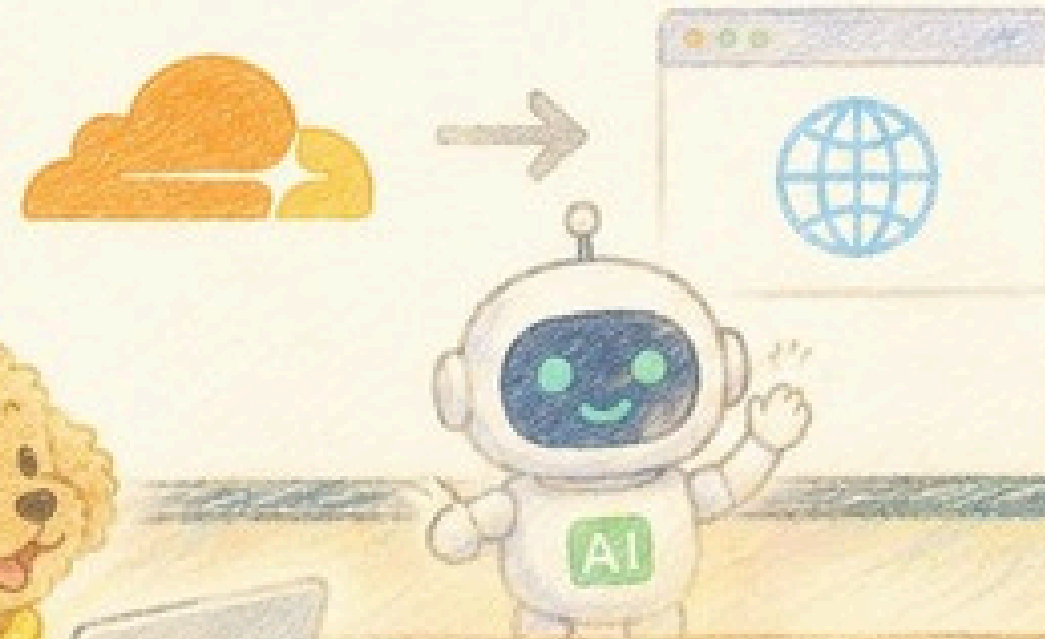


ククデ会

クラフレにクロコでデプロイしようの会



タツヲ

タツヲ (yto)
<https://x.com/yto>

目的

技術要素に対する解像度を
初心者レベルから何段か上げて
自分で決められるようにする

AI にざっくり依頼して、やりとりしながら承認していただくだけでも、Web サイトや Web アプリは作れる。

ただ、AI は何を頼んでも「できますよ」と言う。それが意図通りにできるのか、自分の手に負えるのかを判断できるのは自分だけ。

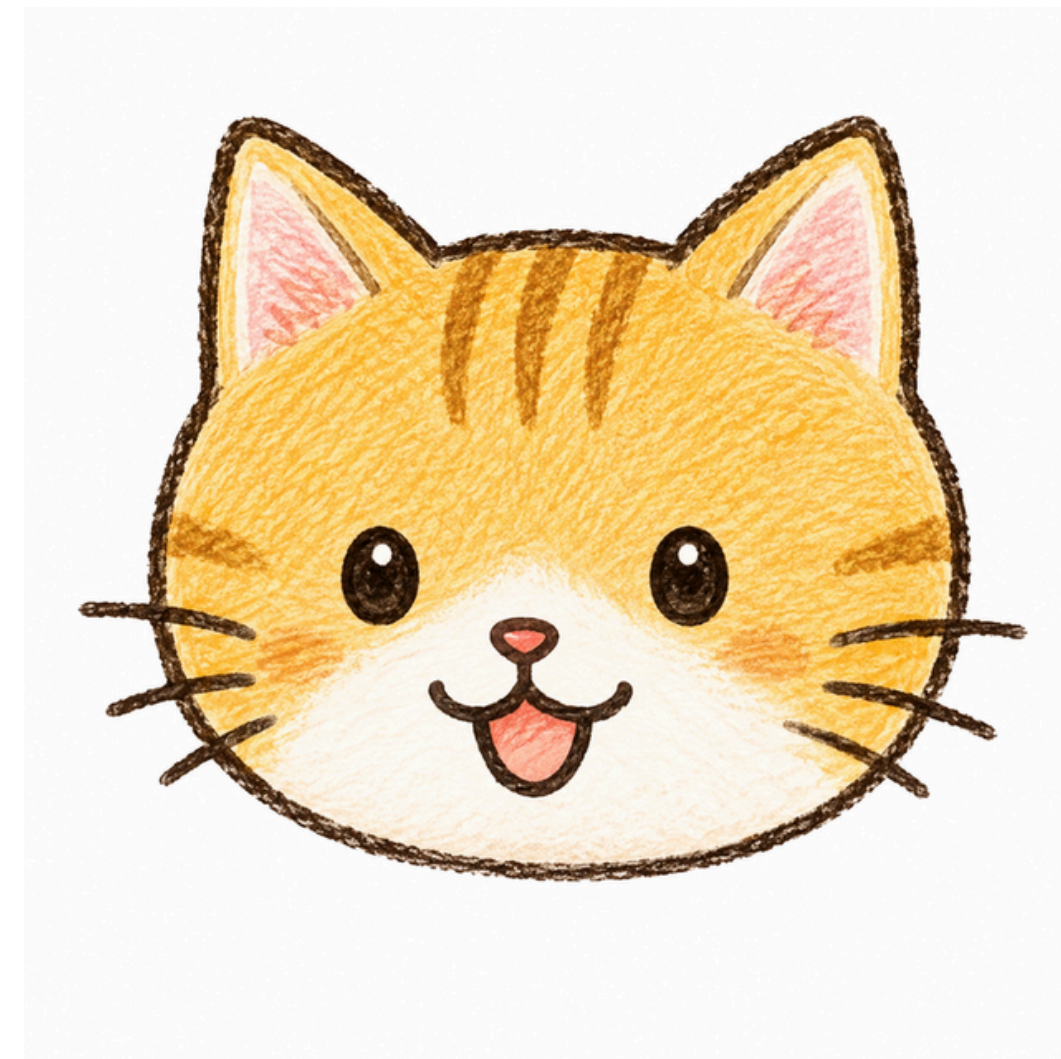
裏で何が起きているかを知っておくと、そこに見当がつく。何を作るか決めるとき、詰まったとき、応用したいとき、コストや権限を判断するとき。

エンジニアになる必要はない。見当がつくところまで、理解を後押しするのが目的。



ククデ会で扱うこと

- クロコでWebサイト/Webアプリを作成
 - プロンプト、Claude Design
- クラフレへのデプロイの方法
 - ブラウザ、コマンド（Wrangler）、GitHub 経由
- コード管理
 - Git、GitHub、リポジトリ、コミット、プッシュ
- データベース（Cloudflare D1）の扱い方
 - 設計、作成、連携、マイグレーション



Vibecoding Notes

<https://vibecodingnotes.com/>
ククデ会のために作った手順書をまとめたサイト

 Vibecoding Notes

🔍 検索

🌙 ライト

Vibecoding Notes

Claude Code に指示するだけで、Web サイトや Web アプリを作って公開できる。そのためのハンズオン集。

はじめよう →



Claude Code にざっくり依頼して、やりとりしながら承認していただけても、Web サイトや Web アプリは作れる。ただ、裏で何が起きているかを理解しておく、詰まったとき・応用したいとき・コストや権限の判断が必要ときに自分で見当がつく。本サイトはその理解を後押しするのが目的。HTML だけの静的サイトから、データベースを使った Web アプリの公開までを扱う。想定するのは個人・小規模な用途で、大規模サービスや商用グレードのシステム構築は対象外。

