnell. Es war sogar so langsam, dass ich in der Wartezeit angefangen habe, Hacker News zu lesen, und das ist einfach der Tod der Kreativität. Blöd, wenn man den Fokus verliert, nur weil man auf den Build wartet.

Timo: *Was hast du daraufhin versucht?*
Jarred: Zuerst dachte ich: Okay, ich migriere esbuild in die Low-Level-Sprache Zig. Das habe ich getan, habe gebenchmarkt und es war tatsächlich rund dreimal schneller für Dinge wie JSX-Transpilation. Irgendwann habe ich Next.js zum Laufen gebracht. Dann kamen mehr und mehr Dinge, in denen ich mich verlor: Den Transpiler zum Laufen bringen, serverseitiges Rendern. Dafür brauchte ich eine JavaScript-Runtime. Also habe ich mich gefragt: Node startet nicht allzu schnell, wie kann ich das verbessern?

Timo: *Was machtest du dann?*
Jarred: Ich habe begonnen, mit V8-Isolaten zu arbeiten, das war deutlich schneller. Aber ich wollte sehen, wo die Grenze nach oben ist. Daher begann ich, mit JavaScriptCore zu experimentieren – das ist die JavaScript-Engine, die Safari nutzt. In allen Benchmarks, die ich fand, startete JavaScriptCore am schnellsten. Allerdings muss man hier einen Kompromiss eingehen: Hermes von Facebook startet etwas schneller, hat aber keine JIT-Funktion. Da müsste man den Just-in-Time-Compiler separat laufen lassen, was der Laufzeitperformance aber schaden würde.

Interviewgast Jarred Sumner

Jarred Sumner ist ein Softwareentwickler aus der San Francisco Bay Area in Kalifornien. Nachdem er sich das Programmieren selbst beigebracht hatte, baute er zunächst mit Ruby on Rails kleinere Projekte und Apps. Nach seinem letzten Job bei Stripe begann er, die JavaScript-Runtime Bun zu schreiben, und gründete sein eigenes Start-up Oven. Mit seinem Unternehmen treibt er die Entwicklung von Bun voran.

Ich habe mich also für JavaScriptCore entschieden und rund einen Monat gebraucht, um es einzubinden. WebKit ist ein ziemlicher Monolith. Es gibt zwar Dokumentationen, wie die C-API sich nutzen lässt, allerdings sind die eher für einfache Anwendungsfälle gedacht. Etwa für ein Spiel, das JavaScript für einen Teil einbettet, nicht für eine ganze Runtime. Daher habe ich WebKit-Quellcode gelesen. Währenddessen war ich zunehmend frustriert über die langsamen Installationszeiten von npm und habe angefangen, bun install zu schreiben.

Timo: *Wieso hast du Zig als Programmiersprache gewählt?*
Jarred: Zuerst wollte ich Rust verwenden, hatte aber Probleme, produktiv damit zu arbeiten. Sehr cool ist, dass Zig eine so einfache Sprache ist. Man versteht alles, was passiert. Und das ist sehr wichtig für die Performance: In C++ gibt es zum Beispiel Destruktoren, die automatisch aufgerufen werden beim Zerstören eines Objekts. Oder man hat Funktionen, die aufgerufen werden, wenn sie den Scope verlassen. Das sind klassische Quellen von verstecktem Verhalten. Wenn man alles versteht, was in einem Programm vor sich geht, ist es einfach zu optimieren. Und Zig hat außerdem noch ein cooles Feature namens „Comptime", mit dem man beliebigen Quellcode – in normalem Zig – während des Kompilierens ausführen kann.

Andere Sprachen habe Makros oder Templates, meistens stark eingeschränkte Versionen der eigentlichen Sprache. In Zig hingegen ist alles ziemlich simpel, etwa wie es Generics implementiert: Statt einer speziellen Syntax übergibt man einen Typ. Dieser Typ ist dann ein comp-time-Parameter, sodass man einen neuen Typ zurückgeben kann. Ich fand das wirklich cool und war binnen zwei Wochen produktiv im Umgang mit Zig. Die Lernkurve ist wahrscheinlich ähnlich wie bei Go, denn beide haben nicht viel Syntax und sind sparsam mit Abstraktionen. Zig ist nicht schwer zu erlernen.

Timo: *Müssen alle Entwicklerinnen und Entwickler, die du einstellst, Zig schon beherrschen – oder lernen sie es während der Arbeit?*
Jarred: Letzteres! Die Leute lernen einfach Zig, immerhin ist es eine sehr einfache Sprache. Wer Erfahrung mit C oder C++ hat, kann Zig rasch meistern.

Timo: *Was brachte dich dazu, Bun zu beginnen, statt zu bestehenden Projekten wie Deno beizutragen?*
Jarred: Ehrlich gesagt hatte ich Deno nicht in Erwägung gezogen. Ich finde es wichtig, dass der Fokus auf Kompatibilität liegt, damit Leute mit bestehendem Quellcode Bun nutzen können und direkt einen Performancezugewinn haben. Entweder durch Bun build – zum Beispiel durch TypeScript, das out-of-the-box funktioniert – oder durch die Runtime.

Timo: *Versuchte Bun von Anfang an, ein vollumfängliches Tool zu sein?*
Jarred: So hat es nicht angefangen. Es ging zunächst nur darum, die Iterationszeit zu verringern, doch dann ist der Scope kontinuierlich gewachsen und wächst immer noch. Ich habe unlängst erst eine Loader API hinzugefügt. Alles in JavaScript kann deutlich schneller sein. Modernes JavaScript ist einfach so komplex mit seiner Vielzahl an Tools. Wenn man alles in ein Tool packt und es deutlich schneller macht, wird das Leben als Entwicklerin und Entwickler deutlich einfacher.

Timo: *Hattest du in Erwägung gezogen, bestehende schnelle Tools wie pnpm einzubinden, statt Bun install zu schreiben?*
Jarred: Es ist schwierig, Bun install auf einem bestehenden Tool aufzubauen. Damit es schnell ist, muss es in den Rest von Bun integriert sein. Zum Beispiel ist der JSON-Parser ein eigener, der sowohl zum Lesen von package.json in Bun install genutzt wird, aber auch vom Transpiler. Durch das gemeinsame Nutzen des Codes ist es deutlich einfacher, ihn zu optimieren. Außerdem ist im Fall von Bun install nichts in JavaScript geschrieben, sondern alles in Zig.

Timo: *Was ist der Plan für Buns erstes Stable Release – wann soll es erscheinen?*
Jarred: In rund fünf Monaten. Es gibt eine Menge zu tun!

Timo: *Worüber hast du bislang noch nicht gesprochen?*
Jarred: Etwas, worüber ich bislang erst wenig gesprochen habe, ist Bun test. Das ist ein sehr schneller Testrunner für kleine Alltagstests. Er ist Jest-kompatibel, sodass Unternehmen ihn einfach für ihre bestehenden Tests nutzen können. Ich muss klar sagen, dass er noch nicht fertig ist und zurzeit gibt es noch keine Dokumentation dazu, aber das wird kommen. Er ist für kleine Jest-Tests rund vierzigmal schneller und bei einer größeren Menge Code etwa viermal schneller.

Benutzen die Tests allerdings TypeScript, habe ich auch schon Fälle gesehen, in denen Bun zweihundertmal schneller war. Das ist schneller, als irgendjemand je glauben würde – mich eingeschlossen. Ziemlich verrückt. Die Geschwindigkeit kommt erneut überwiegend aus der Architektur. Jest zum Beispiel hat ein eigenes Loader-System für TypeScript. Das muss eine Menge Laufzeit-Prototyp-Änderungen vornehmen. Mit Low-Level-Zugriff auf die Runtime kann man allerlei so ändern, dass es besser funktioniert. Damit kann man einiges an Performance rausholen. Daher denke ich, dass Bun test wirklich cool wird. Doch auch Bun install wird wirklich cool, sobald es Workspaces unterstützt und stabiler ist. Dann kann es von größeren Unternehmen eingesetzt werden. Buns Laufzeit wird aufregend. Eine der spannendsten Sachen, die es geben wird, ist ein Bundler mit clientseitigem Fokus. Also

Entwickle mit uns das Einkaufserlebnis der Zukunft

- 5.000 IT-Begeisterte
- Internationales Umfeld
- State of the Art-Technologien
- Nachhaltige IT
- Mobiles Arbeiten
- zukunftsweisende Projekte

Deine Entwicklung. Unsere Vielfalt.

Cloud Services: Wir entwickeln und vertreiben innovative Cloud-Infrastruktur- und Cloud-Plattform-Services (IaaS und PaaS) in Enterprise-Qualität. Unter der Marke STACKIT bieten wir unsere Cloud Lösungen Made in Germany auch externen Unternehmen an.

Java/Go Development: Du baust zukunftssichere „Betriebssystem-Services“, welche unsere internen Kunden auf unserer internen Cloud STACKIT sowie VMware oder den Public Clouds unterstützen. Zusammen mit den Architekten löst du technische Fragestellungen, z. B. durch Produktevaluationen oder der Durchführung von Proof of Concepts.

Entwicklung: Innovative Anwendungen für Europas größten Händler, die die interne Digitalisierung vorantreiben, z.B. Webshop für Lidl und andere Kunden-Webseiten der Gruppe; Interne Apps (Web-Anwendung für interne Prozesse); Umfangreiche ERP-System Projekte.

C# .NET Development: Du gestaltest die (Neu-)Entwicklung unseres Warenwirtschaftssystems in Richtung einer Cloud-Native Applikationsarchitektur von der ersten Idee bis hin zum internationalen Roll Out. Du implementierst Softwarelösungen mit C# und JS-Frameworks (On Premises und Cloud-Umgebungen).

STACKIT XM Cyber A Schwarz Company

› IT.SCHWARZ

wird es Bun build haben und es wird komplett unabhängig von der Laufzeit nutzbar sein, als Alternative zu esbuild. Und es gibt noch viel mehr, was Leute spannend finden werden: Es gibt zum Beispiel eine vollständige AST-Plugin-API, die direkt in der Laufzeit integriert ist. Das funktioniert ähnlich wie Makros, indem man kleine Funktionen zur Build-Zeit ausführen kann, die dann ASTs (abstrakte Syntaxbäume) anstelle von Strings zurückgeben. Doch in Bun kann man einfach ein Objekt zurückgeben, etwa ein Response-Objekt aus der Web-API. Und wenn dieses Objekt JSON enthält, wird es automatisch in das AST-Format konvertiert.

Man kann auch Kontext an diese Makros übergeben. Zum Beispiel kann man eine HTTP-Anfrage haben, die dynamisch generiertes JavaScript zurückgibt, was sich gut eignet zum Erkennen von totem Code. Bislang waren API-Antworten nie statisch analysierbar, denn sie wurden in einer Art abgerufen, die für Minifier und Transpiler undurchsichtig war. Indem man das in ein Makro verschiebt, kann man deutlich kleinere Builds generieren. Man hat ein besseres Bild davon, welcher Code tatsächlich genutzt wird. Also wird Bun auch einen Minifier haben. Doch das wird noch über sechs Monate brauchen, denn es gibt so viel zu tun.

Timo: *Wie stehst du zu TypeScript? Wird Bun es unterstützen?*
Jarred: Ich persönlich mag TypeScript sehr, aber Buns Integration wird hauptsächlich aus einem Transpiler bestehen. Also wie auch esbuild und swc entfernt Bun alle Typen, bevor der Quellcode ausgeführt wird. Das Ganze funktioniert out-of-the-box, nichts muss extra installiert werden. Zudem liest Bun die tsconfig.json aus: Wenn ein bestehendes Projekt also etwa Paths nutzt, dann funktioniert das einfach in Bun. Kein zusätzliches Einrichten nötig. Bun prüft allerdings auch keine Typen, das ist hier der Kompromiss. Ich denke aber, dass es langfristig realistisch ist, den TypeScript-Compiler direkt einzubinden für das Szenario. Derzeit ist das nicht geplant, wir müssen den Scope realistisch begrenzen.

Timo: *Bleibt noch etwas außen vor zurzeit?*
Jarred: Eine interessante Alternative, die Bun nicht verfolgt, ist, eine Light-Version von TypeScript zu erstellen. Sobald das TypeScript Type Annotation Proposal implementiert ist, wäre es deutlich einfacher, so eine schnelle Version zu implementieren. Aber das macht ein ganz anderes Fass auf, denn dann muss man ein eigenes LSP bauen und es entsteht plötzlich eine Inkompatibilität mit bestehenden Tools. Es wird sehr schnell sehr kompliziert. Also ja, das werde ich aktuell nicht machen.

Timo: *Du sprichst in der Ich-Form, hast aber mit Oven auch ein Start-up gegründet, das hinter Buns Entwicklung steht. Welche Teamgröße strebst du an?*
Jarred: Wir stellen ein! Die Teamgröße wird zwischen drei und fünf Entwicklerinnen und Entwicklern liegen in den nächsten sechs Monaten, die meisten sollen dann Zig-Entwickler sein.

Timo: *Jarred, in einem Tweet hattest du gesagt, dass Oven „ein Grind" sein werde, also eine Plackerei. Auf Twitter sorgte das für ziemliche Empörung. Was ist deine Sicht hierzu?*
Jarred: Das hatte ich nicht erwartet. Einige Leute sind heute noch sauer auf mich. Wenn ich das noch mal kommunizieren müsste, dann würde ich sagen, dass Oven ein Start-up in einem frühen Stadium ist. Und es wird eine Menge harter Arbeit brauchen. Es gibt viel zu tun und ich denke, dass Leute, die Interesse an Mitarbeit haben, das vorher wissen sollten. Ich finde es wichtig, vorab ehrlich zu sein. Das Wort „Grind" zu nutzen war aber ein Fehler. Zudem gehen Menschen auf Twitter immer vom Schlechtesten aus und nehmen böse Absichten an. Ich hätte sagen sollen, dass die ersten Mitarbeiter auch eine große Menge an Unternehmensanteilen bekommen werden. Als junges Start-up ist das ein wichtiger Teil der Vergütung. Entwicklerinnen und Entwickler bekommen einen extrem bedeutsamen Anteil am Unternehmen zusätzlich zu dem Gehalt – wir haben sieben Millionen US-Dollar gesichert, um Beschäftigten gutes Gehalt zu zahlen. Das hätte ich sagen sollen. Denn die Leute haben wohl angenommen, dass man bei uns ohne Bezahlung hart arbeiten soll. Das wäre Unsinn.

Timo: *Warum hast du das Start-up Oven gegründet statt Bun als reines Open-Source-Projekt aufzuziehen?*
Jarred: Es war von Anfang an mein Plan, ein Unternehmen aufzubauen und Start-ups zu gründen.

Timo: *Hat deine Zeit bei Stripe deine Entscheidung zur Gründung Ovens beeinflusst?*
Jarred: Früher habe ich Apps für Endkunden gemacht. Das war vor Stripe. Zum Beispiel erstellte ich eine Pokémon-Go-Karte, die 4,9 Millionen Leute genutzt haben. Oder eine Chrome-Erweiterung, mit der man anderen Zugriff zu einer Webseite geben konnte, ohne das eigene Passwort zu teilen – auch das haben ein paar Millionen Leute genutzt. Also habe ich viele verschiedene Sachen gemacht, die aber noch keine Start-ups waren. Und ich habe eine Menge dabei gelernt. Stripe hatte allerdings keinen Einfluss auf meinen Wunsch, ein eigenes Start-up zu gründen. Mit 14 habe ich eine Menge Hacker News gelesen, das war der größte Einfluss.

Timo: *In deiner Twitter-Biografie steht, dass du die High School abgebrochen hast. Was ist denn die Geschichte dahinter?*
Jarred: In der Bay Area aufzuwachsen war mein Glück. Als Kind habe ich meine gesamte Zeit am Computer verbracht.

Building a better working world

Smart entwickeln – in Beruf und Business

Vielfalt für innovative Lösungen

Als Manager im Bereich Data & Analytics bei EY treibt Alexandros Sparos die digitale Transformation voran. Wie er das macht und was ihm daran gefällt, erzählt er hier.

Alexandros, was machst du als Intelligent Automation Manager bei EY?
Zusammen mit meinem Team helfe ich Unternehmen aus unterschiedlichsten Branchen, ihre Prozesse durch Automatisierungen zu verbessern. Hierfür nutzen wir eine Vielzahl von Technologien wie Robotic Process Automation (RPA), künstliche Intelligenz oder Machine Learning.

Ein Beispiel ist die Implementierung von KI-basierten Lösungen in einem Produktionsunternehmen, um Maschinenausfälle vorherzusagen und Wartungsarbeiten zu optimieren. Ein weiteres Projekt, an dem ich beteiligt war, beinhaltete die Prozessautomatisierung durch RPA in einem Finanzunternehmen. So konnten wir die Bearbeitungszeit erheblich reduzieren und die Datenqualität verbessern.

Wie laufen deine Projekte typischerweise ab?
Wir arbeiten eng mit den Unternehmen zusammen, um die Geschäftsprozesse zu verstehen. Nach einer detaillierten Analyse diskutieren wir im Team verschiedene Lösungsansätze und entwickeln ein geeignetes Konzept. Im nächsten Schritt führen wir eine Machbarkeitsstudie durch und evaluieren den Geschäftsnutzen der Automatisierungslösung. Anschließend folgt die Implementierung.

Wie seid ihr im Projektteam aufgestellt?
Das Team ist in der Regel interdisziplinär zusammengesetzt und besteht aus Fachkräften für verschiedenste Bereiche wie Business-Analyse, Softwareentwicklung, Automatisierungstechnologie und Projektmanagement. Dass sich EY aktiv für Vielfalt, Gleichberechtigung und Inklusion einsetzt, zeigt sich auch in der Zusammensetzung unseres Teams und der gesamten Organisation. Wir glauben, dass die Zusammenarbeit mit Menschen unterschiedlicher Hintergründe und Erfahrungen nicht nur die Leistungsfähigkeit des Unternehmens steigert, sondern auch dazu beiträgt, eine Kultur der Innovation und des Wandels zu schaffen.

