SecHack365

hakolate

低スペック端末で動作するIDE
hako+chocolate

開発駆動コース 仲山ゼミ
加藤 颯

これまでの開発環境の課題 大きく分けると3つの課題がありました

課題1 コードを書き始めるまで障壁が高い

事前準備でやる気をなくしがち
手元にiPadはあるけどパソコンがない

ブラウザで動かせるようにしよう！

課題2 環境構築でつまずきがち

PATHの通し方がわからない
どのランタイムを入れたら良いのかわからない

簡単に動かせて**壊せる**環境にしよう！

課題3 iPadで開発がしたい

iPadではSwift以外のランタイムを動かせない
ブラウザで動くIDEは存在するが処理がもっさりしている

サーバで処理させよう！

さらに2022年になり情報Ⅰが共通必履修科目へ

GIGAスクール構想により、iPadなどのタブレット端末が教育現場で普及

MDMなどによる規制でアプリなどインストールが難しい
環境構築の**マニュアルを作っても**生徒が全員出来るわけではない
端末ごとに挙動が違うと**授業がしにくい**

本番環境だと**壊すと怖い**ので自由に**遊べる環境**があると便利

SecHack365を通してやってきたこと

第1回イベント	第2回イベント	第3回イベント	第4回イベント	12月イベント	第5回イベント	発表会
ペルソナ設定 iPadファーストなターミナルという作品像	技術の素振り どのように作品にセキュリティを落とし込むか	デモを示す タブレット端末でのキーボード操作問題が生まれた	DockerからK8sへ アップデートのたびに改修 CI/CDを作る大切さを学んだ	Dockerベースに戻す CI/CDを整えた Passkey対応に取り組んだ	Remixへ移行 Rootless対応 ベンチマーカー作り	バグ修正 コストへの対策 コンテンツ作り

入力の工夫

ソフトウエアキーボード前提のツールに使用としていた
画面の占領率が高いため、別のアプローチを探ることに
アプリ化して独自のキーボードを作ることも検討したがコスパが×
物理キーボードのみにし、ほかの箇所でユーザの利便性を高めること

DockerからKubernetes(K8s)からDockerへの変遷

Docker単体ではオーケストレーションが不可能
多くの使用ケースがあるK8sベースの基盤へ移行を実施
使っていたライブラリの扱いが非常に難しく、開発の難易度が上昇
Dockerへ戻る(Docker Swarmを噛ませると運用が出来る)

SSRへの移行

低スペック端末に焦点を当てたので、それ相応のアプローチが必要
サーバサイドで処理が走るSSRを採用
体感ではあるがスペックの低い端末で体験が向上
SSG/SPAに比べるとデプロイが大変だが、CIを整備し楽に

コストへの対策

サービス安定化のため、ランタイム基盤を切り離す
ユーザがランタイム部分を提供することにより、コストが浮く
できる限りAWSへ移行することにより、詳細なデータを取れる
→ボトルネックの監視が容易に

hakolateの製作過程にて

こだわりポイント

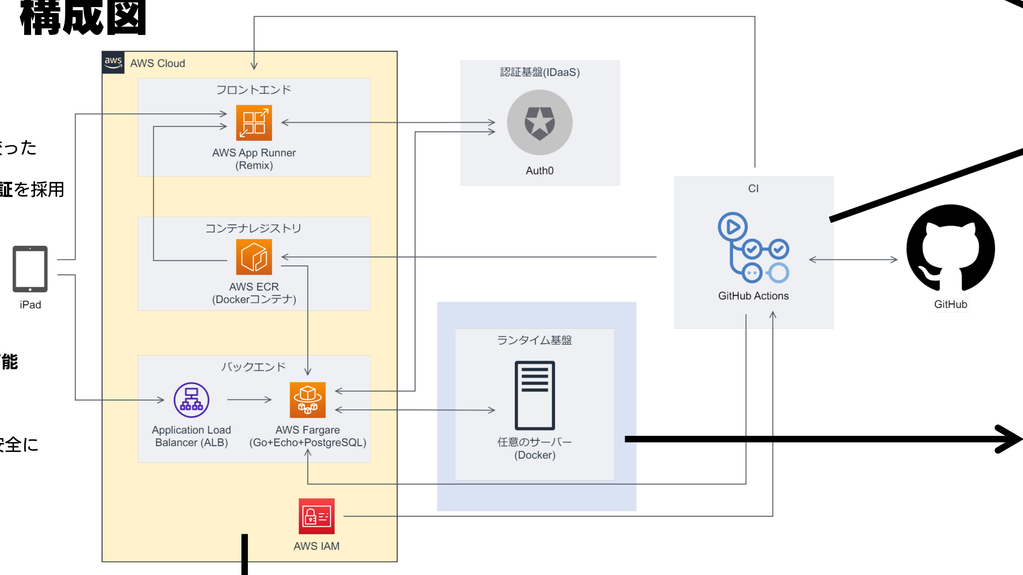
- [ ] ユーザが触る部分はAWSに、開発者が触る部分はGitHubにと**Attack Surface**を絞った
- [ ] GitHubではAWS IAMのシークレットを利用せず**OpenID Connectを利用した認証**を採用
- [ ] GitHubリポジトリに**Renovate/DependaBotを常駐**させる

