Engineer

Automatisierung und CI/CD

82 Leistungen

Die Teile Ihres Systems verbinden

Die unter „Automatisierung und CI/CD“ gesammelten Arbeiten zeigen einen Teil eines funktionierenden Systems. Wir können helfen, diesen Teil mit dem Rest Ihres Betriebs zu verbinden. Zahlen kommen termingerecht dort an, wo sie gebraucht werden, ohne dass am Monatsende jemand eine Tabelle kopieren muss.

Unsere Leistungen entdecken Kontakt aufnehmen →

Eine vergleichbare Herausforderung finden

Folgen Sie einem Beispiel, das Ihrer Situation ähnelt. Die Rückschau beschreibt die Entscheidungen hinter dem Ergebnis und geht über die bloße Technologiebezeichnung hinaus. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

Ein Infrastrukturplan für 14 Abteilungen Kontakt aufnehmen →

30 Tage Stromdaten in 180 ms

Varianten zur stündlichen Aggregation des Stromverbrauchs in Python, SQL und Bash + Jq entwickelt; der Python-Benchmark verarbeitete einen 30-Tage-JSONL-Datensatz in 180 ms.

1.000 Fehler in geografischen Daten behoben

1.000 Probleme in geografischen Daten und Zeitreihendaten mit GDAL, ArcGIS, PostGIS, Mapbox, QGIS, SQL (PL/pgSQL, Transact-SQL) und Bash behoben und eine hochwertige Big-Data-Verarbeitung sichergestellt.

6 ETL/ELT-Pipelines entworfen und betrieben

6 ETL/ELT-Pipelines mit Google BigQuery, MSSQL, PostgreSQL, Shell-Skripten, PL/pgSQL und Transact-SQL entworfen, umgesetzt und administriert sowie Daten für eine effiziente Verarbeitung durch die Python-API integriert.

Das erste Observability-Dashboard aufgebaut

Das erste Observability-Dashboard des Unternehmens entwickelt und eingeführt, das Systemperformance und Datenvisualisierungen in Echtzeit auf dem großen Bürofernseher anzeigt.

10 PostgreSQL-/MS-SQL-Server auf Ubuntu

10 PostgreSQL- und MS-SQL-Server auf Ubuntu-Linux-VPS entworfen, bereitgestellt und gewartet und dabei Serverperformance und Zuverlässigkeit sichergestellt.

1.000 RBAC-Regeln für sichere Linux-Server

Die Datensicherheit durch 1.000 RBAC-Regeln für Entwickler, Anwendungsinstanzen, PostgreSQL, MS SQL und weitere Linux-Server erhöht, unberechtigte Zugriffe verhindert und die Regeln mit Ansible-Automatisierung dokumentiert.

GIS-SaaS-Deployments mit GitHub automatisiert

Bereitstellung, Datenverarbeitung und Berichtswesen einer GIS-SaaS-Anwendung mit GitHub Actions CI/CD, Python, Bash und SQL automatisiert.

20 GIS-Datenpipelines automatisiert ausgeliefert

Die Bereitstellung von 20 GIS-Datenpipelines und ETL-Prozessen für Anwendungsdaten automatisiert und dadurch Infrastrukturautomatisierung und Berichterstattung vereinfacht.

100 wichtige Sicherungen mit Barman automatisiert

100 geschäftskritische Datensicherungen mit Barman, Google Cloud, Bash und Python automatisiert und die Datenintegrität über die Datenbanken hinweg abgesichert.

20 Docker-Apps auf Google Cloud und AWS betrieben

20 in Docker containerisierte Anwendungen bereitgestellt und gewartet, Fehler mit Podman und Kubernetes behoben und R-basierte Anwendungen auf Google Cloud und AWS verwaltet.

Zugriffsverwaltung mit M365 und SSO geleitet

Durch die Leitung von Vorhaben zur Zugriffsverwaltung mit M365, 1Password, Red Hat SSO und OKTA SSO Sicherheitsverletzungen verhindert.