Auf welche Skills kommt es in deinem Job bei EY besonders an?
Ein tiefes technisches Verständnis ist unerlässlich. Nur so kann ich die technischen Anforderungen der Unternehmen verstehen und die beste Automatisierungslösung für ihre Bedürfnisse empfehlen. Da jedes Projekt einzigartig ist, ist es außerdem wichtig, schnell und kreativ auf unvorhergesehene Situationen reagieren zu können und alternative Lösungen zu finden. Dabei spielt gutes Teamplay natürlich auch eine große Rolle.

Warum ist EY für Personen mit IT-/Tech-Hintergrund ein spannender Arbeitgeber?
EY ist ein global tätiges Unternehmen und bietet die Möglichkeit, in verschiedene Branchen und Märkte einzutauchen. Wir arbeiten eng mit führenden Technologieunternehmen zusammen und sind Teil eines umfangreichen Expertise-Netzwerks in der Tech-Industrie. Dazu kommen vielfältige Angebote zur Weiterentwicklung. Insgesamt arbeiten wir hier also in einem dynamischen Umfeld, das uns Zugang zu den neuesten Technologien und Trends ermöglicht.

Das schafft eine Vielzahl an Chancen, die eigenen Fähigkeiten zu erweitern und sich beruflich weiterzuentwickeln. Wer Lust darauf hat, täglich Neues zu lernen, ist hier genau richtig und kann sich auf eine offene und inklusive Unternehmenskultur freuen, in der die persönliche Entfaltung leichtfällt.

Interesse geweckt? Hier findest du spannende Karrieremöglichkeiten mit Fokus auf neue Technologien in Bereichen wie Analytics, Cyber Security, SAP-Beratung und Transformation: **www.ey.com/de_de/careers/technology**

Irgendwann habe ich angefangen, Ruby on Rails zu lernen – das ist nun gut zehn Jahre her – und habe damit eine Anwendung gebaut und sie meinen Lehrern gezeigt. Und ich dachte, wenn ich diese Anwendung ganz alleine bauen kann, dann kann ich vielleicht auch einen Job finden. Denn ich hasste die Schule und war auch nicht gut darin. Also habe ich eine Menge „Cold E-Mails" an Unternehmen in der Nähe verschickt, eines davon hat mich als Praktikant eingestellt. Meine Sicht auf die Dinge war, dass ich entweder sechs Jahre in der Schule verbringe, um einen Job zu finden – oder mir direkt eine Stelle suche. Für mich hat das gut funktioniert.

Timo: *Wie hat dein Umfeld darauf reagiert?*
Jarred: Meinen Eltern hat es nicht gefallen, dass ich die High School abgebrochen habe. Sie waren besorgt. Noch vor zwei oder drei Jahren haben sie mich regelmäßig gefragt, wann ich aufs College gehe. Allerdings war das für mich nie ein Problem durch das Silicon Valley und die Startups hier. In anderen Sektoren wäre das deutlich schwerer. Aber ich denke, eine Sache, die ich wirklich gut gemacht habe – die andere Leute sich abschauen sollten – sind meine Cold E-Mails. Viele Leute, besonders Entwicklerinnen und Entwickler, unterschätzen den Einfluss einer guten Initiativbewerbung. So habe ich nicht nur meinen ersten Job gefunden, sondern auch später noch weitere. Daher würde ich sagen, dass es für mich gut geklappt hat, die High School abzubrechen. Zum einen, weil ich wirklich Glück hatte, aber auch weil ich hart gearbeitet habe. Ich habe dadurch viel Zeit mit Programmieren verbracht.

Timo: *Kannst du ein paar Tipps für erfolgreiche Initiativbewerbungen geben?*
Jarred: Am besten einfach darüber sprechen, wieso man Interesse an dem Unternehmen hat, in normaler Alltagssprache. Auch finde ich es gut, dem CEO direkt zu mailen, die meisten Leute machen das nicht. Man kann auch dem Hiring Manager schreiben, je nach Unternehmen. Und dann spricht man über seine bisherigen Projekte oder relevante Erfahrungen mit möglichst vielen Details. Vielleicht kann man ja sogar ein Projekt vorzeigen, das eine besondere Schnittmenge mit dem Unternehmen zeigt. Ein Freund von mir hat zum Beispiel für ein Praktikum in einem T-Shirt-Start-up deren T-Shirt-Designer nachgebaut als Beleg, dass er Code schreiben kann und interessiert ist. Den hat er dem Unternehmen gezeigt und sie damit beeindruckt. Für Oven erwarte ich allerdings nicht, dass jemand so etwas tut.

Timo: I*n einer Stellenausschreibung hattest du Bewerber gebeten, Projekte zu zeigen, die sie mit Bun gebaut hatten…*

Jarred: Das war keine so gute Idee. Es lagen gerade einmal zwei Stunden zwischen meinem Tweet über Ovens Fundraising und der Banküberweisung auf mein Konto. Über all diese Tweets hatte ich nicht genug nachgedacht im Eifer des Gefechts. Grundsätzlich denke ich, dass es interessanter ist, relevante Projekte zu haben, die Erfahrung mit Low-Level-Softwaresystemen zeigen. Oven sucht aktuell keine Fullstack-Entwickler. Und das finde ich komisch, denn die Leute, die sich am meisten auf Bun freuen und es nutzen werden, sind nicht diejenigen, die es bauen werden. Es ist schlicht ein anderes Skillset aufgrund der Low-Level-Systeme. Jegliche Erfahrung mit C, Rust, C++ oder Zig ist von Vorteil. Besonders im Kontext von JavaScript-Interpretern oder interpretierten Sprachen im Allgemeinen.

Timo: *Was genau wird Ovens Beitrag zu Bun sein?*
Jarred: Bun ist eine Runtime und bindet daher die Engine nur ein. Trotzdem denke ich, dass wir an der Engine mitarbeiten werden. Es ist spannend, all die Performanceverbesserungen in JavaScriptCore zu sehen. Eine Verbesserung, über die ich getwittert habe, war, dass Proxies vier- bis fünfmal schneller geworden sind beim Aufruf von Gettern. Eine andere, über die ich nicht getwittert habe, ist String.replace, das nun doppelt so schnell ist. Oven wird dabei mithelfen, denn für uns ist es wichtig, dass die Runtime-API schnell ist, genauso wie die tatsächlichen Implementierungen von verschiedenen JavaScript-Features.

Timo: *Wieso sollten Leute sich auf Bun freuen?*
Jarred: Alles in JavaScript kann deutlich schneller und viel einfacher werden. Entwicklerinnen und Entwickler beschweren sich oft über die Komplexität in JavaScript. Und ein Hauptgrund hierfür ist, dass es so viele verschiedene Tools gibt, die alle eine einzelne Sache tun. Das macht es schwer zu verstehen, wie alles zusammenpasst. Auch sind diese Tools häufig sehr langsam. Bun packt alles in ein Tool, macht es deutlich schneller und ist zudem kompatibel zum restlichen Ökosystem.

Timo: *Danke für deine Zeit, Jarred!*

Das Interview führte Timo Zander, externer Autor für Heise in der Reihe Young Professionals. Zander hat Angewandte Mathematik und Informatik studiert. Als Entwickler interessiert er sich für Open Source und das JavaScript-Universum. (sih)

Quellen

Die Links zur englischen Version, zur Website von Sumners Start-up Oven sowie zu seinem Twitterkonto finden sich unter ix.de/ztcp.

This ad will not influence the future of retail technology with every line of code. You will.

Bei ALDI SÜD gestalten über 2.400 IT-Kolleg:innen Tag für Tag die IT-Zukunft des Einzelhandels mit. In unseren Teams der Nationalen und Internationalen IT hast du die einmalige Gelegenheit, innovative Technologien für den weltweiten Einsatz voranzutreiben – in einem stabilen und zugleich lebendigen Arbeitsumfeld.

Die Facts.
ALDI SÜD als Arbeitgeber:

Moderner Tech-Stack: SAP, Adobe, Salesforce, M365 u.v.m.

Facettenreiches Team: Über 2.400 diverse Kolleg:innen in der IT

Global Player: IT-Projekte in 11 Ländern auf 4 Kontinenten

Future Work: Arbeit bis zu 100% remote möglich

Super Aussichten.
Und haufenweise Extras:

- Mobiles Arbeiten innerhalb Deutschlands inkl. Equipment
- Internationale Projekte
- Attraktive Vergütung
- Zukunftsorientiertes Training & Development
- Gesundheitsangebote

Deine Chance – bewirb dich jetzt!

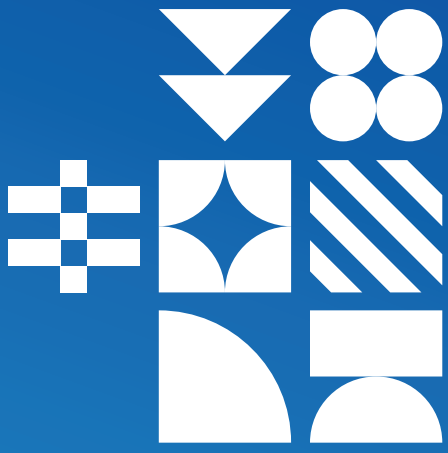
it-jobs.aldi-sued.de

Unabhängig von den Texten und Bildern unserer Recruiting-Materialien möchten wir ausdrücklich betonen, dass bei ALDI SÜD alle Menschen gleichermaßen willkommen sind.

> Asynchrones Programmieren: async minus await

Clemens Sielaff

Viele Programmiersprachen bieten das Keyword await. Dieses Tutorial zeigt anhand eines Python-Beispiels, was bei seinem Einsatz geschieht.



Das Keyword await lässt sich dazu nutzen, asynchrone Funktionen zu schreiben. Dieser Artikel soll dabei helfen, eine Intuition dafür zu entwickeln, was man dem Computer durch `await` mitteilt. Als Beispiel dient das Schreiben einer kleinen Async-Library in Python, ohne dabei das `await`-Keyword zu verwenden. Anschließend folgt ein Vergleich mit Code, der `await` mit dem Python-Standardmodul asyncio nutzt.

Von synchron zu asynchron

Das Beispiel ist denkbar einfach: Alice und Bob möchten beide einen drei Sekunden langen Countdown herunterzählen. Gleichzeitig, und zwar ohne, dass der eine vom anderen weiß. Listing 1 und 2 zeigen eine synchrone Implementation.

In a Nutshell

- Seit den 2010er-Jahren existiert das `await`-Keyword in diversen Programmiersprachen.
- Das Schreiben einer Async-Library in Python soll ein Verständnis dafür vermitteln, was durch den Einsatz des Keywords geschieht.
- Unter Verwendung von `await` kann schwieriger, asynchroner Code wie linearer Code erscheinen.

Alice zählt ihre drei Sekunden, danach Bob seine. Was man stattdessen möchte, ist ein gleichzeitiger Fortschritt in beiden Countdowns. Aber da keine Threads zum Einsatz kommen, können die beiden Funktionen nicht parallel ablaufen. Die erste Einsicht beim asynchronen Programmieren ist, dass Warten auch Fortschritt bedeuten kann. Tatsächlich können beliebig viele Funktionen gleichzeitig warten – und das auf einem einzigen Thread.

Sollen sich Alice und Bob mit dem Zählen abwechseln, können sie problemlos gemeinsam warten. Dazu ist ein Scheduler nötig, der in Listing 3 zu sehen ist. Dessen Output zeigt Listing 4. Zuerst ein paar Worte zur Implementierung des Schedulers (Listing 3):

[1] Um den Kontrollfluss zuverlässig zu steuern, darf es nur einen Scheduler geben. Er ist deshalb als Singleton geschrieben.

[2] Die `while`-Schleife wird von `countdown` in den Scheduler verlagert. Dieser kann somit mehrere gleichzeitige Schleifen verzahnen.

[3] Der Scheduler kann nur Funktionen ohne Parameter ausführen, `countdown` benötigt allerdings einen Namen und die nächste Zahl. Deshalb wird der eigentliche Aufruf in eine parameterlose Lambda-Funktion verpackt. Wer mehr wissen möchte, kann nach Closures oder Pythons `functools.partial` suchen (alle Links unter ix.de/ztv7).

Und tatsächlich, Alice und Bob zählen nun wie gefordert abwechselnd. Zusammen brauchen die beiden Countdowns

Software Craft Community:

Dezentrales Lernen in der Softwareentwicklung

Es gibt vermutlich wenige berufliche Felder, in denen die ständige Auseinandersetzung mit neuen Trends und Technologien so intensiv ist wie die Softwareentwicklung. Ohne kontinuierliches Lernen geht hier gar nichts. Wer dabei an klassische Seminare und Fortbildungen denkt, liegt nur in Teilen richtig: Neben diesen typischen, eher behäbigeren Formen des Lernens hat sich längst eine vollkommen neue Lernkultur etabliert: das selbstgesteuerte Lernen ganz nah an den Anforderungen des Arbeitsalltags. Hier offenbaren sich die Grenzen zentralisierter Weiterbildungsangebote, die den schnellen Lernzyklen und den aus der Praxis geborenen spezifischen Anforderungen nicht gerecht werden können. Bei DATEV haben sich daher sogenannte Communities of Practice etabliert, die sich unter anderem für vielfältige, dezentral organisierte Lernangebote engagieren. Die Learning-Abteilung sorgt im Hintergrund für einen hohen Professionalisierungsgrad.

Voneinander Lernen

Die Communities of Practice bieten ein ideales Umfeld für dezentrales Lernen. Die Grundidee der Communities basiert darauf, dass sich Menschen aus verschiedenen Bereichen und Hierarchieebenen aufgrund eines gemeinsamen inhaltlichen Interesses vernetzen, sich gegenseitig unterstützen, in einen gemeinsamen Austausch treten und voneinander lernen. Dieses Konzept lebt von eigenmotivierten Mitgliedern, die ihre Ideen und Kenntnisse einbringen.

Das Interesse an einem gemeinsamen Thema bringt Mitarbeitende über Hierarchie- und Bereichsgrenzen hinweg zusammen: So ist die Software Craft Community zum Beispiel eine der größten und ältesten Communities of Practice bei DATEV. In ihr vernetzen sich Software-Entwickler:innen untereinander, um das Verständnis von Code-Qualität weiterzuentwickeln und die Coding-Skills ständig zu verbessern. Der Grundgedanke folgt dem Ethos des Handwerks: Ständige Übung macht den Meister – und alle sind ein Leben lang sowohl Lernende wie Lehrende. Ein festes Organisationsteam koordiniert die Community-Aktivitäten, Selbstinitiative ist jedoch gewünscht und gefordert. Jedes Mitglied kann Themen anstoßen. Dezentral organisierte Lernangebote gehören zu den Kernaktivitäten der Software Craft Community. Im Jahr 2022 veranstaltete die Community weit mehr als 50 Lern-Events, darunter Vorträge und Veranstaltungen, aber auch Kleingruppenformate wie Coding Dojos.

Bei einem Coding Dojo finden sich Entwickler:innen zusammen, um gemeinsam ein Code Kata – eine vorgegebene Programmieraufgabe von überschaubarer Dimension – durchzuführen und voneinander zu lernen. Die restlichen Teilnehmenden verfolgen das Geschehen. In einem festgelegten Rhythmus rotiert ein/e Entwickler:in aus dem Programmier-Paar heraus und jemand anderes nimmt den Platz ein. Die Aufgabe wird dabei ständig wiederholt, sodass sich die Lösung kontinuierlich weiterentwickelt.

Freiwillig und eigenmotiviert

DATEV-Mitarbeitende bekommen in der Software Craft Community die Möglichkeit, selbstbestimmt von überall und jederzeit zu lernen, ihre Kenntnisse und Fähigkeiten im Bereich der Softwareentwicklung auf dem neusten Stand zu halten und sich auf die sich ständig ändernden Anforderungen einzustellen. In der Software Craft Community stehen deshalb das gegenseitige Lehren und Lernen im Vordergrund. Dabei ist alles, was in der Community passiert, auf freiwilliger Basis: Wer mitmacht, der tut es aus Überzeugung. Es wird also permanent und selbstorganisiert gelernt.

www.datev.de/karriere

Listing 1: Ein einfacher, synchroner Countdown

```
from time import sleep

def countdown(name: str, n: int):
    while n >= 0:
        print(name, n)
        sleep(1)
        n -= 1

countdown("Alice", 3)
countdown("Bob", 3)
```

Listing 2: Output von Listing 1, zwei Countdowns nacheinander

```
Alice 3
Alice 2
Alice 1
Alice 0
Bob 3
Bob 2
Bob 1
Bob 0
```

allerdings immer noch sechs statt drei Sekunden. Das Problem findet sich im Aufruf von `sleep`. Jedes Mal, wenn Bob oder Alice warten, blockieren sie nicht nur sich selbst, sondern auch den jeweils anderen. Schlimmer noch, auch den gesamten Scheduler. Es reicht also nicht nur, dass `countdown` keine eigene Schleife mehr haben darf – die Funktion selbst darf ebenfalls nicht mehr schlafen. Auch das übernimmt der Scheduler (Listing 5). Nun zählen Bob und Alice gleichzeitig herunter.

Besseres Async mit Generatoren

Während diese Art des asynchronen Programmierens funktioniert, ist sie deutlich weniger intuitiv als traditionelle, synchrone Programmierung:

- Anstatt einer lokalen while-Schleife muss sich die Funktion selbst aufrufen.
- Um die Argumente einzufangen, ist es nötig, den Aufruf in eine Closure zu integrieren …
- … und diese dann explizit an einen globalen Scheduler zu übergeben.

Aber was wäre, wenn es auch einfacher ginge? Noch soll nicht `await` ins Spiel kommen, sondern ein anderes Keyword, das Python anbietet (und weshalb dieses Tutorial nicht beispielsweise in JavaScript geschrieben ist): `yield`. Im Vergleich zum Beispiel in Listing 1 lässt sich damit der in Listing 6 gezeigte Code schreiben.

Um zu verstehen, was genau hier vor sich geht, lohnt sich ein Blick auf das Konstrukt hinter `yield`.

Was ist ein Generator?

Das `yield`-Keyword gibt einen Wert an die aufrufende Funktion zurück, wie ein `return`. Anders als `return` lässt sich jedoch eine Funktion nach einem `yield` weiter ausführen, und zwar beginnend mit dem nächsten Statement. Alle Variablen innerhalb der Funktion behalten dabei ihre Werte bei.