想定環境: Mac (MacBook など)、Google アカウント、Claude Pro 以上、Claude デスクトップアプリ

 Vibecoding Notes

🔍 検索

🌙 ライト

ロードマップ

- VST バイブコーディングをはじめよう
- SUP Cloudflare に HTML をアップロードして公開する**
- WRG Wrangler で Cloudflare に公開する
- CGT Git 連携で Cloudflare に公開する
- DIG D1 でデータ共有型 Web アプリを作る

応用編

- AST Astro でブログを作って公開する
- ASB Astro ブログを運営していく
- ADM 管理画面 + Cloudflare Access で制限する

リファレンス

- CDM Cloudflare Pages へのデプロイ 4 つの方法
- TWA Web アプリの 2 つのタイプ
- ARC データ共有型 Web アプリの基礎知識
- UIT AI への指示に使える UI 用語集
- GBA Git と GitHub の基本 — ローカルから始めて GitHub につなげる
- GHA GitHub Actions で Cloudflare 自動デプロイ
- DOM 独自ドメインを設定すること
- MCM Cloudflare MCP でできること
- MDB Markdown の基本
- HBS Homebrew をセットアップする
- CDG Claude デスクトップアプリで使う

アーカイブ

[➤](#)

Cloudflare に HTML ファイルなどをブラウザでアップロードして Web サイトとして公開する

SUP Slide CloudflarePages ClaudeDesign

Cloudflare には、HTML・CSS・JavaScript などの静的ファイルをブラウザからそのままアップロードして、Web サイトとして公開できる機能がある。

このハンズオンでは、Claude Code で作った HTML ファイルや Web アプリを Cloudflare にアップロードし、実際にインターネットへ公開するところまでを体験する。

1. アカウント作成から公開まで一気に体験する



接続、保護、構築をどこでも

Cloudflareは、Webサイト、アプリ、AIエージェント、ネットワークの保護とセキュリティを向上させます。当社の変換可能なSaaSプラットフォームは、AIの安全な導入を加速し、当社の開発者向けプラットフォームは、AIアプリの開発、運用に最適な場所です。

無料で始める 料金設定を確認

Cloudflare サイトトップ

1-1. Cloudflare アカウントを作る

- Cloudflareの[サイト](#)にアクセスして、右上の ログイン → Sign up (サインアップ)
- Google アカウントがあればそれでサインアップすると楽

目次

概要

- アカウント作成から公開まで一気に体験する
 - Cloudflare アカウントを作る
 - アップするファイルを準備する
 - Cloudflare にアップロードして公開する
 - サイトを更新する
- Web アプリを作って公開する
- 複数ページの Web サイトを作って公開する
- DESIGN.md でデザインの方針をそろえて公開する
 - Awesome Design MD JP のギャラリーから DESIGN.md を取得する
 - Claude Code に適用を依頼する
 - 公開して確認する
- Claude Design でデザインを作って公開する
 - Claude Design でデザインを作る
 - Claude Code に引き継ぐ
 - Claude Code で細部を調整する
 - Cloudflare にアップロードして公開する
- 公開後にやっておきたい仕上げ
 - favicon
 - ページタイトル・description
 - OGP (SNS シェア対応)
 - 404 ページ
 - PWA (スマホアプリ風 Web アプリ)
- 次のステップ

目次

- 初級：ブラウザ経由でデプロイ
- 初級：Wranglerでデプロイ（クロコだけでデプロイ）
- 初級：Webアプリの2つのタイプ（前編）
- 開発基礎：Git と GitHub
- 開発基礎：作業メモの管理
- 開発基礎：公開後のセキュリティ点検
- 中級：アクセス制限
- 中級：Webアプリの2つのタイプ（後編）
- 中級：データベース