Betriebsrisiken mit Grafana/Prometheus gesenkt

Betriebsrisiken durch ein Monitoring-Dashboard mit Grafana und Prometheus verringert und die Systemzuverlässigkeit verbessert.

Teamkommunikation und Zusammenarbeit verbessert

Teamkommunikation und Zusammenarbeit durch die Einführung von Slack, Mattermost, 1Password und Jira verbessert und 8.000 Arbeitsstunden eingespart.

Abläufe überarbeitet und 8.000 Stunden eingespart

Interne Abläufe überarbeitet und durch bessere Softwarearchitektur, Systeme und Terminplanung 8.000 Stunden eingespart.

Netzwerkinfrastruktur administriert

Die Netzwerkinfrastruktur für über 1.000 Server administriert und eine geeignete Systembereitstellung, Sicherheit und Fehlerdiagnose sichergestellt.

SSL/TLS für 100 Docker-Apps automatisiert

Die Erstellung von SSL/TLS-Zertifikaten für 100 Docker-Anwendungen automatisiert und sichere Verbindungen auf Ubuntu-Linux-Hosts sichergestellt.

CI/CD vereinfacht und 4.000 Stunden gespart

CI/CD-Prozesse verbessert und durch die Automatisierung von Softwareentwicklungspipelines 4.000 Stunden eingespart.

4.000 Stunden durch GitHub-/GitLab-CI/CD gespart

Datenanalyse und Softwareentwicklung durch CI/CD-Praktiken mit GitHub, GitLab, Bash und Python verbessert und 4.000 Stunden eingespart.

Ein Datenanalyse-Berichtssystem entwickelt

Ein Berichtssystem für Datenanalysen entwickelt und den quartalsweisen Softwareumsatz durch Python-basierte PDF-Berichte um 400 % erhöht.

Datenarbeit mit Python und PL/pgSQL automatisiert

Datenverarbeitungsaufgaben mit Shell-Skripten, PL/pgSQL, Python und Transact-SQL automatisiert und Produktivität sowie Effizienz gesteigert.

150 Stunden monatlich in 30 Projekten gespart

Die Projekteffizienz durch optimierte Arbeitsabläufe und Ressourcenplanung verbessert und über 30 Projekte hinweg monatlich 150 Stunden eingespart.

600 PL/pgSQL-ETL/ELT-Pipelines entwickelt

600 PL/pgSQL-basierte ETL/ELT-Pipelines für effizientere komplexe Datenverarbeitung in mehreren PostgreSQL-Entwicklungs- und Produktivumgebungen entwickelt.

Den API-Vertrag aus der Datenbank generiert

Den API-Vertrag von der Datenbank aus nach außen generiert — OpenAPI, einen typisierten TypeScript-Client mit 44.076 Zeilen, 61 Mock-Handler und die von der UI durchgesetzten Grenzen — mit einer Prüfung an jedem Übergang, die Abweichungen zurückweist.

Azure-Infrastruktur als Code provisioniert

Azure-Infrastruktur mit Bicep als Code provisioniert — Container Apps, PostgreSQL Flexible Server, Front Door/WAF und Netzwerke — für Entwicklungs-, Staging- und Produktivumgebung.

GitHub-Actions-CI/CD-Pipelines aufgebaut

GitHub-Actions-CI/CD-Pipelines mit einem Distroless-Frontend-Image für den Produktivbetrieb und der Übernahme von Releases über mehrere Umgebungen entwickelt.

578 go-task-Automatisierungsziele verfasst

578 go-task-Automatisierungsziele für native, Docker- und HTTPS-Entwicklungsmodi, Linting, Tests, Datenbank und Bereitstellung verfasst.

Azure-Plattformbereitstellungen verantwortet

Die durchgängige Bereitstellung der Plattform auf Azure verantwortet und Releases über Entwicklungs-, Staging- und Produktivumgebungen hinweg gesteuert.

Englisch-dänisches i18n-System aufgebaut

Ein englisch-dänisches Internationalisierungssystem mit 12.027 Nachrichten je Sprachversion in 458 Namespace-Dateien aufgebaut, mit per Linter durchgesetztem Wortschatz und einer Grenze von 400 Zeilen je Datei.