Der technische Ausdruck für eine Funktion in Python, die ein oder mehrere `yield`-Statements enthält, ist ein Generator. In der Fachliteratur sind Generatoren klassifiziert als ein Subtyp von Koroutinen, die wiederum die theoretische Grundlage von asynchroner Programmierung bilden. Anders als asynchrone Funktionen, die nach außen sichtbar mit dem Keyword `async` zu kennzeichnen sind, sehen Generatoren wie gewöhnliche Funktionen aus. In der Praxis fällt der Unterschied aber spätestens beim Aufruf des Generators auf. Denn anders als eine Funktion gibt dieser nicht etwa einen Wert zurück, sondern ein Generatorobjekt (vgl. Listing 7):

```
gen = foo()  # <generator object foo at 0x7f...>
```

Generatoren kommen, wie der Name vermuten lässt, meist zur Generierung von Werten – etwa für eine Iteration – zum

Listing 3: Countdown mit einem einfachen Scheduler

```
from time import sleep
from typing import *  # type hints - not necessary but informative

class Scheduler:  # [1]
    ready: List[Callable] = list()  # functions ready to execute

    @classmethod
    def call_soon(cls, func: Callable):
        cls.ready.append(func)

    @classmethod
    def run(cls):
        while cls.ready:
            current = cls.ready.pop(0)
            current()

def countdown(name: str, n: int):
    if n >= 0:  # no longer a loop
        print(name, n)
        sleep(1)
        Scheduler.call_soon(lambda: countdown(name, n - 1))  # [2]

Scheduler.call_soon(lambda: countdown("Alice ", 3))  # [3]
Scheduler.call_soon(lambda: countdown("Bob ", 3))
Scheduler.run()
```

Listing 4: Output von Listing 3, zwei Countdowns zur selben Zeit

```
Alice  3
Bob  3
Alice  2
Bob  2
Alice  1
Bob  1
Alice  0
Bob  0
```

Listing 5: Wait Smarter, Not Harder

```
from time import sleep, time as now
from typing import *

class Scheduler:
    ready: List[Callable] = list()
    sleeping: List[Tuple[float, Callable]] = list()  # sleeping functions

    @classmethod
    def call_soon(cls, func: Callable):
        cls.ready.append(func)

    @classmethod
    def call_later(cls, delay: float, func: Callable):
        start_time = now() + delay
        cls.sleeping.append((start_time, func))
        cls.sleeping.sort()  # sort by start_time

    @classmethod
    def run(cls):
        while cls.ready or cls.sleeping:
            if not cls.ready:
                start_time, func = cls.sleeping.pop(0)  # get next
                delta = start_time - now()
                if delta > 0:
                    sleep(delta)  # sleep until next wakes up
                cls.call_soon(func)
            while cls.ready:
                current = cls.ready.pop(0)
                current()

def countdown(name: str, n: int):
    if n >= 0:
        print(name, n)
            # no sleep
        Scheduler.call_later(1, lambda: countdown(name, n - 1))

Scheduler.call_soon(lambda: countdown("Alice ", 3))
Scheduler.call_soon(lambda: countdown("Bob ", 3))
Scheduler.run()
```

Einsatz. Um den Generator den nächsten Wert erzeugen zu lassen, verwendet Python die „magische“ `Generator.__next__()`-Methode, die für Anwender besser aufrufbar ist als `next()`:

```
print(next(gen))  # “one: 1”
print(next(gen))  # “two: 2”
print(next(gen))  # produces a StopIteration exception
```

All das übernimmt Python beim Aufruf eines Generators in einer `for x in`-Schleife automatisch. `x` bekommt dann so lange den zuletzt generierten Wert zugewiesen, bis der Generator eine `StopIteration` erzeugt, was die Schleife beendet.

Wie beim `await`-Keyword kann man dieselbe Funktion auch selbst und ohne Python-Magie implementieren. Beim Scheduler ist nur ein Anpassen der `run`-Methode erforderlich. Wurde zuvor eine Funktion aufgerufen, ist nun ein durch `yield` ausgegebener Wert beziehungsweise eine `StopIteration` zu handhaben (Listing 8).

Das führt zu deutlich weniger Boilerplate-Code bei gleichem Output. Das Streichen der Lambda-Funktionen, die bis-

Listing 6: Countdown als Generator-Funktion

```
[...]
def countdown(name: str, n: int):
    while n >= 0:
        print(name, n)
        yield 1  # sleep for 1 second
        n -= 1
[...]
```

Listing 7: Eine triviale Generator-Funktion als Beispiel

```
def foo():
    i = 1
    yield f"one: {i}"
    i += 1
    yield f"two: {i}"
```

Listing 8: Scheduler mit Generatoren

```
[...]
class Scheduler:
    [...]
    @classmethod
    def run(cls):
        while cls.ready or cls.sleeping:
            [...]
            while cls.ready:
                current = cls.ready.pop(0)
                try:
                    delay = next(current)
                except StopIteration:
                    continue  # discard finished generators
                if delay is not None:
                    cls.call_later(delay, current)
[...]

Scheduler.call_soon(countdown("Alice ", 3))
Scheduler.call_soon(countdown("Bob ", 3))
Scheduler.run()
```

Listing 9: Ein Task besteht aus einem Generator und einem Resultat

```
[...]
class Task:
    def __init__(self, generator: Generator):
        self.generator: Generator = generator
        self.result: Optional[Any] = None
[...]
```

Listing 10: Der Scheduler kann schlafen, einen Wert erwarten oder einen produzieren

```
[...]
class Sleep:
    def __init__(self, duration: float):
        self.duration: float = duration

class Result:
    def __init__(self, result: Any):
        self.result: Any = result

class Await:
    def __init__(self, task: Task):
        self.task: Task = task
[...]
```

lang für das Versehen des ersten Aufrufs von `countdown` mit Argumenten nötig waren, ist nun ebenfalls möglich.

Tasks und Abhängigkeiten

Bislang hat der Scheduler zwei Möglichkeiten zur Ausführung eines Generators: so bald wie möglich mit `call_soon` oder nach einer bestimmten Wartezeit mit `call_later`. Zusammen mag das für einen einfachen Countdown reichen, für eine plausible Async-Library genügt das aber nicht. Statt nur zu warten, dass die Zeit vergeht, erwarten Funktionen zumeist das Eintreten eines bestimmten Ereignisses. Das kann etwa das Laden einer Datei aus dem Internet oder der nächste Tastenanschlag des Nutzers sein.

Um diesen wichtigen Teil der asynchronen Programmierung zu behandeln, wird das Beispiel durch das Aufteilen von `countdown` in zwei Funktionen nun ein wenig komplizierter. Die eine, der Konsument, wird vom User aufgerufen, die andere spielt einen asynchron laufenden Produzenten, der aufwendig die nächste Zahl „berechnet“. Was in der synchronen Welt trivial anmutet, erzeugt hier jedoch sofort ein Problem: Wie soll der Produzent das generierte Resultat an den Konsumenten übergeben? Er kann kein `return` verwenden, ohne synchron zu werden, und jeder mit `yield` übergebene Wert weist den Scheduler an zu warten. Die Lösung besteht aus drei Teilen:

1. Zuerst muss der Produzent befähigt werden, ein Resultat an den Scheduler zu überreichen.
2. Dann muss der Scheduler Abhängigkeiten zwischen Generatoren verstehen, damit er den Konsumenten aufwecken kann, sobald der Produzent seine Arbeit abgeschlossen hat.
3. Und zuletzt braucht der Konsument eine Möglichkeit, das Resultat auszulesen.

Diese Schritte werden im Folgenden von hinten anfangend erklärt. Der Konsument stößt den Produzenten an und wartet dann auf das Ergebnis. Wie auch immer er es anstellt, er muss den Aufruf des Produzenten an den Scheduler übergeben. Was also, wenn der Scheduler dem Konsumenten ein `Task`-Objekt zurückgibt, das ein leeres Feld enthält, in dem später das Ergebnis des Generators geschrieben wird? Auf diese Weise kann der Konsument das Resultat auslesen, sobald der Produzent es bereitstellt, wie Listing 9 zeigt.

Statt mit Generatoren arbeitet der Scheduler nun mit Tasks. Zum Verwalten von Abhängigkeiten zwischen Tasks wird dem Scheduler ein einfaches Mapping hinzugefügt. Das Gruppieren blockierter Tasks erfolgt darin nach Abhängigkeit. Sie können weder „ready“ noch „sleeping“ sein, sondern bleiben blockiert, bis ihre Abhängigkeit erfüllt ist.

Listing 11: Künstlich erschwerter Countdown mit einem asynchronen Produzenten

```
[...]
def producer(n: int):
    yield Sleep(1)
    yield Result(n - 1)

def countdown(name: str, x: int):
    while x >= 0:
        print(name, x)
        task = Task(producer(x))
        yield Await(task)
        x = task.result

Scheduler.call_soon(Task(countdown("Alice ", 3)))
Scheduler.call_soon(Task(countdown("Bob ", 3)))
[...]
```

Nur bleibt die Frage, wie ein Resultat vom Produzenten in seinen assoziierten Task und letztendlich zum Konsumenten gelangt. Ein Produzent weiß nicht, welcher Task ihm zugeordnet ist. Das weiß nur der Scheduler. Daher muss der Produzent – als Generator – mit dem Scheduler auf dem einzigen Weg kommunizieren, den er hat: `yield`.

Statt eines nackten Werts übergibt der Generator ein Befehlsobjekt, das den Scheduler anweist, wie er den Task zu behandeln hat. Für das Beispiel sind drei Befehle nötig, die

- einen Generator für gewisse Zeit schlafen legen,
- ein Resultat nehmen und es in den assoziierten Task schreiben,
- einen Generator so lange schlafen legen, bis ein anderer abgeschlossen ist.

Die Umsetzung dieser Befehle wird dem Scheduler überlassen. Jeder Befehl wird durch eine eigene Klasse abgebildet,

Listing 12: Ausführung von Befehlen im Scheduler

```
[...]
class Scheduler:
    blocked: Dict[Task, Set[Task]] = dict()
    [...]
    @classmethod
    def run(cls):
        [...]
        while cls.ready:
            current = cls.ready.pop(0)
            try:
                instruction = next(current.generator)
            except StopIteration:
                continue

            if isinstance(instruction, Sleep):
                cls.call_later(instruction.duration, current)

            elif isinstance(instruction, Result):
                # copy the result into its task
                current.result = instruction.result
                # reschedule newly unblocked tasks
                if current in cls.blocked:
                    for blocked in cls.blocked[current]:
                        cls.call_soon(blocked)
                    del cls.blocked[current]

            elif isinstance(instruction, Await):
                # block the current task
                if instruction.task in cls.blocked:
                    cls.blocked[instruction.task].add(current)
                else:
                    cls.blocked[instruction.task] = {current}
                # schedule the blocking task
                cls.call_soon(instruction.task)
[...]
```

die dem Scheduler ermöglicht, zu erkennen, was er zu tun hat. Listing 10 zeigt die drei Befehlsklassen.

Mit diesen Änderungen lässt sich der Produzent aus der `countdown`-Funktion lösen. Gleichzeitig ist der Code nun deutlich expliziter und damit lesbarer (Listing 11). Abschließend ist es nötig, die `Scheduler.run`-Funktion ein letztes Mal zu erweitern, um mit den neuen `yield`-Objekten umzugehen (Listing 12). Damit ist der Scheduler abgeschlossen.

Async Queue

Als letztes wichtiges Feature jeder Async-Library wird nun eine asynchrone Queue implementiert. Asynchrone Queues sind interessant, weil sie dabei helfen, langlebige Produzenten zu modellieren, die laufend neue Daten erzeugen. Das können beispielsweise Aktienkurse oder die Mausbewegungen eines Users sein. Der kurzlebige Produzent aus Listing 11 musste stattdessen jedes Mal aufs Neue von einem Konsumenten gestartet werden und gab nur einen Wert zurück, bevor er stoppte.

Queues als Konzept sind denkbar einfach. Ein Produzent und ein Konsument bekommen dieselbe Queue – der Produzent fügt Werte mit `put` hinzu, der Konsument erwartet Werte mit `get`. Die Rate der Produktion neuer Werte ist dabei vollkommen unabhängig von der Rate ihres Auslesens. Echte Queues werden beliebig kompliziert, aber dieses Beispiel hält die Implementation so einfach wie möglich. Zur Verdeutlichung der Idee zeigt Listing 13 den endgültigen Code für den Countdown von Produzent und Konsument.

Listing 13: Asynchrone Queues

```
[...]
def producer(n: int, queue: AsyncQueue):
    while n >= 0:
        queue.put(n)
        yield Sleep(1)
        n -= 1

def consumer(name: str, queue: AsyncQueue):
    while True:
        task = queue.get()
        yield Await(task)
        print(name, task.result)
        if task.result == 0:
            return

alices_queue = AsyncQueue()
bobs_queue = AsyncQueue()

Scheduler.call_soon(Task(producer(3, alices_queue)))
Scheduler.call_soon(Task(producer(3, bobs_queue)))

Scheduler.call_soon(Task(consumer("Alice", alices_queue)))
Scheduler.call_soon(Task(consumer("Bob", bobs_queue)))
[...]
```

Listing 14: Einfügen eines Items in eine asynchrone Queue

```
[...]
class AsyncQueue:
    def __init__(self):
        self.items: List[Any] = list()
        self.waiting: List[Task] = list()

    def put(self, item: Any):
        self.items.append(item)
        if self.waiting:
            Scheduler.call_soon(self.waiting.pop(0))
[...]
```

Anders als zuvor besitzt der Produzent nun die Hauptschleife des Countdowns. Er generiert die Werte und schiebt sie in die Queue. Dabei nimmt er keine Rücksicht darauf, wer sie auf der anderen Seite empfängt. Der Konsument erhält derweil so lange neue Werte, bis die Null erreicht ist, woraufhin er endet.

Die asynchrone Queue an sich besitzt eine Liste aus Werten und eine Liste von wartenden Tasks, falls in einem Zeitraum mehr `gets` als `puts` aufgerufen wurden. Tasks in der Warteliste werden von der Queue verwaltet. Sie sind zwar effektiv blockiert, aber nicht von einem anderen Task im Scheduler, sondern von der Queue selbst. So lange sie in der Queue auf den nächsten Wert warten, sind sie dem Scheduler nicht einmal bekannt.

Die Queue in diesem Beispiel hat nur zwei Methoden: `get` und `put`. Bei jedem `put` schaut die Queue zuerst, ob es einen wartenden Task gibt. Wenn ja, entnimmt sie ihn aus der Liste und übergibt ihn an den Scheduler (Listing 14).

Die `get`-Methode (Listing 15) bietet eine Herausforderung: Es kann nicht zu jeder Zeit garantiert sein, dass ein Wert in der Queue vorhanden ist. Wenn die Queue leer ist, muss der aufrufende Task warten, bis ein neuer Wert hinzugefügt wurde. Um das zu ermöglichen, kreiert `get` einen neuen Task on-the-fly, der den aufrufenden Task blockieren kann.

Dieses Design hat zur Folge, dass jedes `get` den aktuellen Task unterbricht, um den „getter“ auszuführen, selbst wenn schon ein Wert in der Queue bereit liegt. Viele echte Implementierungen bieten deshalb sowohl synchrone als auch asynchrone Funktionen zum Auslesen von Werten an.

Vergleich mit echtem asyncio

Und damit ist die Async-Library fertig. Natürlich ist sie nicht produktionstauglich. Aber sie sollte ausreichende Funktio-

Listing 15: Auslesen eines Items aus einer asynchronen Queue

```
[...]
class AsyncQueue:
    [...]
    def get(self) -> Task:
        # awaitable generator that produces the next item eventually
        def receive():
            while not self.items:
                # add yourself as a waiting task if necessary
                self.waiting.append(task)
                # an empty yield removes this task from the Scheduler
                yield
            # once an item has become available, return it
            yield Result(self.items.pop(0))

        # name the task instance, so it becomes part of the closure
        task = Task(receive())
        return task
[...]
```

Listing 16: Asynchroner Countdown mit Pythons asyncio-Modul

```
import asyncio

async def producer(n: int, queue: asyncio.Queue):
    while n >= 0:
        queue.put_nowait(n)
        await asyncio.sleep(1)
        n -= 1

async def consumer(name: str, queue: asyncio.Queue):
    while True:
        result = await queue.get()
        print(name, result)
        if result == 0:
            return

async def main():
    bobs_queue = asyncio.Queue()
    alices_queue = asyncio.Queue()
    await asyncio.gather(
        asyncio.create_task(producer(3, bobs_queue)),
        asyncio.create_task(consumer("Alice", alices_queue)),
        asyncio.create_task(consumer("Bob", bobs_queue)),
        asyncio.create_task(producer(3, alices_queue)),
    )

asyncio.run(main())
```

nen bieten, um zumindest einfachsten Anwendungen zu genügen – und das alles in weniger als 100 Zeilen Code. In Listing 16 ist noch einmal der gleiche Countdown zu sehen, doch diesmal mit Pythons integriertem asyncio-Modul. Abgesehen davon, dass die Library Tasks unsichtbar handhabt, ist der Code fast identisch. Der einzige Unterschied ist die `run`-Funktion des Schedulers. Während die zuvor erstellte Library läuft, bis alle Tasks abgearbeitet sind, würde die von asyncio sofort enden, wenn die Tasks nicht explizit mit dem Aufruf von `asyncio.gather` erwartet würden.

await ist keine Magie

Es ist faszinierend zu sehen, wie das `await`-Keyword schwierigen, asynchronen Code wie linearen erscheinen lässt. Und das, ohne dabei die Komplexität preiszugeben, die sich hinter dem Begriff verbirgt. Was letztendlich genau passiert, ist von Sprache zu Sprache unterschiedlich, aber die Abstraktion ist universell.

Letzten Endes werden die wenigsten Entwicklerinnen und Entwickler ihre eigene Async-Library schreiben müssen. Das Verwenden einer solchen ist allerdings nahezu unumgänglich. Daher ist es hilfreich, eine Vorstellung davon zu haben, was man damit meint, wenn man `await` schreibt. (mai)

Quellen

Weitere Informationen und das GitHub-Repository des Autors mit den gezeigten Codebeispielen finden sich unter ix.de/ztv7.

