Engineer プラットフォーム

実績紹介

エンジニアリングサービスと、その実績をご紹介します。

Engineer ApSは2025年に登記されました。掲載している仕事の大半は、それ以前に創業者がお客様のチームの一員として実施したものです。会社として実施した仕事も、その後の雇用先で実施した仕事も含まれます。同じ担当者が同じ基準で取り組んでいますが、それぞれの立場を曖昧にせず明示しています。

生かす価値のある経験を探す

この記録から、プロジェクトに関連する経験をお探しください。

サービスを見る お問い合わせ →

記録の背景にある判断を読む

各項目は、さまざまなサービスにわたる実施済みの仕事を説明しています。振り返りを読むと、その背景にある状況と判断が分かります。

事例 お問い合わせ →

カテゴリーから探す

ソフトウェアエンジニアリング

バックエンド開発 / フロントエンド開発 / フルスタック開発 / API・システム連携 / Web開発 / テスト・品質保証 / 性能改善 / Python / 国際化

データ・データベース

データベース / PostgreSQL / SQL / データエンジニアリング / データパイプライン（ETL/ELT） / データ分析 / データガバナンス / GIS・地理空間情報

クラウド・インフラ・DEVOPS

クラウド / コンテナ（Docker/Kubernetes） / DevOps

自動化・CI/CD インフラ Linux・サーバー
システム管理 ネットワーク・VPN 監視・オブザーバビリティ
信頼性・バックアップ セキュリティ
プラットフォーム設計 移行・モダナイゼーション

コンサルティング・リーダーシップ

技術面の統率 チームの統率 アジャイル・Scrum プロジェクト管理 プロダクト・要件
関係者との調整・報告 文書作成 ソリューション設計 ブランド・マーケティング

以下の仕事はすべて **実施済みです** — 提案や計画ではありません。

各項目には、その仕事で何が変わったかを記載し、関連するサービスへのリンクを設けています。

提供するサービス 料金 →

{ クラウド・インフラ・DevOps }

クラウドインフラ・移行

ソフトウェアを動かす場所を整える

実際の使用量に合わせてサーバーを構成し、月額費用を事前に把握したうえで、クラウドへの移行もクラウドからの移転も進めます。

2台目のサーバーをコードで構築

別のクラウド事業者で2台目のサーバーの構築を自動化。先にファイアウォールを作成し、起動時から保護されるようにし、両事業者のファイアウォールをREST APIから直接操作して、第三者のcollectionへの依存を避けた。

PodmanとQuadletでコンテナを運用

Dockerの既定のブリッジネットワークでのポート公開がUFWルールを迂回する場合があるため、systemd配下のPodman・Quadletでコンテナ基盤を構築。デプロイにはrootではなく、固定コマンド1つを使う非特権ユーザーを用いた。

小型サーバー上の公開サービス

利用可能メモリー464 MBを設計制約として、512 MB・1コアのホストで自社の公開サービスを運用。gitホスティング、7ドメインのWebサーバー、Tor、2つの代替プロトコルサーバー、不正接続元の遮断を備え、バックアップ自動化は別の実装として扱った。

リポジトリで定義するインフラ

12,065行のYAMLにわたる19本のAnsible プレイブックと34個のロールで自社インフラを構築し、すべてのプレイを冪等にして、稼働中のホストを宣言した状態へ収束させた。

基幹インフラ全体を一から構築

グリーンエネルギー分野のSaaSスタートアップで初期に採用された社員の1人として、基幹インフラ全体と運用工程を一から設計・構築し、急成長の基盤を整えた。

Azureへのプラットフォーム導入を一貫して担当

プラットフォームのAzureへのデプロイを最初から最後まで担当し、開発・ステージング・本番環境のリリースを管理した。

Azureインフラをコードとして構築

Bicepで、開発・ステージング・本番の各環境に、Container Apps、PostgreSQL Flexible Server、Front Door/WAF、ネットワークを含むAzureインフラをコードとして構築した。

AWSの外へ移し、インフラを整理

海外顧客のインフラを見直し、未使用サービスを廃止して容量を調整し、ワークロードをAWSから移した。創業者の報告では、生産性を損なうことなくクラウド費用を10分の1に削減した。