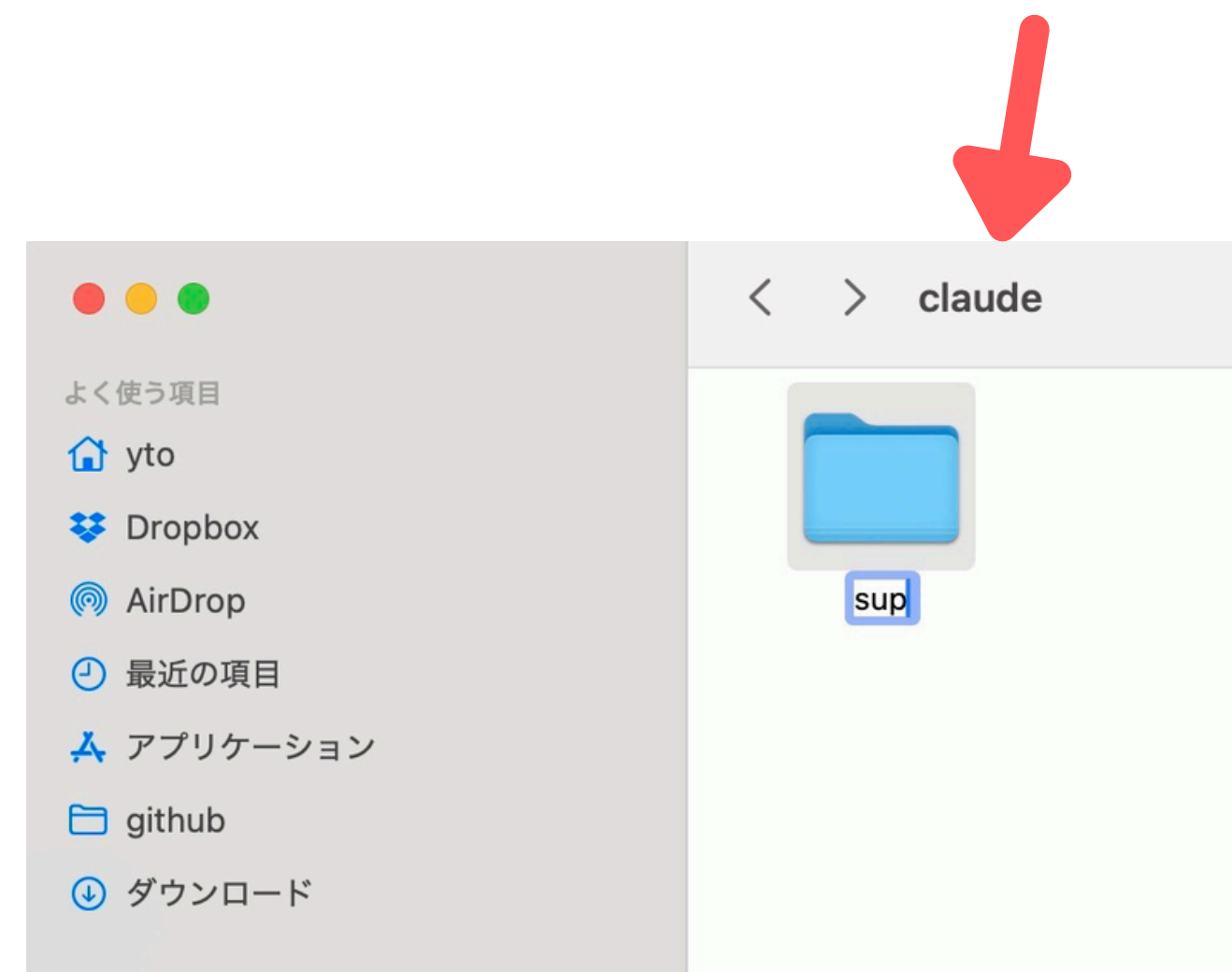
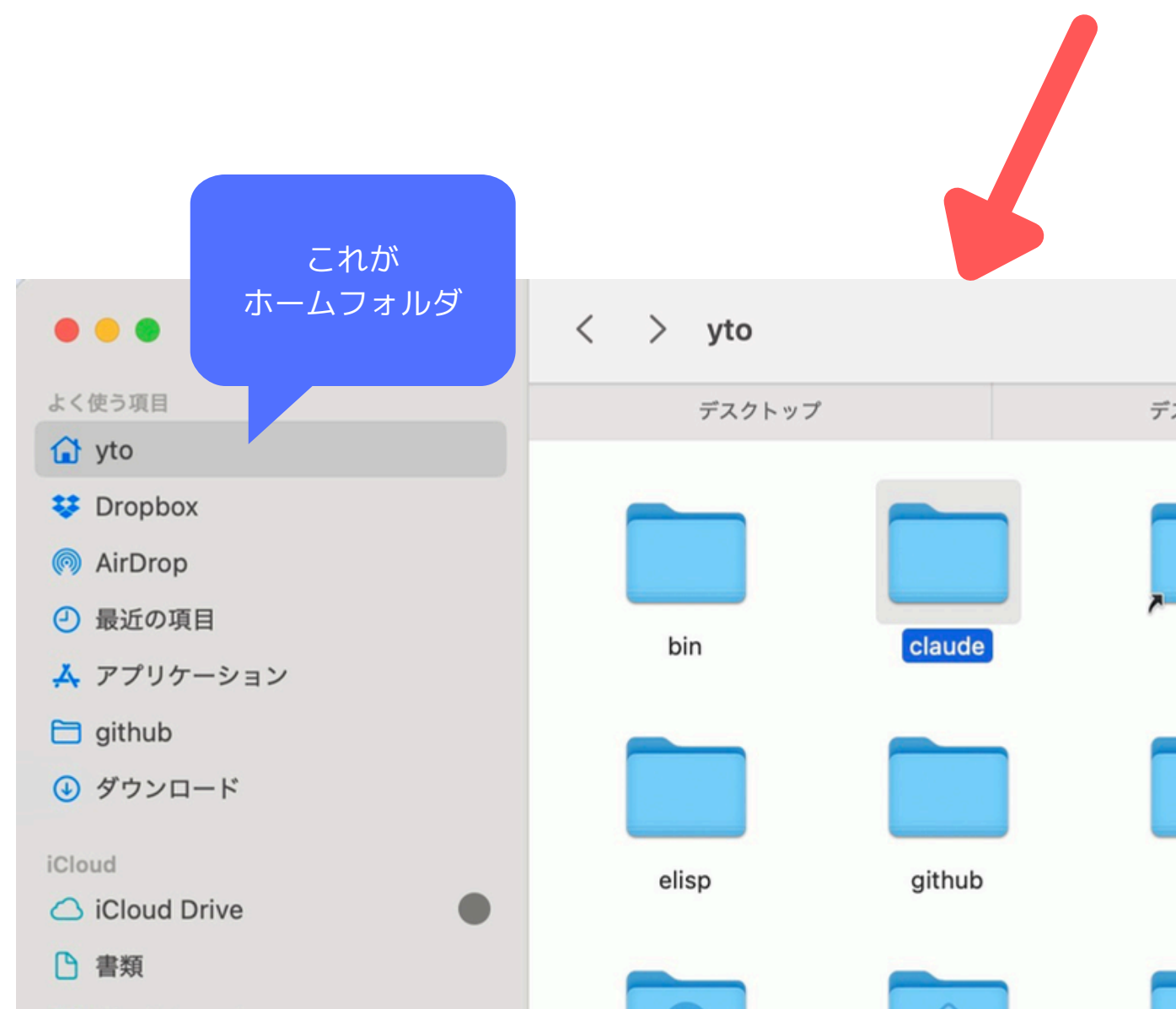
初級