Clemens Sielaff

arbeitet seit zwölf Jahren als Softwareentwickler für Visual Effects und Games, unter anderem bei Weta Digital, Ziva Dynamics und Uncharted Territory. Zur Zeit lebt er im heimischen Norddeutschland und entwickelt Tools für Unity.

> Interview: „Große Bedenken, auf Mastodon zu setzen“

Silke Hahn

Taugt Mastodon als Twitter-Alternative? Softwarearchitekt Golo Roden hält aus technischer Sicht Skepsis für angebracht.

Seit Elon Musk im Herbst 2022 Twitter übernommen hat, sind ehemals passionierte Privatnutzer, aber auch Behörden und Medienvertreter in Scharen zu anderen Diensten abgewandert – oder haben ein zweites Standbein anderswo in den Sozialen Medien aufgebaut. Das dezentrale Fediverse diente vielen Twitterflüchtlingen als Auffanglager, insbesondere der Nischendienst Mastodon ist seither stark gewachsen. Wir haben uns gefragt, wie es von technischer Seite aussieht: Was taugt die Twitter-Alternative? Hätte Mastodon gar das Zeug, Teil der kritischen Infrastruktur zu werden, wie manche behaupten?

Der Softwarearchitekt Golo Roden war neugierig und wollte für seine Firma eine eigene Mastodon-Instanz aufsetzen. Dabei legte er Wert auf Sicherheitsaspekte – und machte überraschende, teils auch eher abschreckende Entdeckungen. Seinen Unmut tat er auf Twitter kund und stieß dabei eine kontroverse Diskussion an. Silke Hahn von *heise Developer* hat mit ihm über seine Erfahrungen gesprochen.

In a Nutshell

- Mastodon ist eine dezentrale Alternative zu Twitter, deren langfristiger und nachhaltiger Erfolg aber noch abzuwarten ist.
- Aus technischer Sicht ist bei Mastodon aufgrund veralteter Module und fehlender Verschlüsselung Skepsis angebracht.
- Gänzlich offen ist die Frage nach einer nachhaltigen Finanzierung der Entwicklung von Mastodon, vor allem aber auch der erforderlichen (Server-)Infrastruktur.

Silke: *Wie stehst du zurzeit zu Twitter?*
Golo: Spricht man über Twitter, muss man zwischen der technischen Implementierung der Plattform und der inhaltlichen Nutzung unterscheiden.

Aus rein technischer Sicht mache ich mir tatsächlich verhältnismäßig wenig Gedanken um Twitter. Nach der Übernahme durch Elon Musk und den Massenentlassungen wurde vielfach geunkt, dass Twitter bald nicht mehr funktionsfähig sei. Allerdings gab es bislang keinen einzigen nennenswerten Ausfall, und im Großen und Ganzen läuft das System nicht weniger stabil oder langsamer als vorher. Zumindest ich persönlich habe bislang ehrlich gesagt überhaupt keinen Unterschied bemerkt. Ob die eingeschlagene Richtung mit kostenpflichtigen blauen und gelben Häkchen richtig ist, wird die Zeit zeigen. Tendenziell finde ich es aber positiv, dass es endlich ein Bezahlmodell gibt.

Silke: *Lass uns dazu nachher unbedingt noch im Detail sprechen. Zurück zu Twitter, hast du auch Kritikpunkte?*
Golo: Im Vergleich zum Technologischen sehe ich Twitter inhaltlich sehr kritisch. Ich finde es in höchstem Maße bedenklich, dass zahlreiche Teams aufgelöst wurden, die nicht direkt einen technischen Beitrag leisten, sondern dafür Sorge

getragen haben, dass ein vernünftiger Umgangston durchgesetzt wird. Zwar war Twitter in der Hinsicht auch vor Elon Musk kein Paradebeispiel, aber zum Beispiel Teams für Menschenrechte und Diversität einfach von heute auf morgen zu streichen, finde ich sehr schwierig. Und das ist etwas, was neben der Stabilität der Plattform unter Umständen der viel wichtigere Faktor ist: Fühlen sich die Nutzerinnen und Nutzer wohl, herrscht ein positiver, konstruktiver und vor allem respektvoller Umgangston vor, und so weiter.

Silke: *Das verstehe ich gut. Du bist auf Twitter ja nicht nur privat unterwegs, sondern auch als Unternehmer, hatte ich gesehen. Nutzt du es noch aktiv?*
Golo: Ich persönlich und auch wir als Unternehmen haben lange überlegt, wie wir weiter verfahren wollen. Meinen privaten Account nutze ich seit Anfang des Jahres nicht mehr. Mit dem Unternehmensaccount sind wir derzeit mangels Alternative noch vertreten. Wie lange das allerdings noch der Fall sein wird, bleibt abzuwarten. Inzwischen ist es um Twitter und Elon Musk wieder etwas ruhiger geworden. Ich denke, wie gesagt, dass die Zeit zeigen wird, ob es sich lohnt, noch auf Twitter vertreten zu sein oder nicht.

Silke: *Es gibt ja auch Alternativen. Wie geht es dir als IT-Profi mit Mastodon?*
Golo: Grundsätzlich finde ich die Idee für ein dezentrales soziales Netzwerk, das auf offenen Standards basiert, eine schöne Sache. Von Mastodon bin ich aber bislang nicht besonders angetan. Das hat zum einen technische Gründe, zum anderen herrscht im Allgemeinen eine aus meiner Sicht verkehrte Erwartungshaltung vor: Des Öfteren ist zu hören, dass genau das, was ich gerade bei Twitter kritisiert habe, bei Mastodon so viel besser sei, nämlich der Umgangston. Dass man dort, anders als bei Twitter, so viele „vernünftige“ Menschen antreffe.

Silke: *Kann ich bestätigen, ich habe mir im November neben Twitter zum Ausprobieren ein Mastodon-Konto eingerichtet, und Heise hat nun eine eigene Instanz. Trolle scheinen Mastodon noch zu meiden.*
Golo: Das mag so sein, das ist aber kein Verdienst von Mastodon. Denn Mastodon ist, genauso wie Twitter, zunächst einmal nur eine technische Plattform, und was daraus gemacht wird, liegt zum größten Teil in der Hand der Anwenderinnen

Interviewgast Golo Roden

Golo Roden ist Gründer und CTO von the native web GmbH. Er beschäftigt sich mit der Konzeption und Entwicklung von Web- und Cloud-Anwendungen sowie -APIs, mit einem Schwerpunkt auf Event-getriebenen und Service-basierten verteilten Architekturen. Sein Leitsatz lautet, dass Softwareentwicklung kein Selbstzweck ist, sondern immer einer zugrundeliegenden Fachlichkeit folgen muss.

und Anwender. Wenn es also bei Mastodon so ist, dass sich dort derzeit vornehmlich eine gewisse Zielgruppe trifft, dann liegt das primär daran, dass das die ersten waren, die von Twitter zu Mastodon gewechselt sind, weil sie sich auf Twitter nicht mehr wohlgefühlt haben. In Anbetracht der Tatsache, welche Teams bei Twitter gestrichen wurden, liegt auf der Hand, dass das primär Menschen sind, denen Themen wie beispielsweise Inklusion und Diversität wichtig sind. Es ist aber naiv zu glauben, dass das an Mastodon läge – denn je mehr Menschen auf diese Plattform wechseln, desto mehr werden sich auch andere Weltanschauungen auf Mastodon finden.

Silke: *Mir fällt auf, dass man auch als kleiner Account leicht ins Gespräch kommt und rasch interessante Kontakte findet. Die Chats laufen etwas sachlicher ab, es wird ernsthafter diskutiert, kommt mir vor. Es ist sicher hilfreich, dass ein Toot doppelt so viele Zeichen umfassen kann wie ein Tweet. Ist die Kultur vielleicht doch eine andere, aufgeschlossenere?*
Golo: Facebook bietet noch mehr Zeichen pro Beitrag, und ich würde jetzt nicht unbedingt behaupten, dass dort noch weitaus ernsthafter diskutiert wird – eher im Gegenteil. Für ausführliche sachliche Diskussionen war Twitter der falsche Ort, das ist richtig, aber eine fundierte Meinung darlegen, erklären und mit Fakten zu unterlegen, dafür sind auch 500 Zeichen nicht ausreichend. Stell Dir mal vor, wir müssten dieses Interview per Mastodon führen!

Silke: *Oh je … stimmt!*
Golo: Unabhängig davon ist das dann auch immer noch eine Frage dessen, was die Betreiberin oder der Betreiber der jeweiligen Mastodon-Instanz zulässt, aber das ist schlussendlich eine Frage der Kultur und der Atmosphäre, die durch die Anwenderinnen und Anwender entsteht, und keine technische Frage. Insofern ist Mastodon keine schlechte Sache – nur wird der Aspekt mit der Positivität aus meiner Sicht deutlich überbewertet. Es würde mich nicht überraschen, wenn Mastodon mittelfristig mit genau den gleichen inhaltlichen Problemen zu kämpfen hat wie Twitter, denn spätestens, wenn es Server gibt, auf denen gewisse Meinungen toleriert werden, verlagern sich auch die zugehörigen Themen zu Mastodon.

Silke: *Was ist aus deiner Sicht besser: ein zentraler Dienst, der von einem Unternehmen gesteuert und überwacht wird, das aber verantwortungsbewusst damit umgeht – oder eine Handvoll Betreiberinnen und Betreiber, die das in ihrer Freizeit machen?*
Golo: Ehrlich gesagt, weder noch. Dass ein zentral gesteuerter und kontrollierter Dienst schwierig sein kann, haben wir mit Twitter nun gerade erlebt. Positiv ist aber, dass ein Unternehmen den Dienst betreibt, das gewisse Ressourcen aufwenden kann. Ob Twitter das nun profitabel gemacht hat oder nicht, sei mal dahingestellt. Der Punkt ist aber, dass eine derartige Plattform viel Aufwand nach sich zieht, in der Wartung, im Betrieb und so weiter. Und da weiß ich nicht, ob ich das in den Händen einiger weniger Freiwilliger sehen möchte, die das unentgeltlich machen. Die Absichten sind sicherlich hehrer Natur, das will ich gar nicht in Abrede stellen – aber es gibt halt einen großen Unterschied zwischen einem professionellen, sicheren und verlässlichen Betrieb und dem, was eine Privatperson leisten kann oder will. Um das zu betonen: Ich möchte niemandem die Kompetenz oder den Willen absprechen, aber es ist einfach ein Unterschied, ob sich zig Leute 24/7 ausschließlich um den Betrieb kümmern, oder ob das jemand nebenher in ihrer oder seiner Freizeit machen muss. Spätestens, wenn auf einer Mastodon-Instanz sehr viele Nutzerinnen und Nutzer registriert sind, dann wird es kritisch. Ich will nicht den Teufel an die Wand malen, aber ich bin da skeptisch.

Silke: *Wieso will man eine eigene Instanz aufsetzen bei Mastodon? Was sind Vorteile, mögliche Nachteile?*
Golo: Es ist erstaunlich, wie wenig Mastodon-Instanzen es bislang gibt. Natürlich entstehen ständig neue, aber in Anbetracht dessen, wie viele Menschen in den vergangenen Monaten von Twitter zu Mastodon gewechselt sind, sind es überraschend wenige. Eigentlich müsste man bei einem dezentralen System ja eher erwarten, dass mehr oder weniger alle ihre eigene Instanz betreiben, denn ansonsten ist es zwar technisch ein dezentrales System, aber wenn sich die meisten Menschen am Ende doch wieder auf einer Handvoll Instanzen wiederfinden, ist das nicht das, was man unter dezentral vermutlich erwarten würde – und es schadet genau der Verlässlichkeit, über die wir gerade gesprochen haben.

Insofern finde ich es sehr wichtig – wenn man Mastodon nutzen möchte – dass man, sofern man technisch dazu in der Lage ist, etwas zum Fediverse beiträgt und eine eigene Instanz aufsetzt. Denn das macht das ganze System resilienter und unabhängiger, und vermeidet außerdem das Problem, dass irgendjemand in ihrer oder seiner Freizeit sich mit der Wartung und Pflege von Servern befassen muss, von denen auf einmal doch wieder tausende oder zehntausende von Menschen abhängen.

Silke: *Klingt plausibel. Wie gehst du das an, machst du das als Unternehmen oder privat?*
Golo: Für uns als Unternehmen lag es nahe, dass wir eine eigene Instanz aufsetzen würden, wenn wir zu Mastodon wech-

seln wollen. Ob man die komplett selbst betreibt oder sich einkauft, steht noch einmal auf einem anderen Blatt, aber einfach nur einen Account bei einer komplett fremden Instanz abzuschließen, käme für uns nicht infrage. Außerdem hat man mit einer eigenen Instanz mehr Kontrolle über zum Beispiel die langfristige Stabilität von Accountnamen – denn niemand garantiert, dass eine Instanz, auf der man lediglich Mitglied ist, dauerhaft erhalten bleibt, und wenn dem nicht so ist, sind zwar nicht unbedingt die eigenen Daten weg, aber zumindest der Accountname, da die Domain eventuell nicht weiter existiert.

Silke: *Und das Ziel dabei, was wäre der Mehrwert?*
Golo: Unterm Strich geht's um Stabilität und Verlässlichkeit, und um die Kontrolle über die eigenen Daten, Accounts und Co. Da spielen dann Themen mit hinein wie zum Beispiel: Wo werden denn die Daten überhaupt gespeichert, also in welchem Land? Das ist für die DSGVO relevant, und die wiederum ist spätestens bei privaten Direktnachrichten – die es in dieser Form auf Mastodon ja gar nicht gibt – wichtig zu beachten.

Silke: *Was ist dir auf technischer Seite aufgefallen?*
Golo: Leider war unser erster Versuch, eine Mastodon-Instanz testweise aufzusetzen, nicht sonderlich erfolgreich, sondern ziemlich ernüchternd. Mastodon weist nämlich eine Reihe von technischen Aspekten auf, die ich persönlich als schwierig ansehe. Dabei geht der erste Eindruck in eine andere Richtung, denn die Dokumentation für Anwenderinnen und Anwender ist schön gemacht, gut verständlich, anschaulich, und so weiter. Leider endet dieser Eindruck, wenn man damit beginnen möchte, eine eigene Instanz zu betreiben – und zwar so, dass sie tauglich für den Produktivbetrieb ist, das heißt hochverfügbar, redundant, mit einem Loadbalancer versehen, über TLS abgesichert und containerisiert.

Container kommen in der Dokumentation mit keinem einzigen Wort vor, die Installation wird ausschließlich für Bare-Metal beschrieben. Docker und Kubernetes scheinen nicht zu existieren. Eine Bare-Metal-Installation ist aus meiner Sicht aber weder zeitgemäß noch erstrebenswert, außerdem erfüllt sie die Anforderungen an (elastische) Skalierbarkeit nur bedingt. Immerhin enthält das Repository von Mastodon ein Dockerfile und ein Docker-Compose-Skript, aber auch hier: keine Dokumentation, auch nicht auf dem offiziellen Docker Hub.

Silke: *Ich bin überrascht, das klingt nicht gut. Gibt es denn keine Anleitung zur Installation?*
Golo: Doch, das schon, aber eben nur zur Installation von Hand, Containerisierung wird wie gesagt nicht erwähnt, so als ob die Dokumentation aus den 90er-Jahren sei. Darin wird dann erwähnt, dass man Ruby on Rails installieren müsse. Das liegt daran, dass Mastodon auf Basis dieser Technologie entwickelt wird. Und das finde ich wiederum für ein Produkt, was es erst seit 2016 gibt, eine schwierige Entscheidung. Ruby on Rails hatte seine Hochzeit um 2007 herum, plus/minus zwei Jahre, aber es ist seit Jahren auf dem absteigenden Ast, was die Verbreitung angeht. Dass es bestehende Projekte auf dieser Basis gibt, ist selbstverständlich – aber ich hätte mir bei einem noch so verhältnismäßig jungen Projekt dann doch eine etwas andere Technologie gewünscht. Es geht dabei nicht darum, das Neueste zu verwenden, sondern etwas zu nutzen, bei dem absehen-

bar ist, dass es nicht beständig Marktanteile und Relevanz verliert – und das war bei Ruby bereits 2016 absehbar, dass das so kommen würde.

Silke: *Stimmt, zu Ruby hattest du dich auf Twitter geäußert und ich hatte das auf Mastodon geteilt. Da schlugen die Wogen der Empörung hoch, es gab auch starke Befürworter dieser Technologie. Was war dein Take-away aus dieser hitzigen Diskussion damals?*
Golo: Auffallend war, welche Gegenargumente genannt wurden. Da hieß es zum Beispiel, dass Ruby aber doch sehr verbreitet sei. Das habe ich allerdings auch gar nicht bestritten, ich habe nur gesagt, dass die Verbreitung stetig abnimmt, weil es für neue Projekte zunehmend weniger eingesetzt wird. Dann wurde gefragt, ob es mir lieber wäre, wenn man auf das Neueste vom Neuen setzen würde. Auch das habe ich so nicht gesagt – mir geht es nicht darum, auf Biegen und Brechen etwas Neues zu nutzen, sondern ich hätte gerne eine Basis, bei der zumindest nicht absehbar ist, dass ihre Verbreitung immer weiter sinken wird. Natürlich kann niemand die Zukunft voraussagen, und gerade Aussagen über die Zukunft von Technologien sind schwierig. Aber trotzdem gibt es Tendenzen, wenn man den Markt über Jahre beobachtet. Und da zeigt Ruby eben eher eine Tendenz nach unten.

Das Ganze endete dann darin, dass mir jemand einen Link zu einer Webseite geschickt hat, wo man in einem Diagramm ganz klar sehen konnte, dass Ruby auf einem der vorderen Plätze der Verbreitung war, was er als Beleg dafür angeführt hat, dass meine These nicht stimmen würde. Hätte er allerdings den Text unter dem Diagramm gelesen, hätte er dort wortwörtlich lesen können, dass Ruby seit Jahren auf dem absteigenden Ast ist. Ich habe diese Diskussion dann nicht weitergeführt, denn das führt zu nichts. Interessant, wenn auch nicht überraschend fand ich, wie wenig faktenbasiert diskutiert wird und wie emotional vieles dann direkt wird, dass sich Menschen persönlich angegriffen fühlen, wenn man eine faktengestützte Aussage über ihre Lieblingssprache trifft … nun ja.