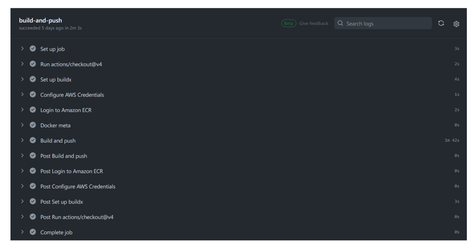
得た効果

- [x] AWSはCloudTrailによるログ記録により、インシデントが起きた際に**詳細調査が可能**
- [x] シークレットキー自体を発行しなくても良いので、**流失リスクがゼロ**になった
- [x] パッケージに更新があれば、テストCIを通過すれば**自動Merge**をするので、より安全に

開発環境がよりセキュアに

構成図





AWS ECRにCIで構築したコンテナイメージをアップロードしている様子

実際にできあがったもの

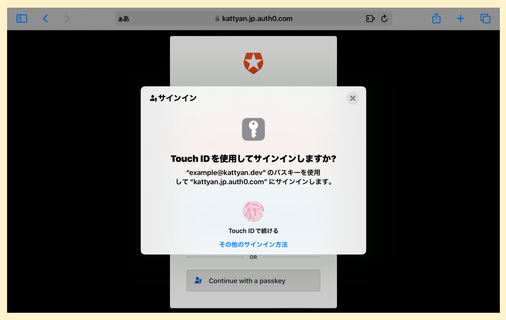
※写真は開発中のスクリーンショットです

AWSで実行

シンプルなUIとパスワードが不要な認証で扱いやすくなり**課題1**を解決

ユーザ

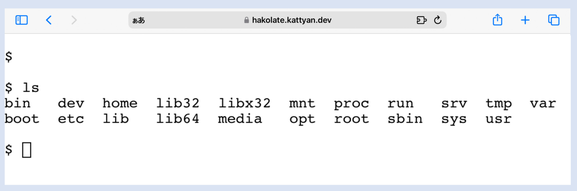
ホーム画面



ログイン画面

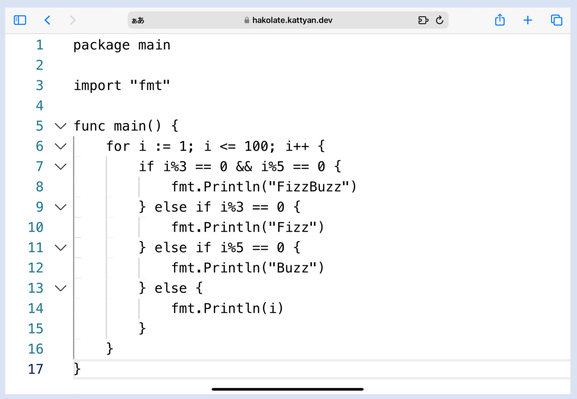
ランタイム基盤で実行

簡単に立ち上がるシェルで**課題2**を解決



シェルが動いている様子

ブラウザ上から編集可能で**課題3**を解決



エディタが動いている様子

hakolateを支えるセキュリティ

ユーザから見える部分 – Passkeyを認証に採用

はじめはGitHubやGoogleが提供しているOpenID Connectを採用しようとしていた
実際に実装してみて、ユーザがパスワードを忘れた際のワークフローがややこしい
パスワードマネージャーも全ユーザーが使えるわけではない
そのため**パスワードを使わないPasskey認証を採用**