Warnungsfreiheit für sechs Sprachen durchgesetzt

Für sechs Sprachen — Go, TypeScript, SQL, Python, Shell und Markdown — einen durch Pre-Commit-Hooks durchgesetzten Qualitätsstandard ohne Warnungen festgelegt.

Zeitmanagement- und Berichtssystem entworfen

Ein Zeitmanagement- und Berichtssystem entworfen, das über Jahre ohne wesentliche Änderungen im Produktivbetrieb blieb.

Teamzusammenarbeit automatisiert

Teamzusammenarbeit, Passwortverwaltung sowie Aufgaben- und Zeitmanagement automatisiert und ein halbautomatisches System zur Projektpräsentation entwickelt, um die Teamproduktivität zu steigern.

Unternehmensweiten Passwortmanager integriert

Ein unternehmensweites Passwortverwaltungssystem integriert, die Sicherheit erhöht und die Zugriffskontrolle vereinfacht.

Testsuite für Go, SQL und Browser aufgebaut

Eine mehrschichtige automatisierte Testsuite aufgebaut — 981 Go-Tests, 543 Frontend- und Browserspezifikationen, 494 verhaltensprüfende SQL-Tests — mit Mutationstests, eigenschaftsbasierten Tests und einer verbindlichen Barrierefreiheitsprüfung.

Repository-Prüfengine mit 268 Checks entwickelt

Die Prüfengine des Repositorys entwickelt — 268 registrierte Commit-Prüfungen, 277 Lint-Regeln und 15 eigene ESLint-Regeln — sowie 146 Tests der Prüfungen selbst, damit der Build den Standard durchsetzt und nicht erst das Review.

Langsame Aufgaben in River-Jobqueue verlagert

Langsame Verarbeitung aus dem Anfragepfad in eine River-Jobqueue verlagert — 15 Worker-Module, 8 geplante Aufgaben und 20 pg_cron-Jobs — sodass Anfragen zurückkehren, während die zugehörige Arbeit weiterläuft.

Kontinuierliche Sicherheitsprüfungen etabliert

Ein kontinuierliches Sicherheitsprogramm aufgebaut — Codeanalyse, DAST, Abhängigkeits- und Schwachstellenprüfungen, SBOM-Erzeugung, Geheimnissuche und per SHA festgelegte Actions — ergänzt durch 21 schriftliche Sicherheitsaudits.

Infrastruktur in einem Repository definiert

Die eigene Unternehmensinfrastruktur als 19 Ansible-Playbooks und 34 Rollen mit 12.065 YAML-Zeilen aufgebaut, die einen laufenden Host mit ausschließlich idempotenten Plays in den deklarierten Sollzustand überführen.

SSH mit 24 geprüften Direktiven gehärtet

SSH auf 24 geprüfte Direktiven gehärtet und dreistufig validiert — Kandidatendatei, zusammengesetzte Konfiguration und anschließend die Rückmeldung des Daemons selbst — nachdem diese gezeigt hatte, dass der laufende Server zwei der vierundzwanzig Direktiven stillschweigend überschrieb.

Angriffssperren bei jedem Lauf nachgewiesen

Den Pfad zur Angriffssperre bei jedem Härtungslauf durchgängig nachgewiesen: eine reservierte Testadresse gesperrt, die entstandene Kernelregel ausgelesen und die Sperre in einem garantierten Aufräumblock aufgehoben, sodass eine ausgefallene Jail den Lauf scheitern lässt, statt einen gesunden Zustand zu melden.

Firewallregeln nach Position geprüft

Firewallregeln nach ihrer Position statt bloß ihrer Existenz geprüft, anhand der nummerierten Regelliste und der aktiven Paketfilterkette, denn eine vorhandene Regel wird nicht zwingend von einem Paket erreicht.