Google Cloud・AWSでDockerアプリ20個を運用

Dockerでコンテナ化した20個のアプリケーションをデプロイ・保守。PodmanとKubernetesで問題を調査し、Google Cloud・AWS上のRアプリケーションも管理した。

{ クラウド・インフラ・DevOps }

コードによるインフラ管理

バージョン管理された設定から再構築できるサーバー

構成全体をファイルに記述し、特定の人の記憶だけに
頼らず、まったく同じ環境を再現できるようにします。

読みやすいTor onionミラーを公開

engineerで始まる読みやすいTor onionアドレスでサイトのミラーを公開。鍵はホスト外で生成し、このリポジトリにもノートパソコンにも置かず、4つの入口のうち計測しない唯一の入口にした。

gitホスティングCIを試し2不具合を発見

gitホスティング側のCIフックを予行実行し、ファイルを読むだけでは到達できない2件の不具合を発見。解決不能な引数を入力へ渡すフォールバックと、検査先に引き継がれ19テストを失敗させるgit自身の環境変数を修正した。

すべての運用秘密情報をホストごとに分離

2ホストで1つのデッドマンズスイッチを共有すると2つのスイッチより監視能力が下がり、バックアップのパスフレーズ共有では、リポジトリが別々でも、そのパスフレーズを入手し、両リポジトリの暗号化データと鍵ファイルを読める相手が2ホストのバックアップを復号できることを確認し、ホストごとの運用秘密情報4つを分離した。

自社サーバーでコードを管理

Web管理画面を持たず、初期状態で非公開のSoft Serveによる自社gitホスティングを導入。SSHポートを踏み台ホストの背後のループバックに限定し、入口のページを手作業の複製ではなく主サイトのビルド成果物にした。

全ユニットのsystemdサンドボックス露出を低減

この基盤が導入するすべてのユニットのsystemdサンドボックスの露出度を低減。デッドマンズスイッチのサービスを9.6 UNSAFEから1.5へ、インターネット公開のgitホスティングを8.3 EXPOSEDから1.5へ改善し、3つのユニットテンプレートで黙って無視されていた設定名の誤記を受け、収束時の構文検査を加えた。

20レコードのDNSゾーンをコードで照合

20レコードのDNSゾーンをCloudflare APIと宣言的に照合し、監査とBIND形式の書き出しを別の実行入口として提供。CDNプロキシは構築後、プライバシーの観点から再び無効にした。

秘密情報の1秒の事前検査を追加

本番実行が9番目のタスクで停止し、すでにスワップ設定が稼働ホストへ書き込まれた事態を受け、運用担当者のローカルな秘密情報に対し、収束処理と同じコードを約1秒で実行する事前検査プレイを追加した。

Ansibleチェックモードを実態に即した検証に

Ansibleのcommandモジュールが--checkでコマンドを省略しながら成功を返すため、ホストから得ていない値で判断していた6個の検査を修正し、プラットフォーム全体のチェックモードが実態を伝えるようにした。

デッドマンズスイッチによるホスト監視を構築

メモリーとディスクが正常な間だけ通知を送るデッドマンズスイッチ監視を構築し、ホストの悪化を通知の停止で知らせるようにした。また、正しく測っていた「available」を「free」と記す変数名を6件発見。464 MBのホストでは両者に1桁の差があった。

復元方法を明示したバックアップ自動化

resticのバックアップを、保持期間、整合性検査、毎月の復元テストとともに自動化し、ストレージの有効化と実環境での復元検証を別の運用手順として記録した。

SSHの24設定項目を検証・堅牢化

SSHの24設定項目を、候補ファイル、統合済み設定、デーモン自身の読み戻しによる3段階で検証するよう堅牢化。読み戻しによって、稼働中のサーバーが24項目のうち2項目を黙って上書きしていたことを検出した。

バックアップと災害復旧をコードで実装

データベースのバックアップ保持をコードで設定し、復旧可能な状態を監査して復元手順を文書化。残る不足を障害時に初めて知ることがないよう明記した。

Linuxサーバーを守る1,000件のRBACルールを作成

開発者、アプリケーションインスタンス、PostgreSQL、MS SQL、その他のLinuxサーバーに1,000件のRBACルールを実装して不正アクセスを防ぎ、データセキュリティを強化。Ansibleによる自動化として記録した。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