Silke: *Zurück zu deinem Testlauf mit der eigenen Instanz. Wie ging es da weiter?*
Golo: Die nächste Enttäuschung war die Anbindung der Datenbanken. Mastodon braucht PostgreSQL und Redis, optional lässt sich noch ElasticSearch anbinden. Abgesehen davon, dass das Aufsetzen der Konfiguration dafür in Mastodon sehr umständlich, fehleranfällig und vor allem schlecht dokumentiert ist, haben wir es nach einiger Arbeit doch geschafft, Mastodon auf Kubernetes zu starten. Leider sind wir nicht weit gekommen, denn wir sind an der Verbindung zu Redis gescheitert: Denn unsere Redis-Installation setzt eine TLS-Verbindung voraus, was Mastodon aber nicht unterstützt, weil das zugehörige Ruby-Modul über vier Jahre alte ist und TLS schlichtweg nicht kennt!

Silke: *Wow, aus Sicherheitsperspektive klingt das eher übel.*
Golo: Das empfinde ich schon als fahrlässig. Es geht nicht darum, dass Mastodon perfekt sein müsste, das wäre zu viel erwartet – aber dass derart grundlegende Sachen wie eine TLS-Verbindung zur Datenbank nicht möglich sind, weil der Client dafür hoffnungslos veraltet ist, das vermittelt leider kein gutes Gefühl, dass es im Code nicht noch mehr solche Probleme gibt, und dass man sich hier eventuell ein größeres Sicherheitsproblem ins Haus holt, denn anscheinend stehen Themen wie Verschlüsselung oder Pflege von Dependencies nicht besonders weit oben auf der Tagesordnung. Insofern war das sehr enttäuschend.

Silke: *Klingt nach einer ziemlichen Durststrecke. Wo stehst du gerade beim Aufbau dieser eigenen Instanz und wie gehst du nun vor?*
Golo: Wie gerade geschildert, habe ich vor allem aus technischer Sicht große Bedenken, auf Mastodon zu setzen. Natürlich lässt sich alles irgendwie lösen, so kann man beispielsweise das Problem mit der TLS-Verbindung mit stunnel umgehen, und theoretisch könnte man den Code auch selbst patchen, aber die Frage ist natürlich: Wie viel Aufwand zieht das nach sich, und welche Probleme gibt es noch alle, von denen man erst einmal gar nichts weiß? Will man das alles wirklich, und lohnt sich das langfristig?

Silke: *Bleibst du dran oder ist die Idee jetzt verworfen?*
Golo: Wir haben die Evaluierung von Mastodon vorerst gestoppt und (für uns) für gescheitert erklärt. Uns ist das zu heikel, und ehrlich gesagt konterkarieren genau solche technischen Unzulänglichkeiten das, was ich vorhin gesagt habe, nämlich, dass es im Sinne eines dezentralen Systems sein müsste, dass es möglichst viele kleine und unabhängige Instanzen gibt. Wenn man das erreichen möchte, dann muss die Einstiegshürde möglichst niedrig liegen, und davon ist Mastodon leider sehr weit entfernt.

Silke: *Was bräuchte es, damit das besser klappt?*
Golo: Damit das im großen Stil funktioniert bräuchte man ein Mastodon, das aus einem Binary besteht, das man herunterladen und starten kann, bei dem man lediglich ein paar Umgebungsvariablen oder CLI-Flags setzen muss, und das war's. Und natürlich müsste das Ganze auch vernünftig dokumentiert sein. Ich weiß, dass ich da gerade „wünsch Dir was" spiele und eine solche Software nicht einfach vom Himmel fallen wird. Aber die Frage war ja, was es bräuchte.

Wenn es eine solche Software gäbe und die dann kostenpflichtig wäre, wäre ich auch gerne bereit, dafür zu bezahlen. Das mag eine unpopuläre Meinung sein, aber mir ist Open Source nicht sympathischer, nur weil es kostenlos ist. Gute Software darf gerne Geld kosten. Immerhin steckt da auch eine Menge Zeit und Arbeit drin. Und sobald es kostenpflichtig wird, steigen die Chancen dafür, dass die Software langfristig und vor allem auch für die Entwicklerinnen und Entwickler nachhaltig entwickelt werden kann, dramatisch. Das ist bei Open Source ja auch nicht anders, nur fällt die Finanzierung dort nicht so sehr auf, weil es beispielsweise über Spenden von großen Unternehmen im Hintergrund läuft. Aber dieser Ansatz funktioniert eben nicht für alle Open-Source-Projekte.

Silke: *Was ist denn der Kern des Problems?*
Golo: Unterm Strich merkt man an der Stelle, dass Mastodon eigentlich noch gar nicht für eine dermaßen hohe Verbreitung geeignet war und es im Prinzip über Nacht von einem Hype überrollt wurde. Das tut mir für den federführenden Entwickler hinter Mastodon, Eugen Rochko, sehr leid, denn eigentlich trägt er etwas dazu bei, die Welt zu einem besseren Ort zu machen, nur ist ein Projekt in dieser Größenordnung schwierig mehr oder weniger allein zu stemmen. Und deshalb ist mir auch wichtig zu betonen, dass ich ihm aus alldem keinen Vorwurf machen möchte – es sind zunächst einmal technische Feststellungen.

Silke: *Inzwischen sind ja auch Bundesbehörden, Hochschulen, Presse und Medien zu Mastodon umgezogen oder haben dort einen Zweitwohnsitz eingerichtet. In Anbetracht deiner sehr grundsätzlichen Kritik interessiert mich eins brennend: Hat Mastodon trotzdem das Potenzial, Teil der kritischen Infrastruktur zu werden – oder ist es das schon?*
Golo: Im Moment sehe ich weder das eine noch das andere. Ja, Mastodon wird verwendet, aber im Vergleich zu Twitter ist es immer noch eine sehr kleine Nische. Wie lange die einzelnen Instanzen sich halten werden, bleibt abzuwarten. Immerhin muss jede Instanz betrieben und gepflegt werden, was Zeit kostet, das heißt, irgendwer muss sich darum kümmern. Das kann man mit einem Bezahlmodell verknüpfen (und das machen ja auch einige Instanzen), aber ob das tragfähig ist, ob das ausreicht, ob das langfristig über Jahre funktioniert, das werden wir noch sehen.

Silke: *Nun hast du ja Mastodon mit Blick auf die Sicherheit unter die Haube geschaut. Wo siehst du das größte Risiko?*
Golo: In Anbetracht meiner eben ausgeführten technischen Zweifel bin ich gespannt, wann es den ersten größeren Hack von Mastodon gibt. Spätestens, wenn dann personenbezogene Daten in Umlauf kommen, vermeintlich private Direktnachrichten, und so weiter, wenn also die DSGVO ins Spiel kommt, dann kann das für die einzelne Betreiberin oder den einzelnen Betreiber schnell teuer werden, und da ist dann vermutlich die wahrscheinlichste Option, dass eine Instanz dann offline geht, weil man sich diesen Ärger nur einmal einhandeln möchte. Ich glaube beziehungsweise ich befürchte, dass es zu viele Instanzen gibt, wo sich über derartiges gar keine Gedanken gemacht wird, und das funktioniert dann unter Umständen nur für eine gewisse Zeit, bis zum erstem großen Knall. Solange es dafür kein Konzept gibt, wie die Plattform beziehungsweise wie die ein-

zelnen Instanzen damit umgehen, habe ich meine Zweifel, ob Mastodon wirklich eine derartige Relevanz wie Twitter entwickeln kann.

Silke: *Bis hierhin wirkt das auf mich ziemlich ernüchternd. Wie könnte denn ein funktionierendes Fediverse und Mastodon ausschauen?*
Golo: Die Antwort auf diese Frage ist einfach, aber nicht schön: Mastodon, genau wie Open Source und das ganze Internet an sich, krankt an einem ganz eklatanten Punkt, den ich eben schon einmal kurz angerissen hatte: Es muss für die breite Masse immer alles kostenlos sein. Das sieht man sehr schön bei Open Source, denn das überzeugendste Argument für Open Source ist in der Regel, dass es nichts kostet. Da geht es nicht um Werte, da geht es nicht um Ideale, da geht es einfach nur darum, dass man kein Geld ausgeben muss.

Bei Mastodon sind die ersten Anwenderinnen und Anwender anscheinend im größeren Stil bereit, dafür ein paar Euro im Monat zu zahlen, aber das skaliert meiner Meinung nach nicht. Die meisten Menschen nehmen, wenn sie es kostenlos bekommen können, die kostenlose Variante, und nehmen dann dafür Werbung oder – viel schlimmer – die „zweckentfremdete Nutzung" ihrer Daten in Kauf, gemäß dem Motto, wenn ein Produkt nichts kostet, ist man selbst das Produkt.

Silke: *Wäre ein Bezahlmodell die Lösung?*
Golo: Ich hatte eben schon einmal angesprochen, dass die Betreiberinnen und Betreiber einer Mastodon-Instanz sich das auch leisten können müssen, dass es mindestens Zeit kostet. Tatsächlich kostet es aber auch jede Menge Geld, wenn man es richtig machen möchte. Immerhin fallen Server, Strom, Backup & Co. nicht vom Himmel, sondern wollen bezahlt werden. Und solange eine allgemeine „Geiz ist geil"-Mentalität vorherrscht, funktioniert das nicht im großen Stil, denn dann tragen einige wenige die Kosten für ein größeres System. Das ist zutiefst unfair, das führt zu Burnout, das führt dazu, dass Menschen irgendwann keine Kraft mehr haben, sich zu engagieren. Solche Fälle gab und gibt es in der Open-Source-Szene immer wieder, und das wird an dieser Stelle nicht anders laufen. Es ist en vogue, für Software nichts bezahlen zu müssen – aber es schadet der gesamten Branche.

Und solange wir als IT-Branche ein Geschäftsmodell vertreten und gutheißen, das unsere Zeit und Mühe nicht wertschätzt, die wir in die Konzeption, die Entwicklung und den Betrieb von Software stecken, wird das nicht nachhaltig und im großen Stil funktionieren. Wir brauchen einen Sinneswandel dahin, dass Software wieder Geld kosten darf und eventuell sogar muss. Das müssen keine überzogenen Preise sein, aber alles auf Dauer ständig kostenlos, das kann es halt auch nicht sein.

Silke: *Das ist im Open-Source-Kosmos aber ein fundamentales Thema, oder?*
Golo: Ja, das stimmt. Aber das ist (wieder einmal) kein technisches, sondern ein kulturelles Problem. Und deshalb gibt es dafür auch keine einfache Lösung, da kann im Prinzip nur jede und jeder einzelne versuchen, für sich im direkten Umfeld einen Unterschied zu machen, und darauf zu hoffen, dass wenn man das immer wieder erklärt, dass es falsch ist, sich ohne Gegenleistung überall gratis zu bedienen, dass das dann irgendwann vielleicht einmal fruchtet. Auch wenn das zugegebenermaßen furchtbar pathetisch klingt, aber es ist moralisch falsch, immer nur zu nehmen, nichts zu geben, und die Zeit anderer nicht wertzuschätzen.

So bemerkenswert und wichtig der Einfluss von Open Source in den vergangenen Jahrzehnten war, so groß ist inzwischen auch der Schaden, der durch die kostenlose Zugänglichkeit geschaffen wurde. Und daher geht es bei Open Source eigentlich um etwas ganz anderes – Open Source bedeutet nicht automatisch auch, dass Software kostenlos sein muss, aber leider wird das häufig synonym verwendet. Das ist aus meiner Sicht eines der ganz großen Probleme nicht nur der IT, sondern vor allem der Gesellschaft, weil es dabei darum geht, die Kosten des Individuums zu minimieren, was aber unterm Strich zulasten der gesamten Gesellschaft geht.

Silke: *Heise hat jetzt eine eigene Instanz. Was rätst du anderen: Bestehende Instanz nutzen, Hoster suchen für eigene Instanz oder von der Pike auf selbst die Infrastruktur betreiben?*
Golo: Eine bestehende Instanz zu nutzen ist, finde ich, die für das Gesamtsystem am wenigsten attraktive Option. Eigentlich braucht es viel mehr kleine Instanzen, und nicht einige wenige große. Der Punkt ist aber, dass die meisten Menschen dazu gar nicht in der Lage sind. Wie soll jemand außerhalb der IT-Branche auch nur ansatzweise eine eigene Mastodon-Instanz betreiben? Das geht schlichtweg nicht. Insofern wird die Masse der Accounts tatsächlich derart sein, dass sie auf einer bestehenden Instanz mitlaufen.

Wer aber in der Lage ist, das selbst zu machen, steht vor der Frage, ob man die Infrastruktur selbst aufbaut oder einkauft. Es gibt eine Handvoll Mastodon-as-a-Service-Anbieter, allerdings nehmen viele davon derzeit (unter anderem aus Kapazitätsgründen) keine neuen Kundinnen und Kunden an, sodass man schon Glück haben muss, damit das klappt. Außerdem gilt es hier sehr genau hinzugucken, wo das Hosting zum Beispiel erfolgt, wie es mit der DSGVO-Einhaltung ausschaut, ob verschlüsselt wird, wie und wann gebackupt wird, welche Garantien es gibt, und so weiter.

Silke: *Also hat es offenbar Vorteile, eine eigene Instanz zu betreiben – sofern man technisch dazu in der Lage ist?*
Golo: Zumindest hat man mit einer eigenen Instanz mehr Kontrolle. Das ist der Vorteil daran. Der Nachteil ist, dass es zeitaufwändig und mühsam ist, eine eigene Instanz zu betreiben, zumindest wenn man den Anspruch hat, dass es wie vorhin erwähnt skalierbar, ausfallsicher und verschlüsselt ist. Da ist einfach die Frage, ob man diesen Aufwand auf sich nehmen möchte. Das lässt sich nicht pauschal beantworten, das muss am Ende des Tages jede und jeder selbst entscheiden.

Tatsächlich gibt es aber auch noch eine vierte Option: Man kann die ganze Situation nämlich auch einmal zum Anlass nehmen, darüber nachzudenken, ob man das Ganze wirklich braucht. Ist der Nutzen der sozialen Medien so hoch, dass er den Preis rechtfertigt? Das ist eine Frage, die ich mir stelle, seit ich meinen persönlichen Twitter-Account stillgelegt habe, aber so verkehrt fühlt es sich zumindest für mich nicht an, nicht mehr auf Twitter, Mastodon & Co. vertreten zu sein. Vielleicht hatte ein solcher Dienst auch einfach seine Zeit, und vielleicht überschätzen wir die Relevanz von sozialen Medien, und vielleicht wäre es gut, davon mehr Abstand zu nehmen.

Silke: *Golo Roden, wir danken für das Gespräch!*

Das Interview führte Silke Hahn, Redakteurin bei iX und heise Developer. Silke ist auf Twitter @_SilkeHahn sowie auf Mastodon zu finden, dort unter sigmoid.social/@silke. (sih)

Quellen

Weitere Informationen zum Softwarearchitekten Golo Roden finden sich auf seiner Website sowie auf YouTube: ix.de/z1wd.

Impressum We Are Developers!

Redaktion: iX | heise Developer
Telefon: 0511 5232-387, **E-Mail:** post@ix.de

Herausgeber: Ansgar Heise

Chefredakteur (verantwortlich für den Textteil): Dr. Oliver Diedrich

Konzeption und redaktionelle Leitung: Silke Hahn (sih@ix.de) -367

Redaktion: Nicole Bechtel, Maika Möbus

Autoren dieser Ausgabe:
Moritz Förster, Silke Hahn, Till Jaeger, Miguel Ojeda, Golo Roden, Robert Ruzitschka, Clemens Sielaff, Jarred Sumner, Timo Zander

DTP-Produktion:
Lisa Hemmerling, Heise Medienwerk, Rostock

Korrektorat:
Lara Bögner, Marei Stade, Heise Medienwerk, Rostock

Titelbild:
Lisa Hemmerling, Heise Medienwerk

Verlag:
Heise Medien GmbH & Co. KG,
Postfach 61 04 07, 30604 Hannover; Karl-Wiechert-Allee 10, 30625 Hannover;
Telefon: 0511 5352-0, Telefax: 0511 5352-129

Geschäftsführer:
Ansgar Heise, Beate Gerold

Mitglieder der Geschäftsleitung:
Jörg Mühle, Falko Ossmann

Anzeigenleitung (verantwortlich für den Anzeigenteil):
Michael Hanke (-167), E-Mail: michael.hanke@heise.de,
www.heise.de/mediadaten/ix

Leiter Vetrieb und Marketing:
André Lux

Druck:
Dierichs Druck + Media GmbH & Co. KG,
Frankfurter Straße 168, 34121 Kassel

Inserenten

Die hier abgedruckten Seitenzahlen sind nicht verbindlich. Redaktionelle Gründe können Änderungen erforderlich machen.

> KI-generierte Bilder: Kampf um das Urheberrecht

Dr. Till Jaeger

Noch ist unklar, wie das Urheberrecht künftig mit KI-generierten Werken umgehen wird. Zu KI-generiertem Content sind aus juristischer Sicht viele Fragen offen.

Das U.S. Copyright Office (USCO) hat im September 2022 die Graphic Novel „Zarya Of The Dawn" für die Künstlerin und KI-Forscherin Kris(tina) Kashtanova registriert (*Anm. d. Red.: Kashtanova identifiziert sich als nonbinär*). Die Bildergeschichte wurde teilweise mithilfe des Machine-Learning-Modells Midjourney produziert und ist damit der erste öffentlich bekannt gewordene Fall, in dem KI-Kunst explizit Urheberrechtsschutz zugestanden wurde. Zwar ist eine Registrierung beim U.S. Copyright Office keine Voraussetzung für urheberrechtlichen Schutz, aber sie bringt praktische Vorteile wie einen Anscheinsbeweis für die Schutzfähigkeit vor Gericht und ist Voraussetzung für Klagen wegen Urheberrechtsverletzungen in den USA.