Sicherungsautomatisierung mit klarem Wiederherstellungsweg

restic-Sicherungsautomatisierung mit Aufbewahrung, Integritätsprüfungen und monatlichem Wiederherstellungstest umgesetzt; Speicheraktivierung und Nachweis einer tatsächlichen Wiederherstellung als getrennte Betriebsschritte dokumentiert.

Totmannüberwachung für Hosts entwickelt

Eine Totmannüberwachung entwickelt, die nur bei gesundem Speicher- und Plattenzustand Signale sendet, sodass ein beeinträchtigter Host durch sein Verstummen alarmiert — und sechs Variablennamen mit "free" gefunden, deren Prüfungen korrekt "available" maßen, auf einem Rechner mit 464 MB um eine Größenordnung verschieden.

Ansibles Prüfmodus aussagekräftig gemacht

Den Prüfmodus auf der gesamten Plattform aussagekräftig gemacht, nachdem sechs Sonden anhand eines vom Host nie gelieferten Werts entschieden hatten: Ansibles command-Modul meldet unter --check Erfolg, obwohl es den Befehl vollständig überspringt.

Sekunden-Preflight für Geheimnisse ergänzt

Ein Preflight-Play ergänzt, das denselben Code wie der Konvergenzlauf mit den lokal beim Betreiber liegenden Geheimnissen in etwa einer Sekunde ausführt, nachdem ein teilweise durchgelaufener Produktivlauf bei Aufgabe neun abbrach, während die Swap-Einstellungen bereits auf dem Produktivhost geschrieben waren.

DNS-Zone mit 20 Einträgen als Code abgeglichen

Eine DNS-Zone mit 20 Einträgen deklarativ über die Cloudflare-API abgeglichen, mit separaten Einstiegspunkten für Audit und BIND-Export, und den zuvor eingerichteten CDN-Proxy aus Datenschutzgründen wieder abgeschaltet.

Alle systemd-Units stärker isoliert

Die Angriffsfläche der systemd-Sandboxen für jede von der Plattform installierte Unit verringert — einen Totmannschaltungsdienst von 9,6 UNSAFE auf 1,5, eine internetseitig erreichbare Git-Forge von 8,3 EXPOSED auf 1,5 — und eine Parserprüfung beim Konvergenzlauf ergänzt, nachdem eine falsch geschriebene Direktive in drei Unit-Vorlagen stillschweigend ignoriert worden war.

7 lesende Host-Berichtsrollen entwickelt

Sieben schreibgeschützte Berichtsrollen entwickelt, die den Zustand eines laufenden Hosts in Markdown abbilden — Fakten, Zugriff, Git, Metriken, Verkehr, Sicherheit und Anbieterinventar — nach der Regel, dass keine Zahl erscheint, die der Lauf nicht gemessen hat.

Ihr Code auf Ihrem eigenen Server

Die eigene Git-Forge auf Soft Serve bereitgestellt, standardmäßig privat, ohne Weboberfläche und mit an Loopback gebundenem SSH-Port hinter einem Sprunghost; die vorgeschaltete Startseite als Build-Artefakt der Hauptwebsite statt als manuell gepflegte Kopie umgesetzt.

Container mit Podman und Quadlet betrieben

Eine Containerebene mit Podman und Quadlet unter systemd statt Docker bereitgestellt, da Dockers standardmäßige Portfreigaben im Bridge-Netz UFW-Regeln umgehen können; für Bereitstellungen einen unprivilegierten Nutzer mit einem festen Befehl statt root eingerichtet.

Einen zweiten Server aus Code provisioniert

Die Provisionierung eines zweiten Servers bei einem zweiten Cloud-Anbieter automatisiert und die Firewall vor dem Rechner angelegt, damit er von Anfang an geschützt ist; die Firewalls beider Anbieter direkt über deren REST-APIs gesteuert, um eine Drittanbieter-Collection zu vermeiden.

Zuständigkeit nach Geheimnisleck klar geregelt

Eine Regel zum Zuständigkeitsumfang im Repository festgeschrieben, nachdem eine Umstrukturierung Inventar, Firewallfreigaben und Texte eines anderen Unternehmens hineingetragen hatte; die isolierten Rückstände weiterhin auf Geheimnisse prüfen lassen, statt sie auszuschließen.