2台目のサーバーをコードで構築 · 小型サーバー上の公開サービス · リポジトリで定義するインフラ · Azureインフラをコードとして構築

{ クラウド・インフラ・DevOps }

DevOps・CI/CD自動化

落ち着いて変更を届ける

変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

4リポジトリで共通の履歴基準を維持

4つのリポジトリに対し、Conventional Commits、絵文字禁止、帰属トレーラー禁止という共通の履歴基準をコミットメッセージフックで適用し、作業方法を定める48件の指示文書とともに運用した。

公開するすべての部品のライセンスを把握

毎回のビルドで644個のRust crateのライセンス互換性を監査し、1つのcrateのライセンスを書き換えた場合と、再生成せずコアの固定リビジョンを変えた場合に検査が働くことを実証した。

未使用だったコアの半分の機能を実装

バックアップ転送、消えるメッセージ、編集・再送、確認済み招待、プロキシ、暗号化方針など、アプリケーションが未使用だったメッセージングコアの半分の機能に対応し、モックなしで実ライブラリに対して全テストを実行した。

テスト群を8.9秒に短縮

メインスレッドを測定し、4,017サンプル中3,989サンプルを費やす2つの呼び出しを除去。6テストで60秒の制限を超えていたテスト群を、135テストが8.9秒で通る状態に改善した。

全FFI呼び出しを検証する構文解析器を作成

手書きの仮Cヘッダーで、削除済みの3関数を呼んでもコンパイル・リンクが通り実行時に落ちた問題を受け、実際の7,308行のCヘッダーを読み、全呼び出し箇所、列挙定数、ポインター所有クラスのfinal指定を検証する構文解析器を作成した。

フックと品質ゲートの不一致を発見

同じ検査を行うという文書の記述に反し、コミットフックと品質ゲートが異なる検査を行っていたことを、一覧を比較するテストで発見。手動でしか動かなかった5個は履歴書の文章を検査するものだった。

検査自体のテストを作成

正常入力しか与えない検査は、何も読まずに正常と報告するようになり得ると確認し、検査自体のテストを作成。誤字を埋めて綴り検査の検出を確かめ、テスト内の固定値ではなくデータベースからID範囲を取得した。

すべての静的検査ルールをエラーに設定

Python静的検査の全ルールをエラーとして有効にし、1,815件の指摘を0件まで解消。少数の例外には理由を書き、うち2件はコメントだけでなく検査によって根拠を守った。

生成器に981テストと92%カバレッジを適用

警告を失敗扱いにし、分岐カバレッジ92%を下限として生成器に981件のテストを適用。空のファイルから全構築処理を2回実行し、2回目に何も変わらないことを要求して冪等性を検証した。

10,242行の品質ゲート自体に静的検査を適用

リポジトリ最大のコード群でありながら検査対象外だった品質ゲートのJavaScript 10,242行に整形・静的検査を適用し、13件の指摘を抑制せずに修正した。

更新日時が共通のサイトマップを修正

エクスポートが毎回すべてのファイルを書き直すことを確認し、git履歴から更新日を取得するよう変更。176件のURL中172件が同じ更新日時だったサイトマップを修正した。

GeminiとGopherへサイトを複製

同じウェブサイトのソースからGemini版とGopher版を生成する仕組みを構築。この実績では、その時点で各形式777文書を記録し、形式の追加によって変更されたHTMLはゼロバイトだった。

30分も待たずに済む品質検査

最長の検査から順に、最大4並列のワーカープールへ割り当て、ブラウザーを使うサイト品質ゲートを1,636秒から615秒に短縮。名前順では224秒に対して320秒を要するという測定を踏まえた。

22静的検査のコミットゲートを構築

HTML、CSS、JavaScript、Python、YAML、Markdown、Shell、リンク、綴り、秘密情報、組版を対象に、1行で説明できる検査22個と詳しい説明が必要な検査5個からなるコミットゲートを構築し、警告扱いや行内抑制を認めない方針にした。

継続的なセキュリティ検査を運用

コード検査、DAST、依存関係・脆弱性検査、SBOM生成、秘密情報検査、SHA固定のActionsと、21件の文書によるセキュリティ監査を組み合わせた継続的なセキュリティ活動を確立した。