ブラウザ経由でデプロイ

[SUP] <https://vibecodingnotes.com/handson-cloudflare-static-upload/>

まず作業フォルダを作成する

Finderで**ホームフォルダ**の直下に **claude** フォルダを作り、
その中にWebサイト用フォルダ **sup** を作る

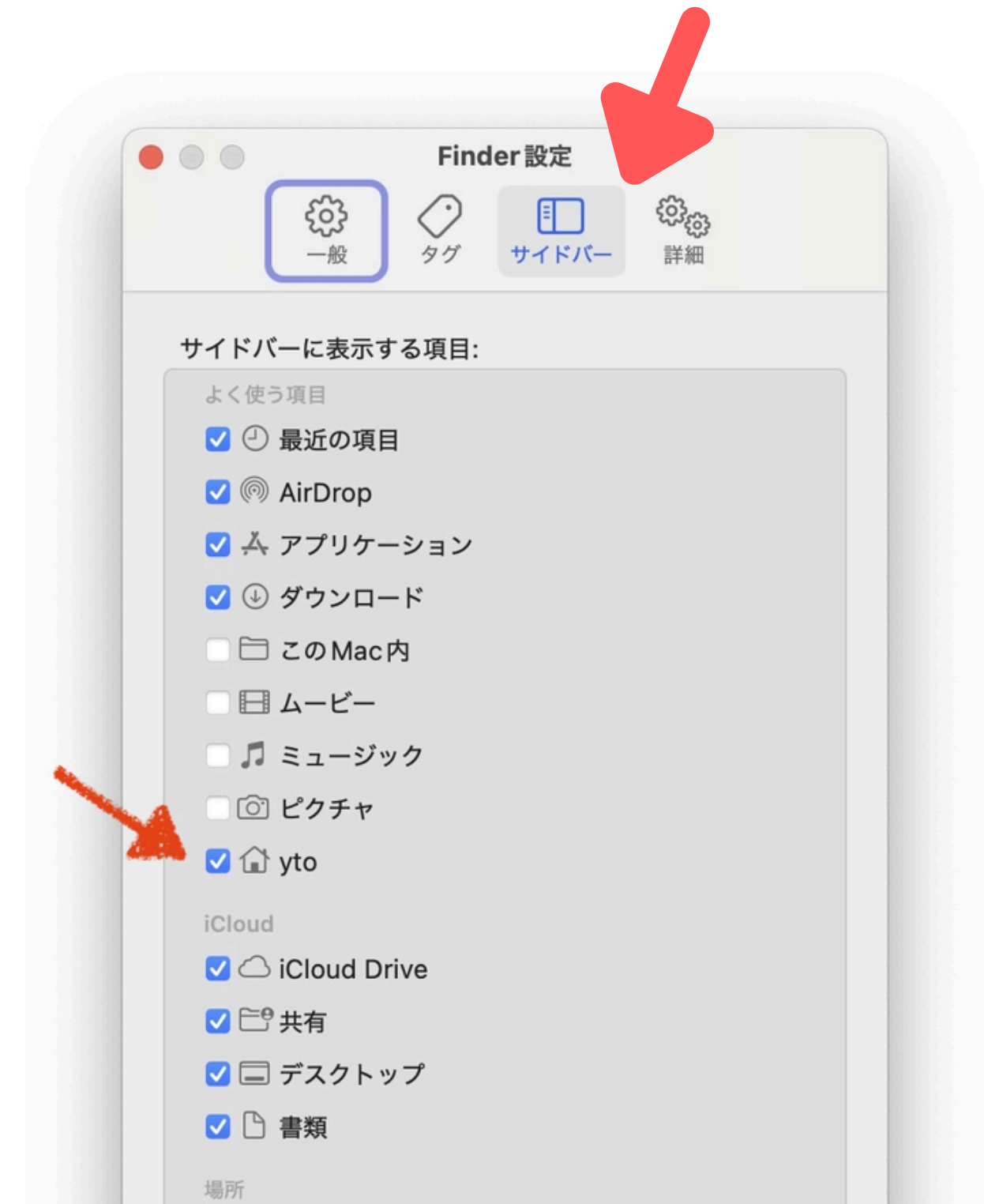


Finderにホームフォルダがない場合

Finder → 設定 → サイドバーを開き、

「サイドバーに表示する項目」→
「よく使う項目」

の中の家アイコンと自分のユーザ名の
項目にチェックを入れる



Claudeデスクトップアプリ

- Claudeデスクトップアプリを起動
- 左上の“Code”を選択
- 新規
- 「ローカル」と書かれたボタンの横のフォルダアイコン → フォルダを開く
- さっき作った作業フォルダ（ホーム → claude → sup）を選択



こうなる！

自動モードにしない

- 画面左下のモード選択を **編集を受け入れる** にしておく
- 自動モードだと Claude が一気に進めてしまい、意図しない結果になりがち
- まずは一つずつ承認しながら進めるのが安全



使うモデルは Sonnet でOK

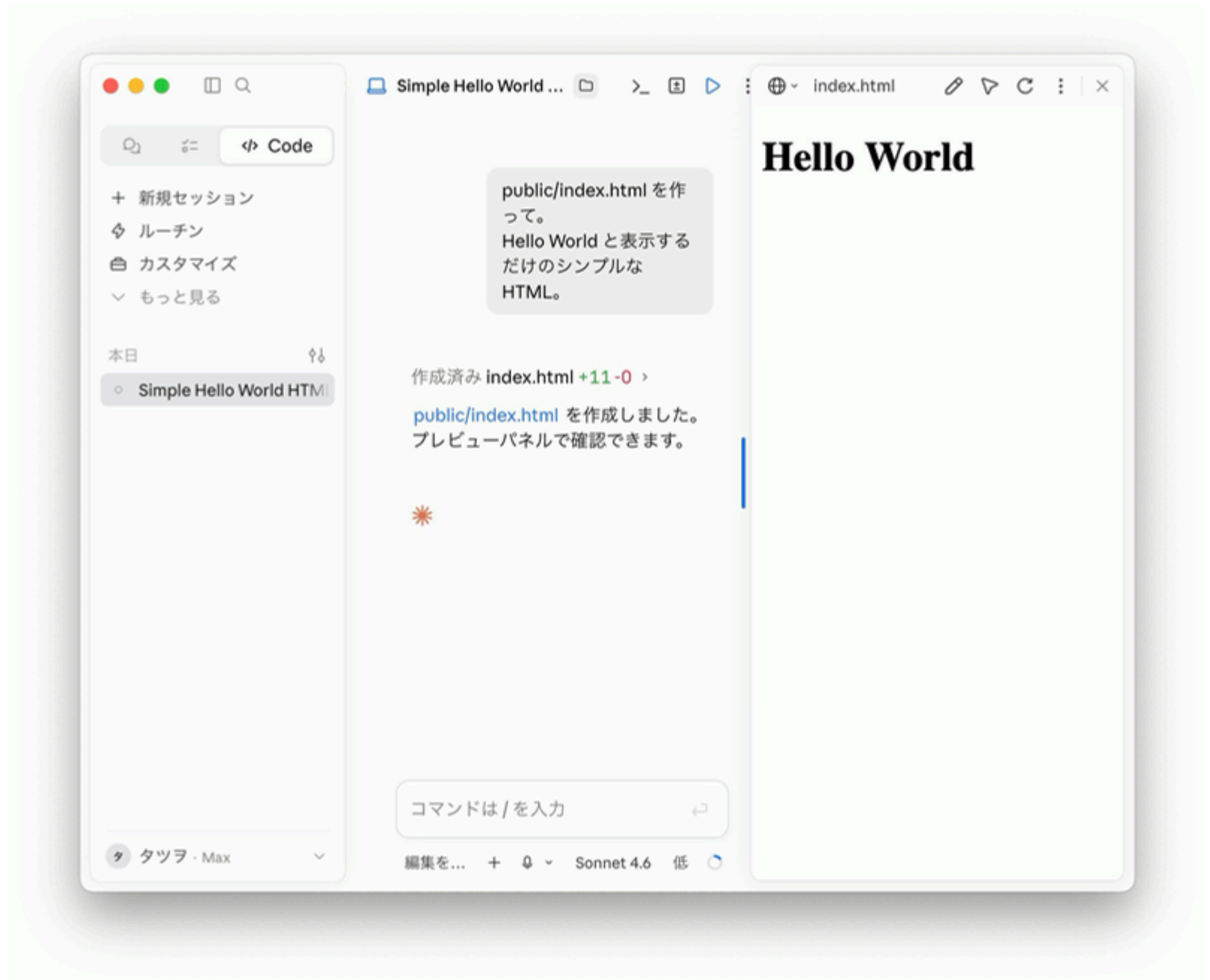
- Pro プランで Fable 使って一瞬で使用制限オーバー（ククデ会での事例）
- Opus でも良いが、Sonnet で問題なし