Jedes Betriebsgeheimnis je Host getrennt

Die vier betrieblichen Geheimnisse je Host getrennt, nachdem sich gezeigt hatte, dass zwei Hosts mit einer gemeinsamen Totmannschaltung weniger zuverlässig alarmieren als zwei getrennte Schaltungen und eine gemeinsame Sicherungspassphrase die Sicherungen beider Hosts auch in getrennten Repositorys offenlegen kann, wenn jemand die Passphrase erlangt und die verschlüsselten Daten und Schlüsseldateien beider Repositorys lesen kann.

Commit-Prüfung mit 22 Lintern aufgebaut

Eine verbindliche Commit-Prüfung aus 22 einzeiligen Lintern und fünf ausführlicheren Prüfungen aufgebaut, ohne Warnungsstufe und ohne Inline-Unterdrückungen, für HTML, CSS, JavaScript, Python, YAML, Markdown, Shell, Links, Rechtschreibung, Geheimnisse und Typografie.

Ein Sicherheitsnetz, das keine halbe Stunde mehr kostet

Die browsergestützte Qualitätsprüfung der Website von 1.636 auf 615 Sekunden verkürzt, indem die Prüfungen in einem auf vier parallele Ausführungen begrenzten Worker-Pool nach längster Laufzeit zuerst eingeplant werden, nachdem die alphabetische Reihenfolge 320 statt 224 Sekunden benötigt hatte.

27 Schriftgrößen durch 6-stufige Skala ersetzt

27 tatsächlich dargestellte Schriftgrößen mit teils nur 0,6 % Abstand durch eine sechsstufige Skala im Verhältnis einer großen Terz ersetzt und den Linter geschrieben, der eine achtundzwanzigste Größe zurückweist.

Barrieren, die das Prüfwerkzeug übersieht

11 Barrierefreiheitsfehler gefunden und behoben, die der Browser als unbedenklich gemeldet hatte — acht trotz pointer-events weiter per Tab erreichbare Footerlinks, eine Scroll-Timeline, die das Kolophon auf drei Seiten abschnitt, und eine Regelengine, die nur 70 ihrer 105 Regeln ausführte.

Farbsystem auf 28 Farben begrenzt

Das Farbsystem auf 28 dokumentierte Farben begrenzt und einen Linter entwickelt, der verwendete, aber undokumentierte, dokumentierte, aber ungenutzte, falsch gemessene, als Literal neu geschriebene oder einer bestehenden Farbe bis auf einen Wahrnehmungsabstand von 0,02 ähnliche Werte zurückweist.

Fehlende Schicht im hellen Farbschema gefunden

Durch die Prüfung gleicher Schichtanzahlen auf allen mehrschichtigen Oberflächen beider Farbschemata festgestellt, dass dem hellen Farbschema seit seiner Entstehung eine ganzseitige Überlagerungsschicht fehlte.

Website nach Gemini und Gopher gespiegelt

Gemini- und Gopher-Ausgaben aus derselben Website-Quelle entwickelt; die Leistung dokumentiert für diesen Stand 777 Dokumente je Format, wobei durch das Hinzufügen der Formate null HTML-Bytes geändert wurden.

963 Metadatenblöcke durch einen Graphen ersetzt

963 anonyme Blöcke strukturierter Daten, die das Unternehmen 1.671-mal erneut beschrieben, durch einen verknüpften Graphen mit 16 Typen aus stabilen Identifikatoren auf Basis des eigenen Web-Ursprungs ersetzt.

Stylesheet von 87 KB auf 31 KB verkleinert

Das Stylesheet-Bündel durch das Auslagern einer Base64-Schrift von 87 KB auf 31 KB verkleinert und 756 KB in 22 bei jeder Bereitstellung veröffentlichten, aber nirgends referenzierten Dateien entfernt.

Sitemap mit einheitlichem Zeitstempel korrigiert

Eine Sitemap korrigiert, in der 172 von 176 URLs denselben Änderungszeitpunkt trugen, indem das Datum nach der Feststellung aus der Git-Historie gelesen wurde, dass der Export jede Datei bei jedem Lauf neu schreibt.