1,022件のマイグレーションを差分検査で保護

新規インストールと、インストール後に全マイグレーションを適用する2通りでデータベースを構築し、不一致で失敗するCIゲートを設け、1,022件のマイグレーションにわたるスキーマの整合性を保った。

独自のエラー監視・トレースを構築

ペイロードの無害化、急増・再発の検知、シンボル解決、擬似ハートビートを備える独自のエラー監視とOpenTelemetryトレースを外部購入せず構築し、11画面の運用ビューから扱えるようにした。

268検査を持つリポジトリの検査基盤を構築

登録済みコミット検査268件、静的検査ルール277件、独自ESLintルール15件からなるリポジトリの検査基盤と、検査自体を試す146件のテストを構築し、人のレビューだけに頼らずビルドで基準を守るようにした。

Go・SQL・ブラウザーのテスト群を構築

981件のGoテスト、543件のフロントエンド・ブラウザーテスト、494件のSQL動作テストを持つ多層の自動テスト群を構築し、ミューテーションテスト、プロパティベーステスト、アクセシビリティの品質ゲートを加えた。

チームの協働を自動化

チームの協働、パスワード管理、タスク・時間管理を自動化し、プロジェクト実績を半自動で紹介する仕組みも構築して、チームの生産性を高めた。

6言語に警告ゼロの基準を設定

Go、TypeScript、SQL、Python、Shell、Markdownの6言語に警告ゼロの品質基準を設け、pre-commitフックで適用した。

go-taskの自動化ターゲット578個を作成

ネイティブ、Docker、HTTPSの開発モード、静的検査、テスト、データベース、デプロイをカバーする、go-taskの自動化ターゲット578個を作成した。

GitHub Actions CI/CDパイプラインを構築

本番フロントエンド用のdistrolessイメージと複数環境間の昇格を備えた、GitHub ActionsのCI/CDパイプラインを構築した。

データベースを起点にAPI契約を生成

データベースを起点に、OpenAPI、44,076行の型付きTypeScriptクライアント、61個のモックハンドラー、UIが適用する制限を生成し、各段階に不一致で失敗する検査を設けた。

PythonとPL/pgSQLでデータ処理を自動化

Shellスクリプト、PL/pgSQL、Python、Transact-SQLでデータ処理を自動化し、生産性と効率を向上させた。

GitHub・GitLab CI/CDで4,000時間を削減

GitHub、GitLab、Bash、PythonによるCI/CDを導入し、データ分析とソフトウェア開発の工程を効率化して4,000時間を削減した。

CI/CDの効率化で4,000時間を削減

ソフトウェア開発パイプラインを自動化してCI/CD工程を効率化し、4,000時間を削減した。

Dockerアプリ100個のSSL/TLS証明書を自動化

Dockerアプリケーション100個のSSL/TLS証明書の作成を自動化し、Ubuntu Linuxホスト上での安全な接続を確保した。

業務工程の刷新で8,000時間を削減

社内工程を見直し、ソフトウェア構成、システム、日程管理の効率化によって8,000時間を削減した。

GrafanaとPrometheusで運用リスクを低減

GrafanaとPrometheusの監視ダッシュボードを導入して運用リスクを低減し、システムの信頼性を高めた。

Barmanで重要データのバックアップ100件を自動化

Barman、Google Cloud、Bash、Pythonで重要データのバックアップ100件を自動化し、データベース間のデータ整合性を確保した。

GISデータパイプライン20本の提供を自動化

20本のGISデータパイプラインとアプリケーションデータのETL処理の提供を自動化し、インフラ自動化とレポート作成を効率化した。

GitHub ActionsでGIS SaaSのデプロイを自動化

GitHub Actions CI/CD、Python、Bash、SQLで、GIS SaaSアプリケーションのデプロイ、データ処理、レポート作成を自動化した。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

gitホスティングCIを試し2不具合を発見 · PodmanとQuadletでコンテナを運用 · 自社サーバーでコードを管理 · 秘密情報の1秒の事前検査を追加 · Ansibleチェックモードを実態に即した検証に · リポジトリで定義するインフラ · 基幹インフラ全体を一から構築 · Azureへのプラットフォーム導入を一貫して担当 · AWSの外へ移し、インフラを整理 · Google Cloud・AWSでDockerアプリ20個を運用