簡単なWeb サイトを作る

プロンプト：

`public/index.html` を作って。
Hello World と表示するだけの
シンプルなHTML。

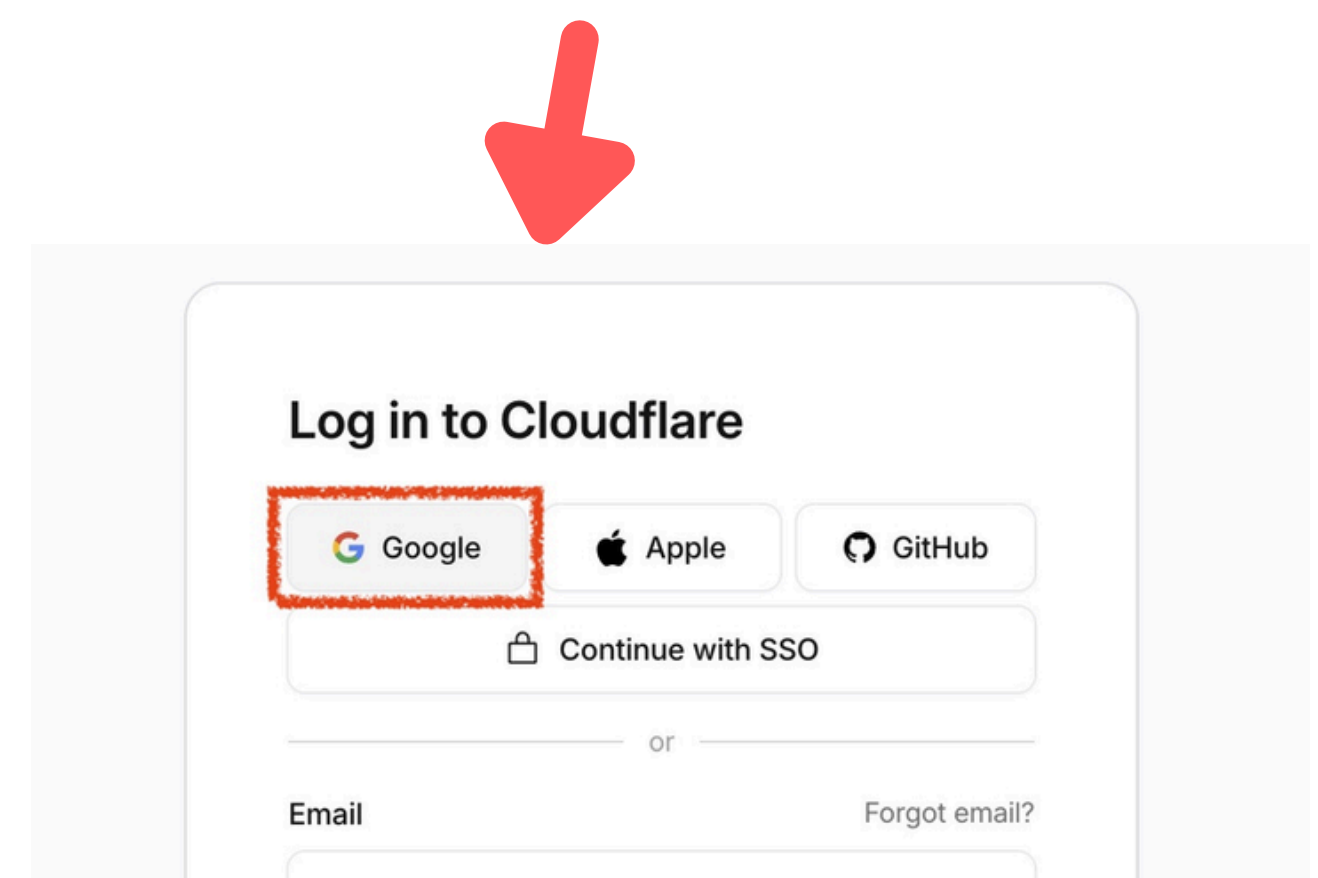


公開するファイルは public フォルダに

- アップロードするのは public フォルダだけ
- そこにあるものだけがネットに公開される
- 見せたくないものは public の外に置いておく
- 分けておくと、関係ないものまで公開する事故を防げる