Dokumentation einem eigenen Linter unterstellt

Die Dokumentation einem eigenen Linter unterstellt — mit nur absenkbarem Zeilenbudget je Datei, Inhaltsverzeichnispflicht und Prüfung echter Pfade — der sechs von neun als erledigt markierte Pläne mit offenen Aufgaben und vierzehn in keinem Dokument genannte verpflichtende Commit-Prüfungen fand.

Qualitätsprüfung mit 10.242 Zeilen selbst geprüft

10.242 Zeilen JavaScript der Qualitätsprüfung unter Formatter und Linter gebracht, nachdem sie sich als größter und zugleich ungeprüfter Codebestand des Repositorys erwiesen hatten; 13 Befunde behoben und keinen unterdrückt.

Datenbankgestützten Lebenslaufgenerator aufgebaut

Einen datenbankgestützten Generator für Lebenslauf, Referenzen, Portfolio und Anschreiben in Python entwickelt — 41 Module, 10.580 Zeilen — der sechs Ausgabeformate aus einer SQLite-Quelle mit 23 Tabellen erzeugt, aufgebaut durch eine idempotente Pipeline mit 19 Schritten.

Generator mit 981 Tests und 92 % Abdeckung

Den Generator mit 981 Testfällen und mindestens 92 % Zweigabdeckung geprüft, Warnungen als Fehler behandelt und Idempotenz durch zwei vollständige Pipeline-Durchläufe aus einer leeren Datei nachgewiesen, wobei der zweite nichts ändern darf.

106 von 106 PDF/UA-1-Profilprüfungen bestanden

Die PDF/UA-1-Validierung für neun Dokumente dokumentiert: 106 von 106 Prüfungen des konfigurierten Barrierefreiheitsprofils bestanden, während die Archivvariante eine von 146 Regeln verfehlte.

Jede Lint-Regel als Fehler aktiviert

Sämtliche Regeln des Python-Linters als Fehler aktiviert und 1.815 Befunde bis auf null abgearbeitet; jede der wenigen Ausnahmen schriftlich begründet und zwei davon durch eine Prüfung statt nur einen Kommentar abgesichert.

Die Prüfwerkzeuge selbst mit Tests geprüft

Tests für die Prüfwerkzeuge selbst geschrieben, nachdem klar war, dass ein nur mit fehlerfreien Eingaben getestetes Werkzeug irgendwann auch nach dem Lesen von nichts Erfolg meldet — einen Tippfehler eingebaut, um die Rechtschreibprüfung zu prüfen, und einen ID-Bereich aus der Datenbank statt einer Testkonstante übernommen.

Abweichende Hooks und Qualitätsprüfung gefunden

Durch den Vergleich beider Listen in einem Test festgestellt, dass Commit-Hooks und Qualitätsprüfung unterschiedliche Prüfungen ausführten, obwohl ein Dokument Gleichheit versprach — die fünf ausschließlich von Hand ausgeführten Prüfungen waren gerade diejenigen für die Lebenslauftexte.

Forge-CI-Hook geprobt und 2 Fehler gefunden

Den CI-Hook auf der Forge probeweise ausgeführt und zwei beim bloßen Lesen unerreichbare Fehler gefunden: einen Fallback mit einem nicht auflösbaren Eingabeargument und eine Git-Umgebungsvariable, die der Qualitätsprüfung in den Checkout folgte und 19 Tests scheitern ließ.

Sprachübergreifende Zahlenprüfung aufgebaut

Eine sprachübergreifende Inhaltsprüfung entwickelt, die bei fehlenden Zahlen gegenüber dem Englischen und bei sprachfremder Notation fehlschlägt — dabei zwei dänische Beschreibungen ohne eine Kennzahl und sechzehn ukrainische Textstellen mit englischen Anführungszeichen gefunden.