{ クラウド・インフラ・DevOps }

コンテナ化・オーケストレーション

どこでも同じように動くソフトウェア

ノートパソコン、テストサーバー、本番環境で同じ動作になるようにまとめ、忙しい週には処理能力を増やせるようにします。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

PodmanとQuadletでコンテナを運用 · 全ユニットのsystemdサンドボックス露出を低減 · GitHub Actions CI/CDパイプラインを構築 · Dockerアプリ100個のSSL/TLS証明書を自動化 · Google Cloud・AWSでDockerアプリ20個を運用

{ クラウド・インフラ・DevOps }

サイト信頼性・監視

お客様より先に異変に気づく

障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

6収集処理が省略した入力を明示

6つのレポート収集処理が、未読ファイル、未導入の検査、認識できなかったjournal行など、読み取れなかった入力範囲を明示するよう変更。収集不能と正常なホストが同じページになると確認し、116件の実測サンプルから閾値を定めた。

追跡せずに13アドレスを測定

Cookie、トラッカー、第三者サービスを使わず、Webサーバーのアクセスログのみから13アドレスを測定。書き込み時に接続元ネットワークをマスクし、21の日別集計と20の月別分析軸からなる永続アーカイブへまとめた。

macOSアプリのイベントパイプラインに上限を設定

毎秒41 MBの増加と、36 GB端末で111 GBの圧縮ページを記録したmacOSの障害を受け、イベントバッファとログ出力に上限を設定。記録された全テスト実行では、コアのログ行が610,996から1,411へ減った。

読み取り専用のホストレポートロールを7個構築

事実、アクセス、git、指標、トラフィック、セキュリティ、プロバイダー資産を、稼働ホストからMarkdownにまとめる7個の読み取り専用レポートロールを構築し、実行で測定していない数値は出力しない規則を設けた。

実行ごとに侵入元の遮断を実証

堅牢化の実行ごとに、予約済みテストアドレスを遮断し、生成されたカーネルルールを読み、必ず実行する後処理で遮断を解除する一連の検証を設け、jailの故障が正常報告ではなく実行失敗になるようにした。

3つあったが、どれも機能していなかった防御

一度も機能していなかったSSH総当たり攻撃対策を3件発見・修正。デーモンが1986番で待ち受けるのに22番を監視する遮断jail、上位の広いルールに隠されたレート制限、バイナリを解決できず遮断を一度も実行できなかったアクションを直した。

異常時拒否のレート制限・不正利用対策を構築

Redisによる22個のレートリミッター、Cloudflare Turnstile、リクエストの冪等性、オリジン制限を備え、異常時は拒否する不正利用対策を構築し、呼び出し元を無条件に信頼せずボットや大量アクセスを排除した。

遅い処理をRiverジョブキューへ移行

時間のかかる処理をリクエスト経路からRiverジョブキューへ移し、15個のワーカーモジュール、8件の定期タスク、20件のpg_cronジョブにより、背後の作業が続く間も応答を返せるようにした。

数年後も稼働するインフラを構築

社内外のシステム向けに新たなインフラ機能を計画・実装し、数年後も最小限の変更で稼働し続ける仕組みを構築した。

IPTV配信信号の連続稼働を維持

配信ネットワークとIP電話を昼夜問わず監視・保守し、供給元と顧客の間でIPTV信号の途切れない配信を維持した。

約1,000台のIPTV/OTTサーバーを24/7で支援

中国、米国、ドイツの世界各地の顧客に向け、IPTV/OTT配信プラットフォームのインフラを24/7体制で支援し、約1,000台のサーバーと顧客所有のシステムを管理した。

切り替えと配信ログを備えるメール機能を構築

メール送信をプラットフォームの共通機能として構築。3つのプロバイダーによるフェイルオーバー、配信Webhook、送信・配信ログ、テンプレート、キャンペーンを備え、送信手段がなければ起動を拒否する検査を設けた。