Cloudflareのアカウントを作る

<https://www.cloudflare.com/ja-jp/> → 右上の ログインボタン → 登録



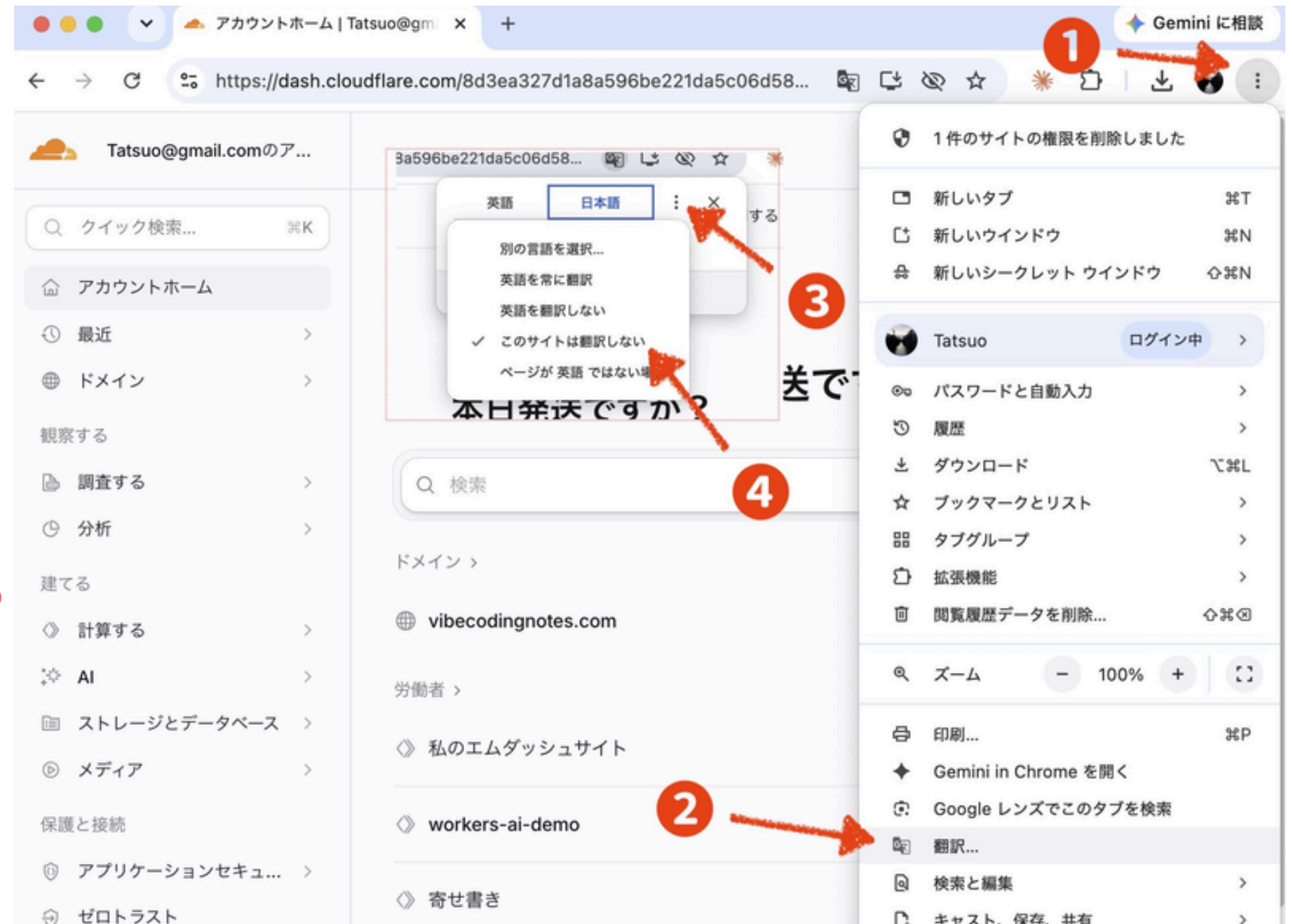
初めてログインしたとき

1. 利用目的などを聞かれることがあるが Skip でOK!
2. 表示を英語にする（この先の資料が英語表記ベースなので）

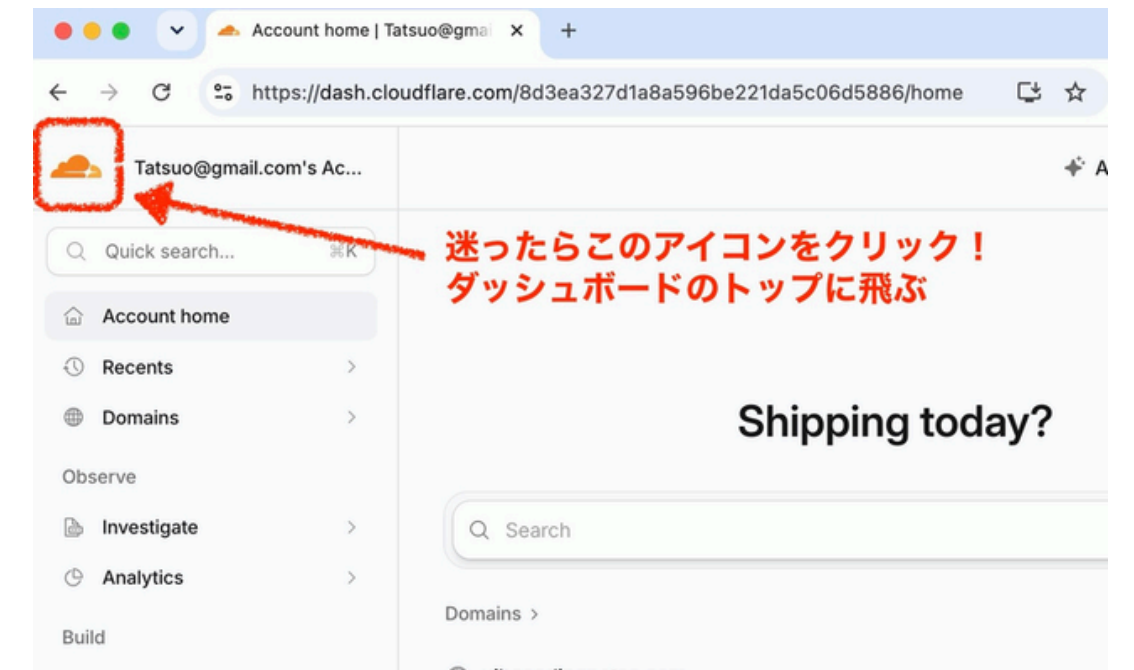
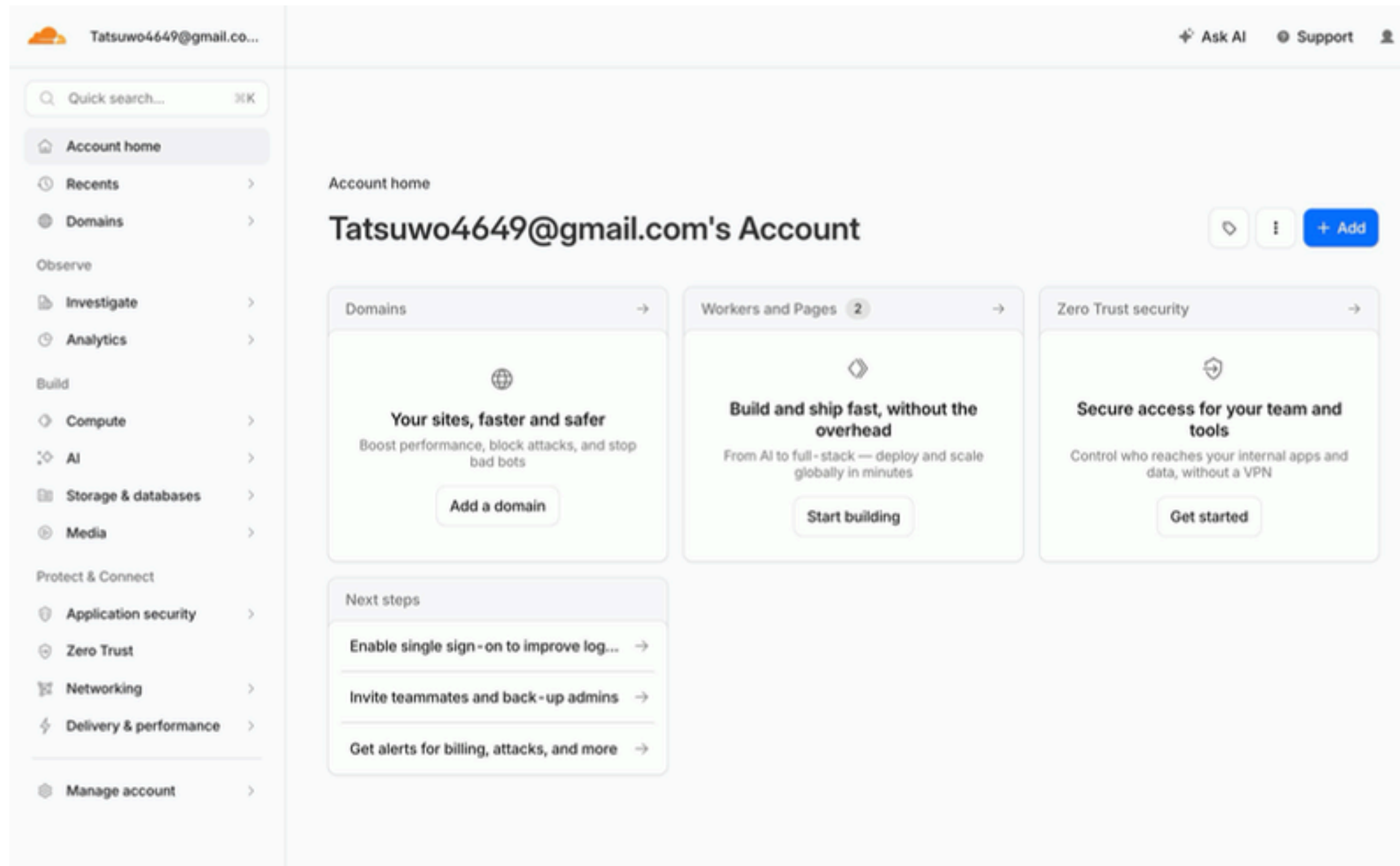
Chromeの自動翻訳機能をオフにする