Parser zur Prüfung aller FFI-Aufrufe geschrieben

Einen Parser geschrieben, der den echten C-Header mit 7.308 Zeilen liest und jede Aufrufstelle, jede Enum-Konstante sowie final für jede pointerbesitzende Klasse prüft, nachdem ein handgeschriebener Platzhalterheader Aufrufe dreier entfernter Funktionen kompilieren und linken ließ, die zur Laufzeit abstürzten.

Testsuite auf 8,9 Sekunden beschleunigt

Die Testsuite von sechs Tests jenseits eines Limits von sechzig Sekunden auf 135 erfolgreiche Tests in 8,9 Sekunden gebracht, indem der Hauptthread profiliert und die zwei Aufrufe entfernt wurden, in denen er bei 3.989 von 4.017 Stichproben festhing.

Alle 17 Symbole für Screenreader benannt

Jedes ausschließlich durch ein Symbol dargestellte Bedienelement für Screenreader benannt, nachdem die Sendeschaltfläche als "Pfeil nach oben im Kreis, Schaltfläche" angekündigt worden war, und einen Linter geschrieben, der die Beschriftung innerhalb von acht Zeilen um das Symbol verlangt.

macOS-Designsystem mit 48 Tokens aufgebaut

Ein Designsystem mit 48 Tokens auf dem plattformeigenen Glasmaterial aufgebaut und einen Linter geschrieben, der unbenannte Zahlenwerte, fest codierte Schriftgrößen und Animationen zurückweist, die die Einstellung für reduzierte Bewegung ignorieren.

Die ungenutzte Hälfte des Kerns erschlossen

Die zuvor ungenutzte Hälfte des Messaging-Kerns erschlossen — Sicherungsübertragung, verschwindende Nachrichten, Nachrichtenbearbeitung und erneutes Senden, verifizierte Einladungen, Proxys und Verschlüsselungsrichtlinien — und jeden Test ohne Mocks gegen die echte Bibliothek ausgeführt.

Die Lizenz aller ausgelieferten Komponenten kennen

644 Rust-Crates bei jedem Build auf Lizenzverträglichkeit geprüft und die Wirksamkeit durch das Ändern einer Crate-Lizenz sowie das Verschieben der festgelegten Kernrevision ohne Neuerzeugung nachgewiesen.

Visuelle Identität aus 2 Zeichnungen entwickelt

Die Symbol- und Favicon-Sätze des Unternehmens aus zwei handgefertigten Zeichnungen aufgebaut, alle abgeleiteten Assets per Skript neu erzeugt und eine Prüfung ergänzt, die vor der Quelle committete Ableitungen zurückweist.

4 Repositorys auf einen Historienstandard geeint

Vier Repositorys auf einen einheitlichen Historienstandard verpflichtet — Conventional Commits, ohne Emojis und Attribution-Trailer, durch einen Commit-Message-Hook erzwungen — ergänzt durch 48 Anweisungsdokumente zur Arbeitsweise.

13 Adressen ohne Tracking gemessen

Dreizehn Adressen ausschließlich anhand des eigenen Webserver-Zugriffsprotokolls gemessen — ohne Cookie, Tracker oder Drittanbieterdienst — das Clientnetz beim Schreiben jeder Zeile maskiert und das Ergebnis dauerhaft mit 21 täglichen Zählwerten und 20 monatlichen Facetten archiviert.

Sechs Sammler weisen übersprungene Eingaben aus

Sechs Berichtssammler um Angaben zu ungelesenen Eingaben ergänzt — ungeöffnete Dateien, nicht installierte Sonden, unbekannte Journalzeilen — und Schwellenwerte anhand von 116 gemessenen Stichproben bestimmt, nachdem eine blinde Erfassung und ein gesunder Host dieselbe Seite ergeben hatten.

Die technische Arbeit gezielt einsetzen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Das Thema in Ihren Kontext einordnen

Erzählen Sie uns, welche Rolle „Automatisierung und CI/CD“ in Ihrem eigenen System spielt. Wir können helfen, die passende Leistung auszuwählen und die damit verbundene Arbeit zu definieren.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/categories/automation/