地図製品を最初から運用まで支える

2年間にわたり、インフラ、データ処理、地図アプリケーションを設計・開発・支援し、ソフトウェアの提供と運用保守の両方を担当した。

Apache・NginxでWebサイト40件を管理

Apache・Nginxを使用するUbuntu Linuxホスティングサーバー上で40件のWebサイトを管理し、高い可用性と性能を維持した。

UbuntuでPostgreSQL・MS SQLサーバー10台を運用

Ubuntu Linux VPS上にPostgreSQL・MS SQLサーバー10台を設計・デプロイ・保守し、性能と信頼性を維持した。

社内初のオブザーバビリティダッシュボードを構築

社内初のオブザーバビリティダッシュボードを開発・公開し、オフィスの大型テレビでシステム性能とデータをリアルタイムに把握できるようにした。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

すべての運用秘密情報をホストごとに分離 · Ansibleチェックモードを実態に即した検証に · デッドマンズスイッチによるホスト監視を構築 · 復元方法を明示したバックアップ自動化 · 独自のエラー監視・トレースを構築 · Azureへのプラットフォーム導入を一貫して担当 · GrafanaとPrometheusで運用リスクを低減

{ クラウド・インフラ・DevOps }

バックアップ・災害復旧

大きな障害のあとにすべてを取り戻す

あると信じるだけでなく、復元を試したバックアップと、
サーバーが起動しない朝に使える手順書を用意します。

Ubuntu Linux VPSインスタンス30台を管理

Ubuntu Linux VPSインスタンス30台を管理し、災害復旧方針の実装とネットワーク構成の最適化を行った。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

すべての運用秘密情報をホストごとに分離 · 復元方法を明示したバックアップ自動化 · バックアップと災害復旧をコードで実装 · Barmanで重要データのバックアップ100件を自動化

{ クラウド・インフラ・DevOps }

システム管理

日々の稼働を支える

サーバー、メール、アカウント、更新を日々管理
し、誰もが頼る仕組みを平穏に動かし続けます。

ファイアウォールルールを位置で検証

存在するルールが実際のパケットに到達されるとは限らないため、番号付きルール一覧と稼働中のパケットフィルターチェーンを読み、ファイアウォールルールを存在だけでなく位置で検証した。

全社共通のパスワード管理を導入

全社共通のパスワード管理システムを導入し、セキュリティを強化してアクセス管理を効率化した。

ベアボーンサーバー100台を管理

ベアボーンサーバー100台、物理ネットワーク、IP電話システムを管理し、会社の成長を支えるインフラを維持した。

Wi-Fiルーター1,000台を設定・導入

1,000台のWi-Fiルーターを設定・導入し、顧客のネットワーク接続性と性能を改善した。

ネットワークインフラを管理

1,000台を超えるサーバーのネットワークインフラを管理し、適切なシステム構築、セキュリティ、障害対応を行った。

WireGuard・IPsec VPN接続8本を管理

WireGuard VPN・IPsec VPNの8接続を管理して問題に対処し、Google CloudとLinuxシステム間の安全な通信を確保した。

M365・SSO全体のアクセス管理を主導

M365、1Password、Red Hat SSO、OKTA SSOを用いたアクセス管理を主導し、セキュリティ侵害を防いだ。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

読みやすいTor onionミラーを公開 · 6収集処理が省略した入力を明示 · gitホスティングCIを試し2不具合を発見 · 自社サーバーでコードを管理 · 読み取り専用のホストレポートロールを7個構築 · 全ユニットのsystemdサンドボックス露出を低減 · 20レコードのDNSゾーンをコードで照合 · デッドマンズスイッチによるホスト監視を構築 · 復元方法を明示したバックアップ自動化 · 実行ごとに侵入元の遮断を実証 · SSHの24設定項目を検証・堅牢化 · 小型サーバー上の公開サービス · リポジトリで定義するインフラ · 基幹インフラ全体を一から構築 · 数年後も稼働するインフラを構築 · IPTV配信信号の連続稼働を維持 · 約1,000台のIPTV/OTTサーバーを24/7で支援 · Apache・NginxでWebサイト40件を管理 · Ubuntu Linux VPSインスタンス30台を管理 · Linuxサーバーを守る1,000件のRBACルールを作成 · UbuntuでPostgreSQL・MS SQLサーバー10台を運用

{ クラウド・インフラ・DevOps }

ネットワーク・VPN構築

拠点と働く人をつなぐ

Wi-Fi、ファイアウォール、リモートアクセスを整え、どこからでも働け、権限のある人だけが接続できるようにします。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