- ① Chromeの右上の ⋮（メニュー）
- ② 翻訳
- ③ 表示されたポップアップの ⋮
- ④ このサイトは翻訳しない

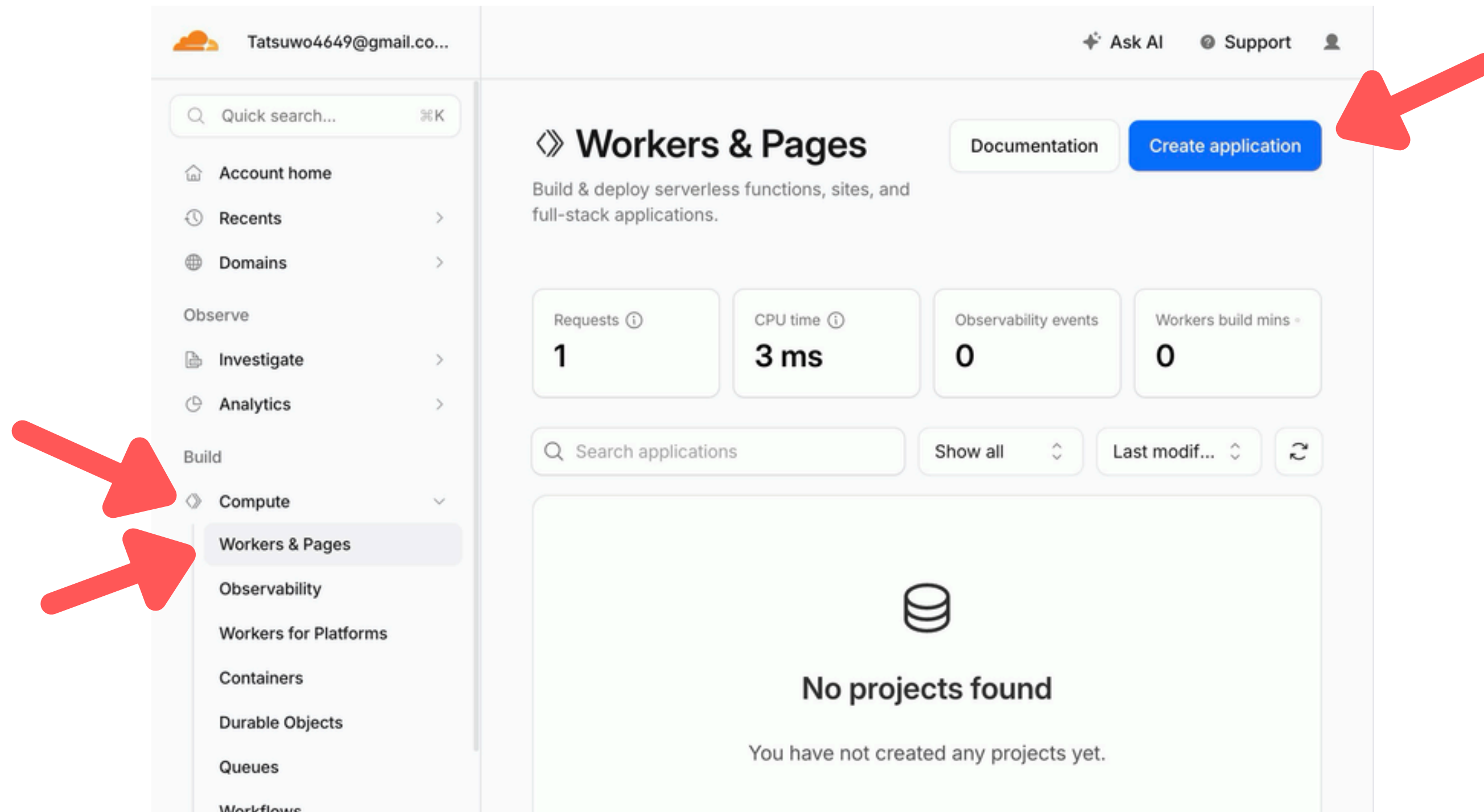
それでもメニューに日本語が残る場合は、
Cloudflareのダッシュボード右上の 👤 アイコン → 言語 → English