Kashtanova hat mit den Online-Berichten zu der Registrierung erhebliche Aufmerksamkeit erregt, die wiederum das U.S. Copyright Office zu einer Überprüfung der Entscheidung bewogen hat. Denn urheberrechtlicher Schutz setzt in den USA, wie auch in Deutschland, ein von einem Menschen geschaffenes Werk voraus. In den Worten des deutschen Urheberrechtsgesetzes: eine „persönliche geistige Schöpfung" (§ 2 Abs. 2 UrhG). Abbildung 1 zeigt das Cover der Graphic Novel.

In einer Stellungnahme beruft sich Kashtanova darauf, die Texte der Graphic Novel vollständig selbst verfasst und Midjourney als Werkzeug für das Erstellen der Bilder verwendet zu haben. Ähnlich wie ein Fotograf eine Szene gestalten könne und die Kamera als ein Werkzeug zur Abbildung verwendet werde, sei die Bildgenerierung durch Midjourney von der Künstlerin in einem aufwendigen und iterativen Prozess gestaltet worden, wie der Anwalt von Kashtanova in seiner Gegendarstellung an das U.S. Copyright Office im Detail darlegt.

Am 21. Februar 2023 hat das U.S. Copyright Office den Urheberrechtsschutz für die Bildergeschichte bestätigt und dabei deutlich gemacht, dass sowohl der Text als auch die Anordnung von Bild- und Textelementen schutzfähig sind, nicht aber die einzelnen, mithilfe von Midjourney erzeugten Bilder. Der Fall zeigt anschaulich die Probleme des klassischen Urheberrechts im Umgang mit KI-generierten Texten, Bildern oder anderen Werkgattungen. Noch im Februar 2022 hatte das U.S. Copyright Office einen Antrag von Steven Thaler auf Registrierung des Bildes „A Recent Entrance to Paradise" mit der Begründung abgelehnt, rein maschinell erzeugte Arbeiten seien „original work of authorship" im Sinne des US-Urheberrechtsgesetzes (17 U.S.C. § 102a), da dies stets eine menschliche Gestaltung voraussetze.

Im Gegensatz zu Kashtanova hatte Thaler die Registrierung jedoch damit begründet, dass das Bild „autonom von

In a Nutshell

- KI-generierter Content wirft zahlreiche urheberrechtliche Fragen auf, die noch unbeantwortet sind. Urheberrechtlich ist zwischen Trainingsmaterial, KI-Modell und generiertem Output zu unterscheiden.
- KI kann sowohl als Tool bei der eigenen kreativen Gestaltung eingesetzt werden als auch als einfacher Generator von Inhalten – entsprechend unterschiedlich kann die urheberrechtliche Beurteilung sein.
- Noch unklar ist, wie das Urheberrecht künftig mit KI-generierten „Werken" umgehen wird, insbesondere vor dem Hintergrund, dass der Entstehungsprozess und damit der Einsatz von KI regelmäßig nicht erkennbar ist.

einem Computeralgorithmus erzeugt wurde, der auf einer Maschine ausgeführt wird“, und dass das computergenerierte Werk als Auftragsarbeit für den Eigentümer der „Kreativitätsmaschine“ zu registrieren sei. Im Fall Thaler ging es also um die Frage, ob der Urheberrechtsschutz auf kreative Leistungen eines Menschen beschränkt ist, was vom U.S. Copyright Office eindeutig bejaht wurde, während es im Fall Kashtanova um die Frage geht, ob und unter welchen Voraussetzungen Bilder trotz des Einsatzes von KI eine kreative Leistung des Anwenders der KI darstellen können.

Menschliche Kreativität als Ausgangspunkt

Die entscheidende Frage wird sein, ob es sich bei den erstellten Bildern – oder anderem mittels KI erzeugten Outputs – im Kern um ein Werk menschlicher Urheberschaft handelt, bei dem der Computer lediglich als Hilfsmittel gedient hat, oder ob die traditionellen Elemente der Urheberschaft in dem Werk (wie der künstlerische Ausdruck) von einer Maschine und nicht von einem Menschen erschaffen wurde.

Urheberrechtlich ist der Ausgangspunkt in den USA und in Deutschland sehr ähnlich. Gemäß 17 U.S.C § 102 sind

Quelle: AI Comic Books (Kashtanova)

>> Buchcover zu Kris(tina) Kashtanova, „Zarya of the Dawn“ (Abb. 1)

Gegenstand des Urheberrechtsschutzes „original works of authorship fixed in any tangible medium of expression“, in § 2 Abs. 2 UrhG heißt es: „Werke im Sinne dieses Gesetzes sind nur persönliche geistige Schöpfungen.“ Aus diesen Definitionen wird seit jeher geschlossen, dass nur eine menschliche Leistung urheberrechtlichen Schutz genießen kann. Hierbei geht es um das Menschsein: Ein Affen-Selfie ist daher schon von vornherein von einem solchen Schutzrecht ausgeschlossen. Dies gilt ebenso für reine Maschinenerzeugnisse.

Technik als Hilfsmittel

Allerdings können sich menschliche Urheber durchaus technischer Mittel bedienen, wenn sie diese als Hilfsmittel

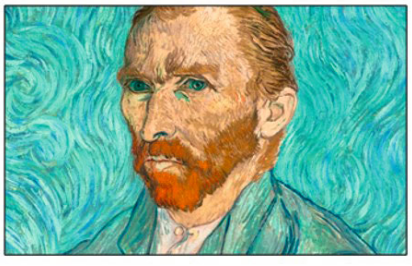
Maschinell erstellte Werke

§ 9 Abs. 3 des Copyright, Designs and Patents Act 1988 (CDPA): Im Falle eines literarischen, dramatischen, musikalischen oder künstlerischen Werks, das mit Hilfe eines Computers erstellt wurde, gilt als Urheber die Person, die die für die Schaffung des Werks erforderlichen Vorkehrungen getroffen hat.

steuernd einsetzen, wie das vielfach bei einem Computer oder einer Kamera der Fall ist. So kann etwa das steuernde Element darin bestehen, dass für ein Foto ein Bildausschnitt gewählt wird, selbst wenn die Aufnahme – wie bei einer Wetterkamera – dann automatisiert erfolgt. Für den urheberrechtlichen Schutz reicht eine menschliche Steuerung bei der Erstellung eines Werkes alleine noch nicht aus, das Werk benötigt auch die erforderliche „Schöpfungshöhe".

Dies bedeutet, dass ein Gestaltungsspielraum bestehen muss und dieser Gestaltungsspielraum in einer Weise genutzt wird, die sich vom rein Handwerklichen und Alltäglichen abhebt. Die Gerichte sind hier durchaus großzügig und lassen schon die sogenannte „kleine Münze" gelten, also Werke von einfacher Qualität, die gerade noch eine individuelle Gestaltung enthalten. Natürlich sind die Grenzen hier fließend und unterschiedliche Gerichte können bei einem vergleichbaren Sachverhalt zu abweichenden Ergebnissen kommen.

Eine solche Schöpfungshöhe ist bei einer Wetterkamera, die automatisiert Bilder aufnimmt, regelmäßig nicht gegeben. Allerdings hat der deutsche Gesetzgeber auch für Lichtbilder ohne die erforderliche Schöpfungshöhe in § 72 Urheberrechtsgesetz (UrhG) ein eigenes Leistungsschutzrecht geschaffen, das dem Urheberrecht sehr ähnlich ist, aber eine kürzere Schutzdauer besitzt. Dies zeigt, dass der Gesetzgeber in der Vergangenheit auf Abgrenzungsprobleme mit neuen Schutzrechten reagiert hat, so beispielsweise bei Fotografien auf das praktische Problem, dass oft nur sehr schwer zu beurteilen ist, ob es sich um ein bloßes Knipsbild handelt oder ob der Fotograf gestalterisch tätig geworden ist. Ob eine solche Ausweitung der Schutzrechte immer sinnvoll ist, kann angesichts des Abmahngeschäfts mit Trivialbildern durchaus infrage gestellt werden.



>> Selbstporträt Vincent van Gogh, 1889. – Rechts: KI-generiertes Bild zum Prompt „self portrait by Vincent van Gogh, intense colors" (Abb. 2).

Gesetzlicher Schutz für computergenerierte Werke?

Im Vereinigten Königreich existiert eine Regelung, die den Schutz KI-generierten Outputs zu erfassen scheint: § 9 Abs. 3 des Copyright, Designs and Patents Act 1988 (CDPA) sieht auch für maschinell erstellte Werke einen Urheberrechtsschutz vor:

Allerdings wurde diese Regelung nicht erst im Rahmen der aktuellen KI-Diskussion in den Copyright, Designs and Patents Act eingefügt, sondern existiert schon seit 1988. Offenbar wollte man damit klarstellen, dass auch bei der Verwendung von Computern urheberrechtlich geschützte Werke entstehen können, etwa bei zufallsgesteuerten Sequenzersystemen zur Verwendung mit Synthesizern. Der bislang einzige bekannte Gerichtsfall betrifft auch keine KI-generierten Werke, sondern ein Computerspiel, bei dem wesentliche Gestaltungen von Menschen vorgenommen wurden (konkret geht es um folgenden Rechtsstreit: Nova Productions Ltd v Mazooma Games Ltd & Ors).

Daher verwundert es nicht, wenn die Regelung bei der Anwendung auf aktuelle KI-Modelle Fragen aufwirft, nämlich: Welche Person die „erforderlichen Vorkehrungen getroffen" hat, wenn ein Bild mithilfe eines Modells wie Stable Diffusion erstellt wird. Hier kommen sowohl die Entwickler von Stable Diffusion als auch deren Nutzer in Betracht. Einfacher wird die Rechtslage also auch nicht in Ländern wie dem Vereinigten Königreich, Irland und Neuseeland, die eine solche Regelung besitzen.

Sind KI-generierte Bilder schützbar?

Betrachtet man den aktuellen Stand des Urheberrechts und seiner Auslegung in Deutschland, dann werden viele Bilder, die mit Stable Diffusion, DALL-E oder Midjourney erstellt wurden, wohl keinen Urheberrechtsschutz besitzen. Denn der konkrete Output eines Text-to-image Modells lässt sich durch einfache Prompts, also Texteingaben wie zum Beispiel „a photograph of an astronaut riding a horse", nicht so weitgehend steuern, dass er als Ergebnis eines gestalterischen Prozesses eines Menschen angesehen werden kann. Dies zeigt schon der Umstand, dass die Eingabe von Prompts im Regelfall nicht zu einem von einem Menschen planbaren Ergebnis führt, sondern nur zu der Reproduktion eines Stils oder der Wieder-

gabe von vorgegebenen Bildelementen wie einem Gegenstand, einer Person oder einer Situation. Dabei existieren aber noch erhebliche Varianz und Unvorhersehbarkeit.

Letztlich ist das einer der Gründe, warum das Experimentieren mit diesen Modellen eine so erhebliche Faszination ausübt. Ein bloßer Malstil ist urheberrechtlich nicht schutzfähig, sodass hier keine Handhabe gegen Nachahmer besteht (siehe Abb. 2). Dies bedeutet jedoch nicht, dass Stable Diffusion und Co. nicht auch als Tools eingesetzt werden könnten, bei denen menschliche Kreativität weiterhin eine wesentliche Rolle spielt. Hier kommen mehrere Konstellationen in Betracht.

Outpainting und Inpainting: Weiterbearbeitung und Urheberrecht

So können etwa KI-generierte Bilder von einem Menschen in einem Bildbearbeitungsprogramm so modifiziert werden, dass dies zu einem urheberrechtlich geschützten Werk führt – oder ein schon vorbestehendes Foto oder Bild wird mittels KI im Wege des In- oder Outpainting verändert. So wird beim Outpainting ein Bild über den ursprünglichen Rand hinaus fortgesetzt, wie das Beispiel des „Mädchens mit dem Perlenohrring“ von Vermeer zeigt, das von dem OpenAI-Mitarbeiter August Kamp mithilfe von DALLE·E ergänzt wurde (siehe Abb. 3).

Urheberrechtlich sind die Bilder von Vermeer nicht mehr geschützt, weil seit seinem Tod mehr als 70 Jahre vergangen sind. Was ist aber mit Bildern, für die noch Urheberrechtsschutz besteht? Hier erlaubt § 23 UrhG zwar die Erstellung von Bearbeitungen eines vorbestehenden Werkes, aber für die Veröffentlichung, wie etwa die Wiedergabe per Twitter, ist eine Erlaubnis der Urheber des Ausgangswerkes (oder von deren Erben) erforderlich.

Zustimmung erforderlich

§ 23 UrhG: Bearbeitungen oder andere Umgestaltungen eines Werkes, insbesondere auch einer Melodie, dürfen nur mit Zustimmung des Urhebers veröffentlicht oder verwertet werden. Wahrt das neu geschaffene Werk einen hinreichenden Abstand zum benutzten Werk, so liegt keine Bearbeitung oder Umgestaltung im Sinne des Satzes 1 vor.

Persönliche geistige Schöpfung: als Bearbeiter ein Urheberrecht erwerben

Bearbeiter können gemäß § 3 UrhG ein eigenes Bearbeiterurheberrecht erwerben, wenn die Änderungen selbst wieder

Quelle: OpenAI / August Kamp

>> Outpainting von August Kamp für OpenAI/ DALL·E – Original: „Mädchen mit dem Perlenohrring“ von Johannes Vermeer diente als Vorlage (Abb. 3).

auf einer „persönlichen geistigen Schöpfung“ beruhen. Damit kann bei Bildern, deren Schutzfrist abgelaufen ist, neuer Urheberrechtsschutz nur entstehen, wenn ein Mensch eine eigene Gestaltung – zum Beispiel durch Einsatz von KI als Werkzeug – vorgenommen und das Gestalten nicht alleine der KI überlassen hat.

Beim Inpainting werden nur Teile eines vorbestehenden Bildes verändert. Hier gilt Ähnliches wie beim Outpainting: Ein bestehendes Urheberrecht des vorbestehenden Werkes ist zu beachten. Ein Bearbeiterurheberrecht an dem veränderten Bild entsteht nur, wenn KI als gestaltendes Tool eingesetzt wurde, da ist die Lage nicht anders als beim Einsatz von Photoshop und anderen Programmen zur Bildbearbeitung. Besonders interessant ist die Möglichkeit, in einem Grafikprogramm oder von Hand eine Skizze zu erstellen und diese dann durch ein KI-Tool ausgestalten zu lassen (siehe Abb. 4).

Die Verwendung eines KI-Modells als Tool ist aber nicht auf die Fälle des In- und Outpaintings oder der Bearbeitung vorbestehender Bilder beschränkt. Denn auch bei dem bloßen Einsatz von Prompts lässt sich das Ergebnis zum einen durch die Auswahl der Vorgaben durch den Text als auch durch weitere Konfigurationseinstellungen beeinflussen und damit gestalten. So können per Prompt für fotorealistische Ergebnisse etwa Vorgaben wie ein Kameramodell, das zu verwendende Objektiv, Lichtverhältnisse und die Tiefenschärfe gemacht werden, die das Ergebnis erheblich beeinflussen (siehe Abb. 5).

Auch die Reihenfolge der in dem Prompt verwendeten Begriffe spielt für die Gewichtung und damit das Ergebnis eine Rolle. Zudem können mithilfe „negativer Prompts“ unerwünschte Effekte oder Gestaltungen unterbunden werden. Zusätzlich zu der Beeinflussung durch Prompts hängt das Ergebnis eines Text-to-image-Prozesses von den Einstellungen der Parameter in der eingesetzten Software ab. Durch die Classifier-Free Guidance Scale lässt sich bestimmen, inwieweit das Modell dem Prompt folgen soll oder „kreative Freiheit“ besitzt. Entsprechend stark oder schwach finden sich die Vorgaben aus dem Prompt in dem Bild wieder.

Je umfangreicher die Vorgaben im Text, desto höher muss die Guidance Scale sein, um diese Vorgaben auch erkennbar umzusetzen. Mithilfe der „Steps“ kann die Anzahl der von dem Modell durchzuführenden Denoising-Schritte be-

Quelle: GitHub / CompVis, Team Stable Diffusion

>> Interessante Möglichkeit: eine Skizze erstellen und diese durch ein KI-Tool ausgestalten lassen (Abb. 4).

stimmt werden, was sich ebenfalls auf das Ergebnis auswirkt, wobei bei einer höheren Anzahl die Bilder ausdifferenzierter werden. Durch die Angabe des „Seeds“ wiederum lässt sich der Startpunkt für die Initialisierung vorgeben. Dies ist deswegen von Bedeutung, weil durch die Angabe eines konkreten Seeds bei ansonsten gleichen Parametern und Prompt ein identisches Bild erzeugt werden kann. Entsprechend ist durch kleinere Variationen des Prompts eine Detailgestaltung des Outputs möglich (Beispiele bei GetImg.AI). Auch durch die „img2img“-Funktion, bei der ein zunächst generiertes Bild als Input für die weitere Bearbeitung verwendet wird, kann eine Gestaltung des Outputs vorgenommen werden.

Durch die Auswahl eines Diffusion-Samplers wird schließlich die Art der Berechnung des nächsten Denoising-Schritts beeinflusst, was wiederum Einfluss auf die erforderlichen Schritte und die Dauer der Berechnung hat.

Gestaltung als iterativer Prozess

Daher überrascht es nicht, dass der Anwalt von Kris Kashtanova in seiner Gegendarstellung einige der von Kashtanova eingesetzten Gestaltungsmöglichkeiten aufgreift und die Arbeit als iterativen Prozess der Auswahl und Ausdifferenzierung der Bilder beschreibt. In der Tat kennt das Urheberrecht die Auswahl als ein schöpferisches Element, wie § 4 UrhG für Sammelwerke zeigt. So kann zum Beispiel durch die Auswahl und Anordnung von Gedichten oder Musikstücken ein eigenes Recht am Sammelwerk

>> Beispiele für ein fotorealistisches KI-Bild und dessen Weiterbearbeitung. Links: Verwendung von 20 Steps (dem Eichhörnchen fehlt ein Bein...). – Rechts: Verwendung von 45 Steps bei gleichem Prompt und Seed (das Eichhörnchen schaut realistischer aus). (Abb. 5).