2台目のサーバーをコードで構築 · 20レコードのDNSゾーンをコードで照合 · ファイアウォールルールを位置で検証 · 3つあったが、どれも機能していなかった防御 · IPTV配信信号の連続稼働を維持 · Azureインフラをコードとして構築 · ベアボーンサーバー100台を管理 · Wi-Fiルーター1,000台を設定・導入 · ネットワークインフラを管理 · WireGuard・IPsec VPN接続8本を管理 · Ubuntu Linux VPSインスタンス30台を管理

{ クラウド・インフラ・DevOps }

セキュリティ・アクセス管理

誰が何にアクセスできるかを決める

アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

認証情報漏出後に対象範囲の規則を記述

構成変更によって別会社の資産一覧、ファイアウォール許可、文章が混入したことを受け、リポジトリの対象範囲を定める規則を記述し、隔離した残存物も除外せず秘密情報検査の対象に残した。

企業所有権の申請・審査を構築

利用者による企業所有権の申請から管理者の審査までを実装し、承認時に編集権限を決める認可グラフが更新される、一連の仕組みを構築した。

Stripe・アプリ内購入の利用権限を構築

StripeとApple・Googleのアプリ内購入に対応する決済・利用権限層を構築し、サーバーで検証する11テーブルのアクセスモデルによって、一覧、検索、エクスポートの利用を制御した。

nonce付きCSPでアプリケーションを堅牢化

nonce付きCSP、HSTS、SameSite Cookie、最小権限のデータベースロール、サーバー側の利用権限再検証により、アプリケーションを堅牢化した。

利用者の組織コンテキスト切り替えを構築

クライアントのlocalStorageとサーバー側Cookieへの複製を用いて組織の操作コンテキストを切り替える仕組みを設計し、最小権限の認可を維持しながら管理対象の組織として操作できるようにした。

チームの意思疎通と協働を改善

Slack、Mattermost、1Password、Jiraを導入してチームの意思疎通と協働を改善し、8,000時間の作業を削減した。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

読みやすいTor onionミラーを公開 · 追跡せずに13アドレスを測定 · 公開するすべての部品のライセンスを把握 · 未使用だったコアの半分の機能を実装 · すべての運用秘密情報をホストごとに分離 · 2台目のサーバーをコードで構築 · PodmanとQuadletでコンテナを運用 · 全ユニットのsystemdサンドボックス露出を低減 · 20レコードのDNSゾーンをコードで照合 · 秘密情報の1秒の事前検査を追加 · ファイアウォールルールを位置で検証 · 実行ごとに侵入元の遮断を実証 · SSHの24設定項目を検証・堅牢化 · 3つあったが、どれも機能していなかった防御 · 継続的なセキュリティ検査を運用 · 異常時拒否のレート制限・不正利用対策を構築 · 全社共通のパスワード管理を導入 · チームの協働を自動化 · Azureインフラをコードとして構築 ·

切り替えと配信ログを備えるメール機能を構築 · Dockerアプリ100個のSSL/TLS証明書を自動化 · ネットワークインフラを管理 · WireGuard・IPsec VPN接続8本を管理 · M365・SSO全体のアクセス管理を主導 · Linuxサーバーを守る1,000件のRBACルールを作成

{ クラウド・インフラ・DevOps }

AIエージェントのセキュリティ

AIが扱ってよいものだけを扱わせる

各エージェントのアクセスを必要な範囲に限定し、エージェント自身が改変できない場所に操作を記録し、安全策を信じるだけでなく検証します。

このサービスは、このページの別の箇所に掲載した仕事にも含まれています：

すべての運用秘密情報をホストごとに分離 · 認証情報漏出後に対象範囲の規則を記述 · 秘密情報の1秒の事前検査を追加 · 実行ごとに侵入元の遮断を実証 · 3つあったが、どれも機能していなかった防御 · 継続的なセキュリティ検査を運用 · 異常時拒否のレート制限・不正利用対策を構築 · M365・SSO全体のアクセス管理を主導 · Linuxサーバーを守る1,000件のRBACルールを作成

ご自身の出発点と比べる

比較できる実績を選び、ご自身の状況との違いを明らかにしてください。

料金を見る お問い合わせ →

状況に合う進め方を相談する

関連する経験を、ご評価いただける進め方に具体化するようご依頼ください。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://platform.engineer.company/ja/portfolio/