Engineer

DevOps

44 Leistungen

Die Teile Ihres Systems verbinden

Die unter „DevOps“ gesammelten Arbeiten zeigen einen Teil eines funktionierenden Systems. Wir können helfen, diesen Teil mit dem Rest Ihres Betriebs zu verbinden. Monitoring und Alarme melden sich, solange ein Fehler noch klein ist; Kennzahlen zeigen, wie oft das gelingt.

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. Das erste Observability-Dashboard aufgebaut — der erste Einblick des Unternehmens in die eigenen Systeme

Das erste Observability-Dashboard aufgebaut Kontakt aufnehmen →

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.

30 Ubuntu-Linux-VPS-Instanzen verwaltet

30 Ubuntu-Linux-VPS-Instanzen verwaltet, Strategien zur Notfallwiederherstellung umgesetzt und geeignete Netzwerkkonfigurationen sichergestellt.

Betriebsrisiken mit Grafana/Prometheus gesenkt

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

GIS-Kartenprodukt zu 10-fachem Umsatz geführt

Die Softwareentwicklung einer GIS-Kartenanwendung geleitet, den Umsatz um das 10-Fache gesteigert und das Produkt als zentralen Datenbestand positioniert.

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.

Infrastruktur vereinfacht und AWS verlassen

Die Infrastruktur eines internationalen Kunden überarbeitet, ungenutzte Dienste entfernt, Kapazitäten angepasst und Workloads von AWS weg verlagert. Die Gründerin berichtet von einer Senkung der Cloudkosten um den Faktor 10 ohne Produktivitätsverlust.

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.

Sicherungen und Notfallwiederherstellung als Code

Die Aufbewahrung von Datenbanksicherungen als Infrastructure as Code konfiguriert, anschließend die Wiederherstellbarkeit geprüft und das Wiederherstellungsverfahren dokumentiert — mit ausdrücklich benannten Lücken, statt sie erst im Ernstfall entdecken zu lassen.

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.

Technische Grundlagen der Plattform festgelegt

Die grundlegenden Plattformentscheidungen in den Wochen nach der Eröffnung des Repositorys im November 2025 festgelegt — Schichtenarchitektur, datenbankzentrierter Datenzugriff und Warnungsfreiheit — die neun Monate später weiterhin gelten.

Die gesamte Kerninfrastruktur neu aufgebaut

Als eine der ersten Mitarbeiterinnen die gesamte Kerninfrastruktur und die unterstützenden Prozesse eines Green-Energy-SaaS-Start-ups von Grund auf entworfen und aufgebaut und damit die Grundlage für schnelles Wachstum gelegt.

Teamzusammenarbeit automatisiert

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

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.

Eigene Fehlerüberwachung und Tracing entwickelt

Eigene Fehlerüberwachung und OpenTelemetry-Tracing entwickelt statt eingekauft — Bereinigung von Nutzdaten, Erkennung von Fehlerspitzen und Regressionen, Symbolauflösung und synthetisches Lebenszeichen — mit 11 Betriebsansichten.

1.022 Migrationen mit Schemaabweichungsprüfung

Die Konsistenz des Schemas über 1.022 Migrationen mit einer CI-Prüfung abgesichert, die die Datenbank auf zwei Wegen aufbaut — Neuinstallation sowie Installation mit sämtlichen Migrationen — und bei Unterschieden fehlschlägt.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Die technische Arbeit gezielt einsetzen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Das Thema in Ihren Kontext einordnen

Erzählen Sie uns, welche Rolle „DevOps“ 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/devops/