Data Mining: gesetzliche Ausnahme

§ 44b UrhG

(1) Text und Data Mining ist die automatisierte Analyse von einzelnen oder mehreren digitalen oder digitalisierten Werken, um daraus Informationen insbesondere über Muster, Trends und Korrelationen zu gewinnen.

(2) Zulässig sind Vervielfältigungen von rechtmäßig zugänglichen Werken für das Text und Data Mining. Die Vervielfältigungen sind zu löschen, wenn sie für das Text und Data Mining nicht mehr erforderlich sind.

(3) Nutzungen nach Absatz 2 Satz 1 sind nur zulässig, wenn der Rechtsinhaber sich diese nicht vorbehalten hat. Ein Nutzungsvorbehalt bei online zugänglichen Werken ist nur dann wirksam, wenn er in maschinenlesbarer Form erfolgt.

entstehen. Es ist daher folgerichtig, dass Kashtanova ein Urheberrecht an ihrer Bildergeschichte zugestanden wurde, nicht aber an jedem einzelnen mit KI erzeugten Bild.

Angesichts der schon komplizierten Abgrenzung von urheberrechtlich nicht geschützter KI-Kunst und geschützten Werken, bei denen KI als Tool zur Gestaltung eingesetzt wurde, stellen sich einige ungeklärte Folgefragen. Wenn dem Ergebnis nicht mehr anzusehen ist, ob eine menschliche Gestaltung vorgenommen wurde, stellt sich zum Beispiel das Problem der Beweislast, wenn eine Urheberrechtsverletzung geltend gemacht wird. Besteht eine Vermutung für eine menschliche Gestaltung oder muss der Schöpfungsprozess dokumentiert werden, wie dies Kashtanova zumindest teilweise getan hat? Ist es überhaupt noch sinnvoll, für Urheberrechtsschutz auf eine menschliche Gestaltung abzustellen? Was macht dann noch den Kern des Urheberrechts aus?

Goldgräberstimmung bei KI-Anbietern, Anwälten, Künstlern

Noch intensiver als die Frage nach der Schutzfähigkeit des Outputs wird derzeit allerdings die Frage diskutiert, ob eine Zustimmung der Urheber der Werke, die zum Training eines KI-Modells verwendet wurden, eingeholt werden muss. Erste Urheberrechtsklagen sind bereits anhängig, etwa von den Künstlerinnen Sarah Andersen, Kelly McKernan und Karla Ortiz gegen Stability AI und Midjourney in Kalifornien. Als Begründung für die Urheberrechtsverletzung wird angeführt, dass die mit Stable Diffusion und Midjourney erzeugten Bilder Bearbeitungen der als Trainingsmaterial verwendeten Bilder seien und letztlich komplexe „Collage-Tools“ darstellten.

Pikanterweise wurde die Klage als Class Action, also als Sammelklage, eingereicht, um möglichst vielen Künstlern die Möglichkeit zu geben, sich zu beteiligen und entsprechend hohe Schadenersatzforderungen geltend zu machen. Bei der Class Action muss eine „Gruppe“ definiert werden. Wenn man dazu gehört, kann man vom Ergebnis profitieren. Auch Getty Images hat vor dem High Court in London Klage gegen Stability AI wegen Urheberrechtsverletzung eingereicht.

Künstler protestieren auf Artstation

Der Widerstand eines Teils der Kunstszene drückt sich aber nicht nur in Klagen aus. Auf der Kunstplattform Artstation wurde eine Protestaktion mit dem Label „No to AI-generated images“ initiiert – unter anderem, um das weitere Training mit den dort gezeigten Bildern zu erschweren. Denn auf Artstation wird in Prompts gerne Bezug genommen, um die generelle Bildqualität zu verbessern, da die dort veröffentlichten Bilder umfangreich für das Training genutzt wurden: „Trending on Artstation“ ist ein häufig verwendeter Promptzusatz (siehe Abb. 6).

Trainingsdaten: eine Grauzone

Schaut man sich das Thema Trainingsdaten genauer an, sind zwei Fälle auseinanderzuhalten: Wenn Bilder, Texte oder Programmcode lediglich zum Training eines neuronalen Netzes verwendet werden, ist zunächst auch nur dafür eine urheberrechtliche Erlaubnis erforderlich. Wer urheberrechtlich geschützte Bilder kopiert, braucht eine Erlaubnis. Entweder durch eine Lizenz (wie CC-lizenzierte Bilder) oder gesetzlich. Gesetzlich ist der § 44b UrhG die Erlaubnis – die gerade deswegen eingeführt wurde, weil Big Data Analysen von urheberrechtlich geschützten Inhalten sonst faktisch nicht möglich sind.

Anders ist die Situation, wenn ein vorbestehendes Werk erkennbar auch im Output enthalten ist, etwa weil es aus einer Datenbank für ein Outpainting verwendet wurde. Hier wird im Regelfall die Erlaubnis des Urhebers benötigt. Die neue „Pastiche“-Schranke des § 51a UrhG enthält allerdings eine gesetzliche Erlaubnis, die in solchen Fällen einschlägig sein könnte. Zu dem Begriff Pastiche gibt es noch keine Definition. Die Gesetzesbegründung spricht von einer Erlaubnis von „modernen Formen transformativer Nutzung urheberrechtlich geschützter Inhalte insbesondere im digitalen Umfeld“ (BT-Drs. 19/27426, S. 89).

Auch wenn der Begriff des „Pastiche“ noch keine klaren Konturen besitzt, geht es im Ergebnis um „eine Auseinandersetzung mit einem vorbestehenden Werk, das erkennbar ist, aber nicht bloß zur weiteren Verwertung kopiert wird“. In den Worten der Gesetzesbegründung: „Anders als bei Parodie und Karikatur, die eine humoristische oder verspottende Komponente erfordern, kann diese beim Pastiche auch einen Ausdruck der Wertschätzung oder Ehrerbietung für das Original enthalten, etwa als Hommage.“ Dies dürfte zu weiteren offenen und von den Gerichten zu klärenden Auslegungsfragen führen: Eine ausführliche Darstellung dazu findet sich bei Kreutzer, Gutachten „Der Pastiche im Urheberrecht“.

Copyright und ML

Für das bloße Training stellt sich die Rechtslage in den USA und Deutschland ganz unterschiedlich dar. In den USA wird Machine Learning anhand der allgemeinen Regeln behandelt und es dürfte die Frage im Vordergrund stehen, ob die Verwendung von Bildern zum Training unter Fair Use fällt und damit ohne Lizenz erlaubt ist. Im deutschen Urheberrecht sieht § 44b UrhG eine besondere gesetzliche Ausnahme für Data Mining vor, die hier als Erlaubnis in Betracht kommt.

Die Regelung des § 44b UrhG basiert auf europäischem Recht, und

Quelle: mit Stable Diffusion erstellt vom Autor

>> Textprompt: „Final battle of authors against AI, trending on Artstation“ (Abb. 6).

zwar auf der DSM-Richtlinie (Digital Single Market) zum Urheberrecht im digitalen Binnenmarkt. Die Verwendung von Trainingsdaten im Rahmen des Machine Learnings dürfte ein typischer Anwendungsfall sein, da die automatisierte Analyse der Mustererkennung dient.

Machine Learning gesetzlich erlaubt …

Damit können urheberrechtlich geschützte Trainingsdaten (wie Bilder aus dem Internet) lizenzfrei für das Machine Learning verwendet werden. Die Trainingsdaten müssen nach Abschluss des Trainings lediglich gelöscht werden. Die Urheber haben jedoch die Möglichkeit eines Opt-out, wenn sie ihre Werke nicht für ein Training zur Verfügung stellen möchten. Dies soll durch einen maschinenlesbaren Vorbehalt geschehen. Die Details dieser Regelung, die erst 2021 in das Urheberrechtsgesetz aufgenommen wurde, sind allerdings noch recht unklar. Kann ein Urheber die Nutzung seines Werkes verhindern, wenn ein Vorbehalt nicht möglich ist, beispielsweise bei einer urheberrechtswidrigen Nutzung auf einer fremden Website? Reicht es aus, wenn die AGB einer Website die Nutzung zu Trainingszwecken verbieten, da auch AGB maschinenlesbar sind, oder muss die entsprechende Information in einer robots.txt enthalten sein? Auch bei der Auslegung der Voraussetzungen der Schranke des § 44b Abs. 1 UrhG gibt es noch Auslegungsfragen, die einer gerichtlichen Klärung bedürfen.

In den USA gibt es keine dem § 44b UrhG vergleichbare Schrankenregelung. Teilweise wird angenommen, dass die allgemeine Schranke des Fair Use greift, die wiederum in Deutschland und anderen EU-Staaten nicht existiert. Auch für die Rechtslage in den USA bleibt abzuwarten, welchen Weg die Gerichte letztlich einschlagen werden. Für die aktive Modellentwicklung stellt sich damit die interessante Frage, welche Urheberrechtsordnung die größere Rechtssicherheit bietet. Denn im internationalen Urheberrecht gilt das Schutzlandprinzip, wonach das Urheberrecht des Staates anzuwenden ist, für den um Schutz nachgesucht wird. Vereinfacht bedeutet dies, dass deutsches Urheberrecht anwendbar ist, wenn das Training in Deutschland stattfindet, weil es von hier aus gesteuert wird oder die Trainingsdaten in Deutschland verarbeitet werden. Es ist auch durchaus möglich, dass mehrere Urheberrechtsordnungen parallel anwendbar sind, weil eine Nutzungshandlung in mehreren Staaten stattfindet, wenn etwa Server an verschiedenen Standorten genutzt werden. In diesem Fall muss die Handlung in allen anwendbaren Urheberrechtsordnungen zulässig sein. Bei einem Training in den USA und in Deutschland müssten dann sowohl die Voraussetzungen des § 44b UrhG als auch die des Fair Use erfüllt sein.

Bei den bereits initiierten Gerichtsverfahren könnte wesentlich sein, ob aus einem Modell die für das Training verwendeten Bilder wieder generiert werden können, und zwar in einer Weise, dass die konkreten Bilder reproduzierbar sind und sich nicht nur die verwendeten Bilddateien identifizie-

Beyond Stable Diffusion: die Lizenzfrage

Dass komplexe technische Neuentwicklungen auch neue urheberrechtliche Fragen aufwerfen, zeigt die rechtswissenschaftliche Diskussion um die Schutzfähigkeit von künstlichen neuronalen Netzen (KNN). KI-Modelle bestehen nicht nur aus Computerprogrammen, sondern zumeist auch aus einem oder mehreren trainierten KNNs. Für Stable Diffusion existieren bereits Marktplätze mit unterschiedlich trainierten Netzen. Die Ursprungsmodelle wurden mit NSFW-Filtern (Not-Safe-For-Work) trainiert, die unerwünschte Inhalte aussortieren. Daher sollten damit generierte Bilder keine Hakenkreuze, jugendgefährdende oder sonstige illegale Darstellungen enthalten. Die Lizenz des Modells, die CreativeML Open RAIL-M, ist deshalb auch keine klassische Open-Source-Lizenz, die die unbeschränkte Nutzung gestattet. Sie verbietet den Einsatz des Modells für bestimmte aufgelistete Verwendungen. Jedoch kann jedermann diese Modelle weiter trainieren, auch ohne Filter, und die Ergebnisse – meist als Checkpoint-Dateien – wieder anbieten. Es ist offensichtlich, dass die Beschränkungen der CreativeML Open RAIL-M-Lizenz dabei nicht immer eingehalten werden.

betterCode() building IoT dt <webdev> » Continuous Lifecycle » [Container Conf]

data2day enterPy enterJS // heise devSec() Herbstcampus

inside agile Minds Mastering Machines

ONLINE & VOR ORT

DEVELOPER März – Juli 23 KONFERENZEN

Weitere Informationen unter:
heise.de/developer

Veranstalter:

iX heise Developer dpunkt.verlag

- **30.03. + 27.04.2023**
  betterCode() API 2023 (Online)
  api.bettercode.eu
- **26. – 27.04.2023**
  building IoT (München)
  buildingiot.de
- **10. – 11.05.2023**
  Minds Mastering Machines (Karlsruhe)
  m3-konferenz.de
- **16.05.2023**
  heise devSec Thementag (Online)
  Zwei-Faktor-Authentifizierung
  heise-devsec.de
- **16.06.2023**
  Flight Levels Day (Online)
  fld.inside-agile.de
- **20. – 23.06.2023**
  CloudLand (im Phantasialand/Brühl)
  cloudland.org
- **21. – 22.06.2023**
  enterJS (Darmstadt)
  enterjs.de
- **27. – 28.07.2023**
  WeAreDevelopers World Congress (Berlin)
  mit heise Developer Area
  wearedevelopers.com/world-congress
- **CfP bis 07.05.2023**
  Continuous Lifecycle + ContainerConf (Mannheim)
  14. – 16.11.2023
  continuouslifecycle.de
- **CfP bis 08.05.2023**
  data2day (Karlsruhe)
  11. – 12.10.2023
  data2day.de

ren lassen. Denn dann ließe sich argumentieren; die Modelle sind eben nicht nur große, intelligente Bildarchive, die nur mit Zustimmung der Rechteinhaber genutzt werden dürfen. Die bloße Identifikation der Trainingsdaten dürfte hingegen urheberrechtlich belanglos sein: das Modell ist dann ein aliud (lateinisch für „etwas anderes"), es enthält die für das Training verwendeten Bilder nicht und ist von deren Urheberrecht unabhängig. Hier wird also zunächst eine technische Vorfrage zu klären sein, die offenbar durchaus uneinheitlich beantwortet wird und von dem jeweiligen Modell abhängen mag. Einigen Wissenschaftlern ist es jetzt gelungen, einzelne Trainingsbilder aus Stable Diffusion heraus zu berechnen.

Damit stellen sich zwei Fragen: ob bereits die Trainingsbilder urheberrechtswidrig weiterverbreitet werden – oder ob die relevante (und zulässige) urheberrechtliche Nutzungshandlung erst die Neugenerierung durch die Nutzer darstellt. Denn das Modell wird öffentlich zugänglich gemacht und daher eine eigenständige urheberrechtliche Nutzungshandlung vorgenommen, die von § 44b UrhG nicht erfasst ist. Überträgt man diese Frage auf mp3-Dateien mit Musik, wird die Brisanz deutlich. Wenig überzeugend ist hingegen die Behauptung der Künstlerinnen Sarah Andersen, Kelly McKernan und Karla Ortiz in ihrer Klage in Kalifornien, die Modelle von Stable Diffusion und Co. würden „komprimierte Kopien" der Trainingsbilder enthalten. Auch hier wird das richtige Technikverständnis entscheidend für die anschließende rechtliche Beurteilung sein.

Bei der Durchsetzung der Lizenzbedingungen der Lizenz CreativeML Open RAIL-M, aber auch generell, stellt sich die Frage, ob künstliche neuronale Netze (KNN) urheberrechtlich geschützt sind. Anders als herkömmlicher Programmcode enthalten sie keine funktionalen Anweisungen an einen Computer, sondern die Qualität wird durch Gewichte bestimmt. Dies spricht gegen einen Schutz als Computerprogramm. Ob ein Schutz als Datenbank in Betracht kommt, ist umstritten, da anders als bei klassischen Datenbanken der systematische Abruf einzelner Daten keine Rolle spielt. Fraglich ist daher auch, ob das besondere Leistungsschutzrecht für Datenbankhersteller, das es nur innerhalb der EU gibt, Anwendung finden kann. Es bleibt abzuwarten, ob hier eine gesetzgeberische Klärung erfolgt oder ob die Gerichte über diese Fragen zu entscheiden haben.

Quelle: mit Stable Diffusion erstellt vom Autor

>> Textprompt: „Copyright lawyers dream, confused clients asking for advice" (Abb. 7).

KI und Urheberrecht: Work in Progress

Beim Urheberrecht im Bereich generativer Künstlicher Intelligenz ist vieles noch im Fluss. Am 15. Juni 2023 erscheint das neue KI-Sonderheft der iX „KI verstehen und anwenden" mit einer erweiterten Version des Artikels und einem Update zur Rechtslage. Das Heft wird digital und in Print im heise-Shop sowie in gedruckter Form im Buch- und Zeitschriftenhandel erhältlich sein.

Prompts als urheberrechtlich geschützte Werke?

Es ist absehbar, dass urheberrechtliche Ansprüche nicht nur von den Urhebern der Trainingsdaten geltend gemacht werden, sondern auch von den Urhebern der sogenannten Prompts, also der Texteingaben, die die Erstellung eines Bildes steuern sollen. Während einfache Prompts kaum die für einen urheberrechtlichen Schutz erforderliche Schöpfungshöhe erreichen dürften, werden immer umfangreichere und gezieltere Prompts eingesetzt, um das Bildergebnis zu beeinflussen. Es ist daher nicht verwunderlich, dass Prompts bereits zur Handelsware oder als Geschäftsgeheimnis unter Verschluss gehalten werden.

Der Konzeptkünstler Nils Pooker veröffentlicht die für seine Gemälde verwendeten Prompts nicht mehr, während auf dem Marktplatz PromptBase für 9,99 US-Dollar Prompts zur Erstellung von Logo-Designs angeboten werden. Da an die urheberrechtliche Schöpfungshöhe von Texten keine allzu hohen Anforderungen gestellt werden und bereits die sogenannte „kleine Münze" für die Schutzfähigkeit ausreicht, ist durchaus davon auszugehen, dass einige Prompts urheberrechtlichen Schutz genießen. Dies bedeutet jedoch nicht, dass auch die mithilfe des Prompts erstellten Bilder diesem Schutz unterliegen.

Denn mit ein und demselben Prompt können sehr unterschiedliche Bilder erzeugt werden, wenn man für die rele-

vanten Parameter wie Seed, Guidance Scale und Anzahl der Steps unterschiedliche Werte verwendet. Ein Prompt kann auch nicht mit dem Quellcode eines Computerprogramms verglichen werden, der in Objektcode übersetzt wird. Selbst wenn man Text-to-Image-Modelle als eine Art „Übersetzer" verstehen möchte, ist das Bild durch einen Prompt noch nicht hinreichend konkretisiert, um als eine andere Form desselben Werkes angesehen werden zu können.