＜参考:Passkeyについて＞

- フィッシングサイトによる詐欺的認証試行が防げる
- 公開鍵暗号方式を用いた認証方式
- 異なるサービス間でPasskeyの共有が可能

これら3点を踏まえ、従来のパスワード認証よりも安全かつ簡単な認証体験を享受できる

処理を動かすランタイム基盤 – セルフホストを可能に

自分で調達したサーバで実行できるようにすることで、自宅サーバなどを運用している人にとって刺さりやすい
セルフホスト先では好きなイメージをダウンロードできかつ選択できるようにする（予定）

自分でイメージを作成することができ、**幅が広がる**
hakolateの**運用コストを減らす**ことができ、安定的な運用へとつながる

Rootlessで実行可能に

ユーザーネームスペースの有無にかかわらずDockerデーモンはroot権限を必要とするので、デーモンに脆弱性があると、**他人のコンテナにアクセス可能**である
Rootlessで実行することにより、デーモンがuser空間で実行させるため、**安全に動かすことができる**

セキュアでユーザーフレンドリーな世界へ

Passkey、セルフホスト、Rootlessの3機能によって、使用する側と運用する側の両方にとって、使いやすく安全な基盤を提供することに大きく貢献
セキュアでユーザーフレンドリーな世界を構築するためには開発者が使い込み、ユーザの声を聞き、継続的に改善していくことが重要
2つのバランス（セキュリティとユーザビリティ）が重要であり、どちらかに偏るのは良くない

技術自体をOSSにして、開発方針を示す（GitHub ActionsでCIを組んでいるので、リソースへのアクセスが容易なため要注意）
コミュニティを構築する（Googleなどでも引っかかってほしいのでIssue等で対処）

これまでの世界

課題1 コードを書き始めるまで障壁が高い
課題2 環境構築でつまずきがち
課題3 iPadで開発をしたい

hakolateがある世界

ブラウザで**hakolateにアクセスすれば**ターミナルが立ち上がる
すべてインストール済みの**コンテナイメージを用意すれば**何も入れなくて良い
エディタ機能もあるので、**気軽に試せる**

ブラウザから扱えるので**授業教材として**使える
生徒に気軽に**壁打ち道具**として使ってもらえる

休み時間にも**「ものづくり」**ができる
家に帰って別端末でも**再現できる**

セルフホストが出来て**自宅サーバ**でも動く
マイクラサーバ構築 → **hakolateセルフホスト?**

hakolateを実際に作ってみて

1年間いろいろな技術を触ってみてそれぞれの面白さや奥深さを感じることが出来た
プロダクトにセキュリティを持ち込むのは技術を知った上に設計も必要でハードルは高かったが、実践できた
AWSやGitHub ActionsでCIを構築したりとクラウドネイティブな開発が出来て、時代の流行に乗りながらそれらの問題点も議論できた
ボトルネックになる部分の特定だったり、監視の技術を身につけることができ、これからのものづくりに生かせる一生物の力を手に入れた

今後の展望

GUI機能を実装してQEMUが動かせたりとよりhakolateの活用幅の広がる機能を実装する
一括管理が出来るような便利で使いやすいコンソール画面を実装し、安心してつかるような仕組みを整える
現状はコンテナ技術の上にランタイムが走っているが、WASMでここの技術を置き換えることも検討している
認証基盤に関しては、現状は外部の基盤を組み込んでいるがPasskeyベースの認証基盤を自作することを視野に入れている（IDaaS自作）

NICT National Institute of Information and Communications Technology

1年間支援してくださった仲山トレーナーをはじめとするSecHack365事務局、
トレーナー、アシスタント、トレーニーの皆様へ心から感謝申し上げます

ここから体験できます
https://hakolate.kattyan.dev/