GitHub Copilot, ChatGPT & Co.

Die aufgeworfenen Urheberrechtsfragen betreffen nicht nur Text-to-Image-Modelle. Auch gegen Microsoft und GitHub Inc, den Anbieter von GitHub Copilot (einem Tool, das Programmierer bei der Softwareentwicklung unterstützen soll), wurde Klage erhoben, weil Open-Source-Software von GitHub für das Training verwendet wurde und „der Output oft eine nahezu identische Reproduktion des Codes aus den Trainingsdaten" sei.

Dies stelle einen Verstoß gegen den Digital Millenium Copyright Act (DMCA) dar, da urheberrechtliche Informationen, also Copyright-Vermerke und Lizenztexte, unterdrückt würden. Das ist insofern überraschend, als Open-Source-Lizenzen stets die freie Nutzung gestatten und ein rein internes Training auch keine Lizenzpflichten auslöst. Wie schon in der Klage gegen Stable Diffusion und Co. vertritt die auf Sammelklagen spezialisierte Kanzlei Joseph Saveri die Urheber. Wie bei der Klage der Künstlerinnen Sarah Andersen, Kelly McKernan und Karla Ortiz wird die Nutzung der eigenen Werke für das Training des Modells – zumindest von einigen Urhebern und Urheberinnen – als unfair angesehen.

Andere Künstler sehen in der neuen Technologie eine Chance für ihre Kunst. Bei ChatGPT und ähnlichen Textgeneratoren werden sich ähnliche oder noch weitreichendere Fragen stellen, da die KI-generierten Texte in allen Lebensbereichen zu finden sein werden und nicht mehr von herkömmlichen Texten unterscheidbar sind. Die anstehenden Gerichtsverfahren werden zeigen, ob es sich bei den Klagen nur um ein Rückzugsgefecht enttäuschter Urheber handelt, deren Arbeit zu einem Teil durch KI ersetzt wird, oder ob die Gerichte im Output eine Verletzung der für das Training verwendeten Werke erkennen werden. (sih)

Quellen

Im Artikel werden zahlreiche rechtliche Quellen genannt. Links zu den Primärquellen sowie zu weiterführenden Beiträgen gibt es unter ix.de/zzsa.

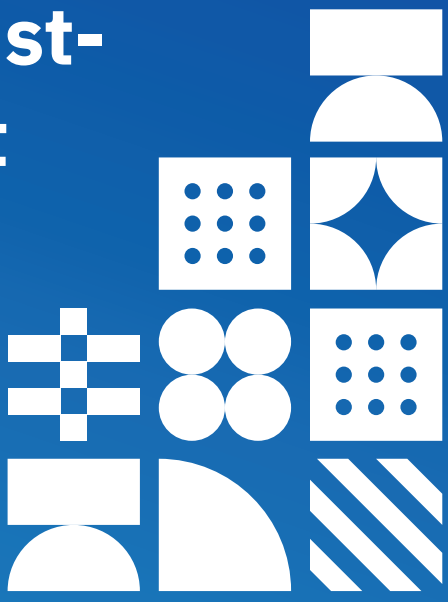
Dr. Till Jaeger

ist Fachanwalt für Urheber- und Medienrecht und seit 2001 Partner der Kanzlei JBB Rechtsanwälte. Einen besonderen Schwerpunkt bilden die Rechtsfragen der Open Source Software, Creative-Commons-Lizenzen und Open Data. Dr. Till Jaeger ist Mitbegründer des „Instituts für Rechtsfragen der Freien und Open Source Software" (ifrOSS) und dort auch wissenschaftlich im Softwarerecht und Urheberrecht tätig.

> Interview: Was die Rust-Zukunft für Linux bringt

Moritz Förster

Rust soll für Linux deutlich mehr Sicherheit bieten sowie mehr Komfort beim Entwickeln von Kernel-Modulen. Die Implementierung der Programmiersprache in den Linux-Kernel steht noch am Anfang – die Kombination wirkt bereits vielversprechend.



Mit Linux 6.1 hielt Rust Einzug in den Kernel. Doch außer einem simplen „Hello, World!“ bietet die erste Beispielimplementierung nichts – und das mit Absicht. Um Module nicht mehr nur in C programmieren zu können, ist noch einiges an Rust-Integration nötig. Der Softwareingenieur Miguel Ojeda betreut das Rust-für-Linux-Projekt und sitzt somit an der Quelle. *iX*-Redakteur Moritz Förster hat bei ihm nachgefragt, welche Features als Nächstes in den Kernel wandern sollen, mit welchen Herausforderungen das Projekt konfrontiert ist und was auf Linuxnutzer zukommt.

Moritz Förster*: Zum Start ist der Rust-Support in Linux experimentell. Welche Features wollt ihr als Nächstes anpacken?*
Miguel Ojeda: Unser primäres Ziel ist es momentan, genügend Abstraktionen in den Haupt-Branch zu bringen, damit später erste Rust-Treiber im Kernel darauf aufbauen können. Diese Abstraktionen packen Kernel-Subsysteme in sichere APIs, damit Rust-Treiber so weit wie möglich in sicherem Rust geschrieben werden können.

Die ersten Treiber und die ihnen zugrundeliegenden Abstraktionen, die den Weg in den Kernel-Haupt-Branch finden werden, sind wahrscheinlich die GPU-Treiber für Asahi Linux, Androids Binder und die NVMe-Treiber. All diese sind nicht trivial und werden als Vorbild für kommende Rust-Kernel-Abstraktionen und -Treiber dienen. Dabei müssen wir auch sicherstellen, dass die relevanten Kernel-Betreuer so weit wie möglich eingebunden sind, zum Beispiel durch Code-Reviews und Kommentare. Dass sie also den Code durch ihren Tree nehmen, und so weiter.

Außerdem sind viele Infrastrukturverbesserungen in Arbeit, darunter ein besseres Build-System für Rust-Code im Kernel und eine weitere Integration in die Testing- und Dokumentationssysteme des Kernels. Zudem hat ARM begonnen, Code für die Rust-Unterstützung der AArch64-Architektur für die Übernahme in den Haupt-Branch des Linux-Kernels einzureichen, basierend auf unserem prototypischen Support für Rust für Linux.

Es sind auch sehr interessante neue Tools in der Entwicklung, darunter klint: Es nutzt den Rust-Compiler als Bibliothek, um zusätzliche statische Codeanalysen im Rust-Kernel-Code durchzuführen. Einer der ersten Tests, an denen wir arbeiten, stellt sicher, dass Rust-Code die Kernel-Locking-Regeln einhält, indem der Preemption Count bereits zur Übersetzungszeit geprüft wird. Auch bei der Toolchain arbeiten wir an Verbesserungen: Hierunter fallen der Kernel-CFI-Support im Rust-Compiler, Security-Compiler-Flags, potenziell bessere Unterstützung für Rust in pahole und einiges mehr.

Aber vielleicht am interessantesten für den Einsatz von Rust außerhalb des Kernels sind die Vorschläge zum Ver-

bessern der Sprache, Bibliothek und des Tooling, die aus dem Projekt heraus entstanden sind. Besonders bemerkenswert ist der Field Projection RFC, der den Zugriff auf Felder spezifischer Wrapper-Typen auf ergonomische und sichere Weise vereinfachen soll.

Moritz Förster*: Manche Nutzer beschweren sich darüber, dass es zu langsam vorangeht. Welchen ungewöhnlichen Herausforderungen muss sich das Projekt stellen?*
Miguel Ojeda: Ich weiß nicht, auf welche Nutzer sich die Frage bezieht – aber einige Leute vertreten die Ansicht, dass es andersherum wäre! So sind wir manchen zu voreilig. Vielleicht haben sie den Eindruck, dass Rust als Sprache und Toolchain noch zu jung und unreif sei.

Es hängt also davon ab, wen man fragt. Wir als Projekt glauben, dass wir vorsichtig sein sollten bei dem Code, den wir in den Kernel einbringen. Vor allem braucht es Zeit, damit der Code genug Reviews durchlaufen hat und die Kernel-Betreuer ihn guten Gewissens akzeptieren können. Außerdem sind viele Details noch zu diskutieren und zu entscheiden – und jetzt ist es an der Zeit, das innerhalb des Kernels in Angriff zu nehmen.

Dieser Prozess dauert, aber bereits im Kernel zu sein, bietet uns auch weitere Vorteile. So werden jetzt Unternehmen, andere Projekte, potenzielle Nutzerinnen und Nutzer sowie Entwicklerinnen und Entwickler anfangen, sich Rust im Kernel näher anzusehen – und das wird die Unterstützung verbessern, die wir erhalten. Tatsächlich spüren wir genau das bereits jetzt!

Was Herausforderungen betrifft – da mussten wir uns natürlich einigen stellen. Eine wichtige war, dass wir beweisen mussten, dass es tatsächlich möglich ist, Abstraktionen mit sicherem Rust-Code rund um existierende Kernel-Funktionen zu erstellen, also ohne alles in Rust neu schreiben zu müssen und ohne die Performance zu beeinträchtigen. Das alles zu koordinieren war für mich persönlich eine Herausforderung, die ich aber auch genossen habe – dank der Leute, die am Projekt arbeiten, und der Firmen, die ebenfalls einen Beitrag leisten.

Moritz Förster*: Sobald es mit dem Rust-Support in Linux wirklich losgeht: Welche Vorteile winken Kernel-Entwicklerinnen und -Entwicklern und Linux-Nutzerinnen und -Nutzern?*
Miguel Ojeda: Für Endanwender und Unternehmen gibt es einen klaren Vorteil: die Zahl der Schwachstellen zu reduzieren. Mehrere Studien großer Firmen und Software-Projekte zeigen, dass etwa 70 Prozent der Sicherheitslücken auf Speichersicherheitsprobleme der C/C++-Codebasis zurückgehen. Wenn wir also den Großteil dieser Probleme eliminieren und den Schweregrad der Schwachstellen reduzieren können, bedeutet das hoffentlich eine verbesserte Security für Endnutzer und weniger dringende Kernel-Updates, die sie einspielen müssen, sowie weniger Sicherheitslücken für Unternehmen.

Für Kernel-Entwicklerinnen und -Entwickler bietet Rust als Sprache, Bibliothek und Tooling viele moderne Verbesserungen im Vergleich zu C. Das ist nicht überraschend, da Rust auf Basis der Erfahrungen aus der Arbeit mit älteren Programmiersprachen sowie neuesten, wissenschaftlichen Erkenntnissen gestaltet wurde. Unserer Meinung nach wird sich das in insgesamt weniger Logik-Bugs im Kernel niederschlagen, wird aber auch rascheres Debugging und mehr Komfort beim Coding nach sich ziehen. Und all das bedeutet wiederum eine höhere Produktivität für Firmen, die Entwicklerinnen und Entwickler für den Kernel beschäftigen, sowie zuverlässigere Treiber für Endanwender. Das unterstreichen auch die Erfahrungen der Entwicklerinnen und Entwickler der ersten Rust-Treiber.

Idealerweise können sich Kernel-Betreuer beim Akzeptieren von Patches und Refactoring-Code in ihre Subsysteme sicherer sein – aus denselben Gründen. Falls zum Beispiel ein Patch einen sicheren Rust-Treiber ändert, sollte es auf diese Weise nicht möglich sein, Speicherzugriffsfehler in den Kernel einzubringen. Hier steht mehr auf dem Spiel als im Userspace. Wir glauben, dass diese zusätzliche Schutzschicht in Rust für den Code sehr wertvoll ist.

Zu guter Letzt hoffen wir, dass sich neue und jüngere Generationen mit der Kernel-Entwicklung beschäftigen wird – dank der angesagten Sprache und ihrer Techniken. Im Idealfall wird all das Linux dabei helfen, auch in den kommenden Jahrzehnten der am weitesten eingesetzte Kernel der Welt zu bleiben. (sih)

Im Interview: Miguel Ojeda

Miguel Ojeda ist Softwareingenieur und Betreuer des Rust-für-Linux-Projekts. Ferner ist er Teil des ISO-C-Komitees und interessiert sich für undefiniertes Verhalten und Themen der Speichersicherheit.

Quellen

Die englischsprachige Originalversion des Interviews findet sich unter ix.de/zn9z.

> Agile Praxis

Robert Ruzitschka

Interne Developerplattformen entbinden Entwickler und Entwicklerinnen von operativen Aufgaben.

Immer mehr Unternehmen investieren in Developerplattformen. Denn in der agilen Softwareentwicklung steht die Autonomie der Teams und ihre Verantwortung für den gesamten Softwarelebenszyklus ganz oben auf der Liste der Anforderungen, ganz im Sinne von „You build it – You run it“, der Prämisse von Werner Vogels, CTO von AWS. Die Verantwortung ist die Basis für eine Teamkultur, bei der enge Zusammenarbeit in einem multifunktionalen Team unerlässlich ist.

Aber auch für hochprofessionelle und gut ausgebildete Teams ist die Bewältigung aller Aufgaben von Design, Softwareentwicklung und Test bis zum Management der Entwicklungswerkzeuge und der Infrastruktur eine große Herausforderung. Eine Plattform, die den Teams Teile dieser Aufgaben abnimmt, kann Kapazität für die eigentliche Aufgabe der Teams freimachen: Kundennutzen zu erzeugen. Die Implementierung einer solchen Plattform ist allerdings nicht ohne Tücken und kann bei schlechter Ausführung zu Problemen führen.

Aus der Psychologie ist bekannt, dass es eine Grenze für die Anzahl der unterschiedlichen Tätigkeiten gibt, die ein Mensch und damit auch ein Team gut erledigen kann: die maximale kognitive Kapazität. Überschreitet die kognitive Last diese Schwelle, ist keine qualitativ hochwertige Arbeit mehr möglich.

Developerplattformen können die kognitive Last für die Teams reduzieren, indem sie wichtige Aspekte von Entwicklung und Betrieb zentralisieren: Von zentral gemanagten Coderepositorys über eine zentralisierte Pipeline bis zu kompletten Applikationslaufzeitumgebungen mit der entsprechenden Instrumentierung ist alles möglich. Allen Implementierungen von Developerplattformen ist gemeinsam, dass die Verantwortung für den Betrieb der Applikation beim Produktteam bleibt.

Zentralisierte Applikationsumgebungen und Entwicklungswerkzeuge sind nichts Neues. Allerdings haben moderne Developerplattformen Eigenschaften, die sie für die agile Arbeitsweise anwendbar machen. Teamautonomie ist in der agilen Welt ein hoher Wert: Die Plattform anzuwenden bleibt für die Teams in den meisten Fällen freiwillig. Ziel muss es sein, in der Developerplattform Funktionen anzubieten, die die kognitive Last der Produktteams reduzieren und gut angenommen werden.

Die Plattform als internes Produkt

Hilfreich ist es, das zentralisierte Tooling als Produkt aufzufassen und entsprechend zu organisieren. Die Kunden dieses Produkts sind die internen Entwicklungsteams. Notwendig ist, dass es für die Developerplattform eine ausreichend große Mannschaft gibt, die sicherstellt, dass die Plattform kontinuierlich weiterentwickelt wird und die nötige Performance verlässlich zur Verfügung steht. Dem Produktgedanken entsprechend gilt es, auch Dokumentation und Support sicherzustellen. Die zentrale Plattform darf nicht zum Flaschenhals werden und die Produktteams in ihrer Arbeit behindern. Durch einen konsequenten Schwerpunkt auf Self-Service beim Bereitstellen der Funktionen lässt sich das vermeiden. Plattformen sollten so schlank wie möglich sein und das Plattformteam muss vermeiden, dass immer mehr Funktionen in das zentrale System wandern – auch hier ist der Produktansatz hilfreich. Implementiert wird nur, was die Kunden wirklich benötigen. Die Erfahrung zeigt, dass Produktteams gut implementierte Plattformen akzeptieren. Eine vereinheitlichte Arbeitsweise über Produktteams hinweg ist vor allem in größeren Unternehmen erwünscht, da sie die Zusammenarbeit erleichtert.

Zentralisierte Entwicklungswerkzeuge und Laufzeitumgebungen sind einfach erweiterbar, um allen Entwicklerinnen und Entwicklern benötigte Daten und Metriken zur Verfügung zu stellen. In vielen regulierten Umgebungen wie in der Finanzindustrie oder der Autobranche sind die Complianceanforderungen für die Entwicklungsteams hoch. Können die zentralen Plattformen hier Aufgaben automatisiert abnehmen, erhöht sich ihre Attraktivität für die Produktteams noch weiter.

Developerplattformen können ein machtvolles Werkzeug sein, das Produktteams unterstützt. Ein konsequenter Produktansatz vermeidet die Probleme traditioneller zentralisierter Entwicklungs- und Laufzeitumgebungen. Weiterhin gilt das Motto „You build it – You run it“, aber das „You run it“ bezieht sich nicht mehr auf den gesamten Softwarestack, sondern nur auf einen Teil davon. Die wichtige Rückkopplungsschleife bleibt geschlossen. (nb)

Robert Ruzitschka

ist DevOps Community Lead und Engineering Coach bei der Raiffeisen Bank International und beschäftigt sich mit Aspekten agiler Softwareentwicklung.

SEI TEIL DER FÜHRENDEN VERANSTALTUNG FÜR ENTWICKLER:INNEN

10.000+
TEILNEHMER:INNEN

300+
SPEAKER:INNEN

2
TAGE

// Jetzt exklusiven -20% Discount sichern: heiseberlin2023

World Congress 2023

worldcongress.dev

27.–28. JULI 2023 BERLIN

Chancen erkennen. Herausforderungen annehmen.

Softwareentwicklung bei Weidmüller

Mit Innovationen rund um Smart Industrial Connectivity und das Industrial Internet of Things leistet Weidmüller einen wesentlichen Beitrag zur digitalen Transformation und einer lebenswerten und nachhaltigen Zukunft.

Bewerben Sie sich jetzt und werden Sie Teil unseres Teams.

Mehr erfahren auf:
https://weidmueller.de/karriere-entwicklung

Weidmüller