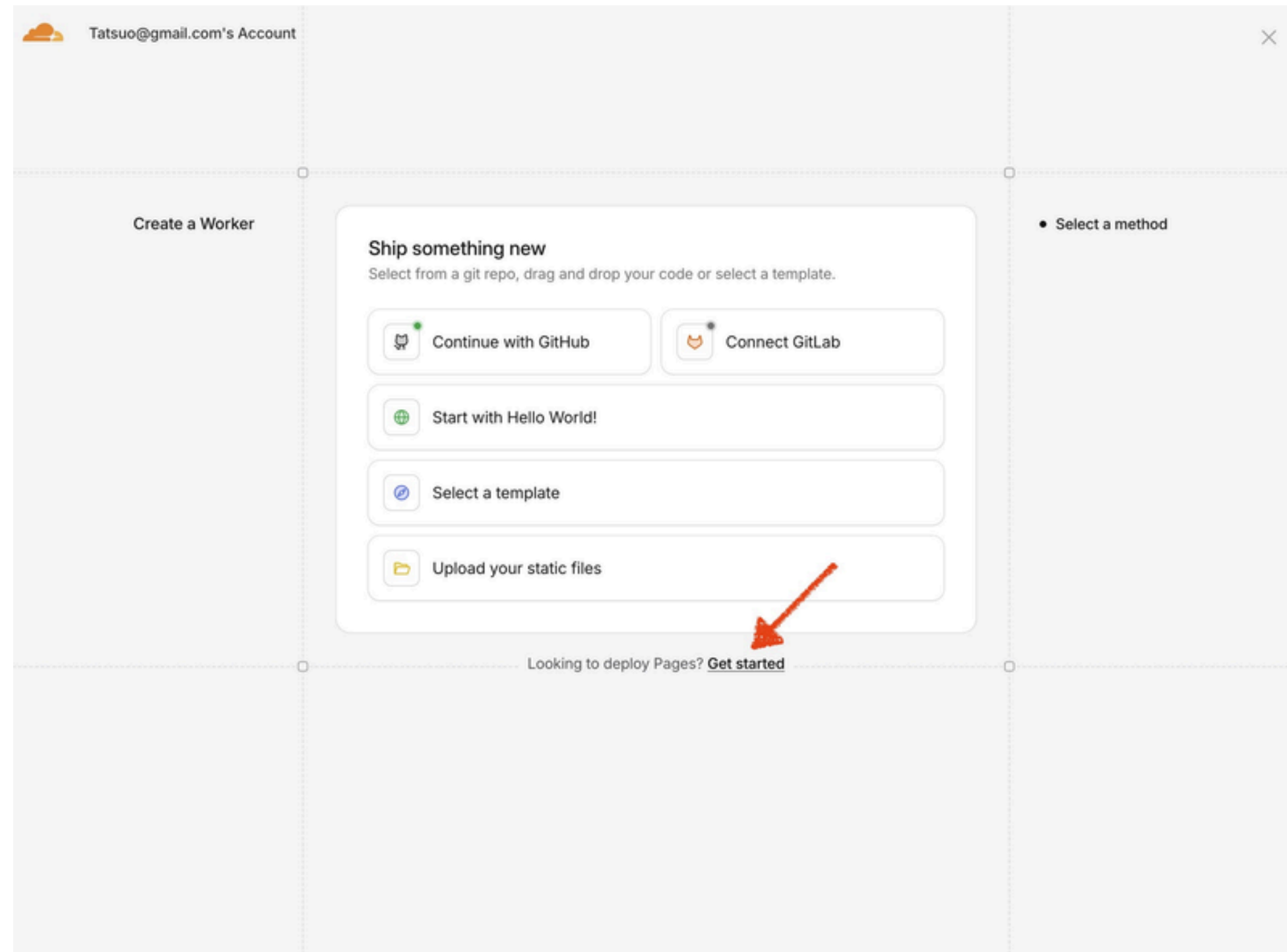
ダッシュボード



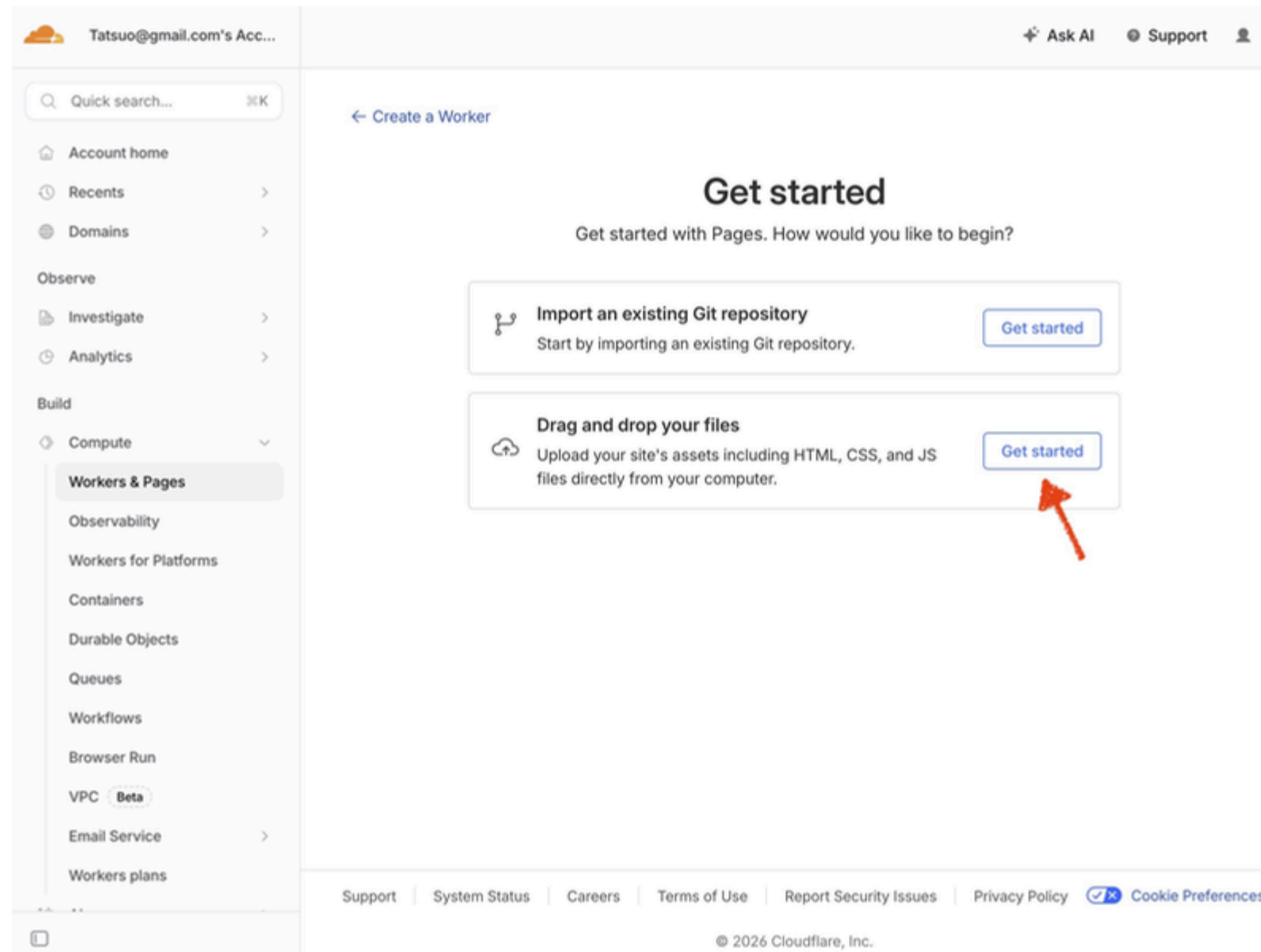
プロジェクトの作成①



プロジェクトの作成②



プロジェクトの作成③



プロジェクトの作成④

← Create an application

Deploy a site by uploading your project

Name your project and upload your site's assets including HTML, CSS, and JS files that you would like to deploy.

- 1 Create a name for your project:**

Project name

This name will be used to generate a permanent hostname on pages.dev.

Create project
- 2 Upload your project assets:**

Deploy

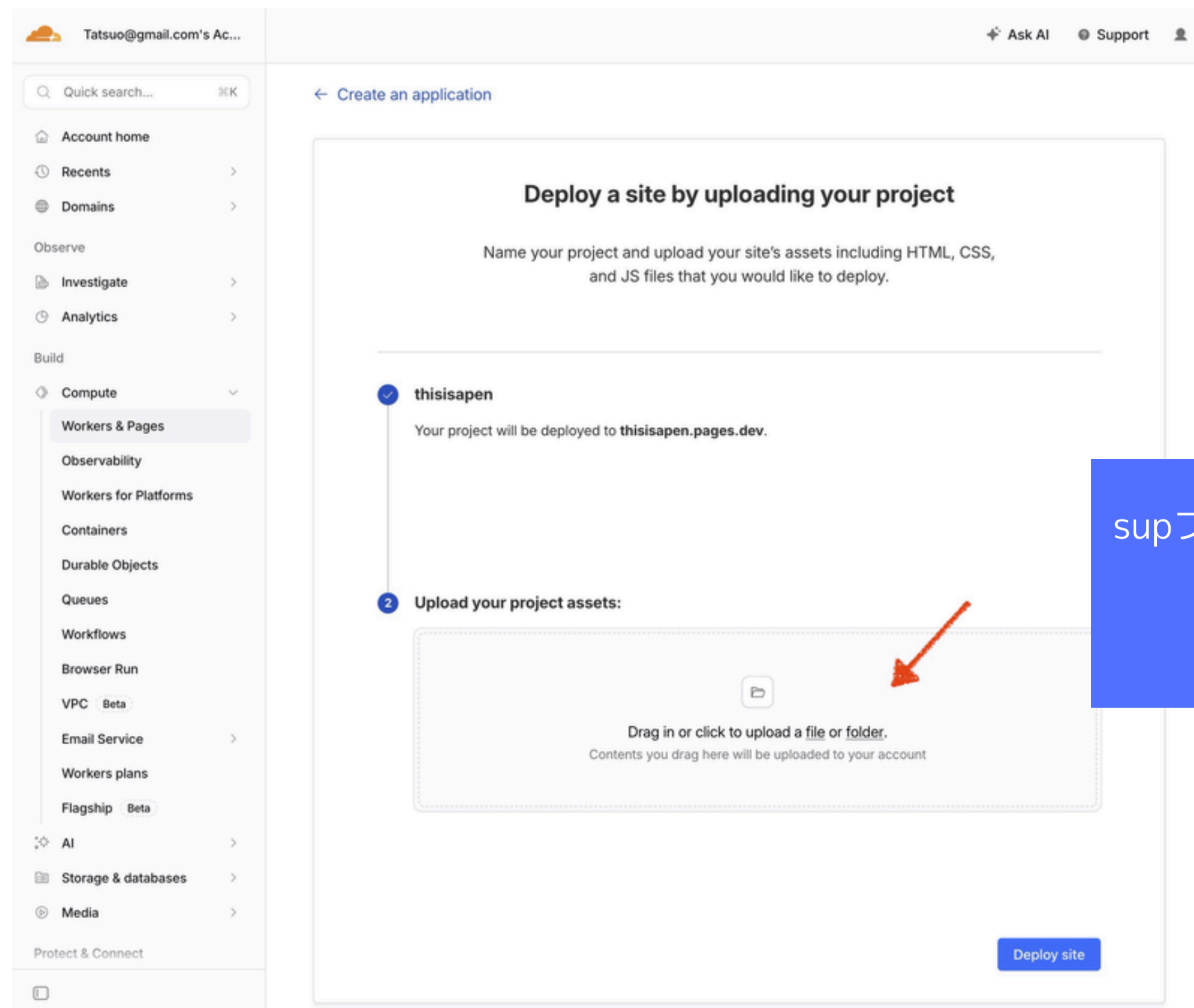
Need a project to upload? Download a demo [Cloudflare Pages Demo](#)

公開URL

<プロジェクト名>.pages.dev

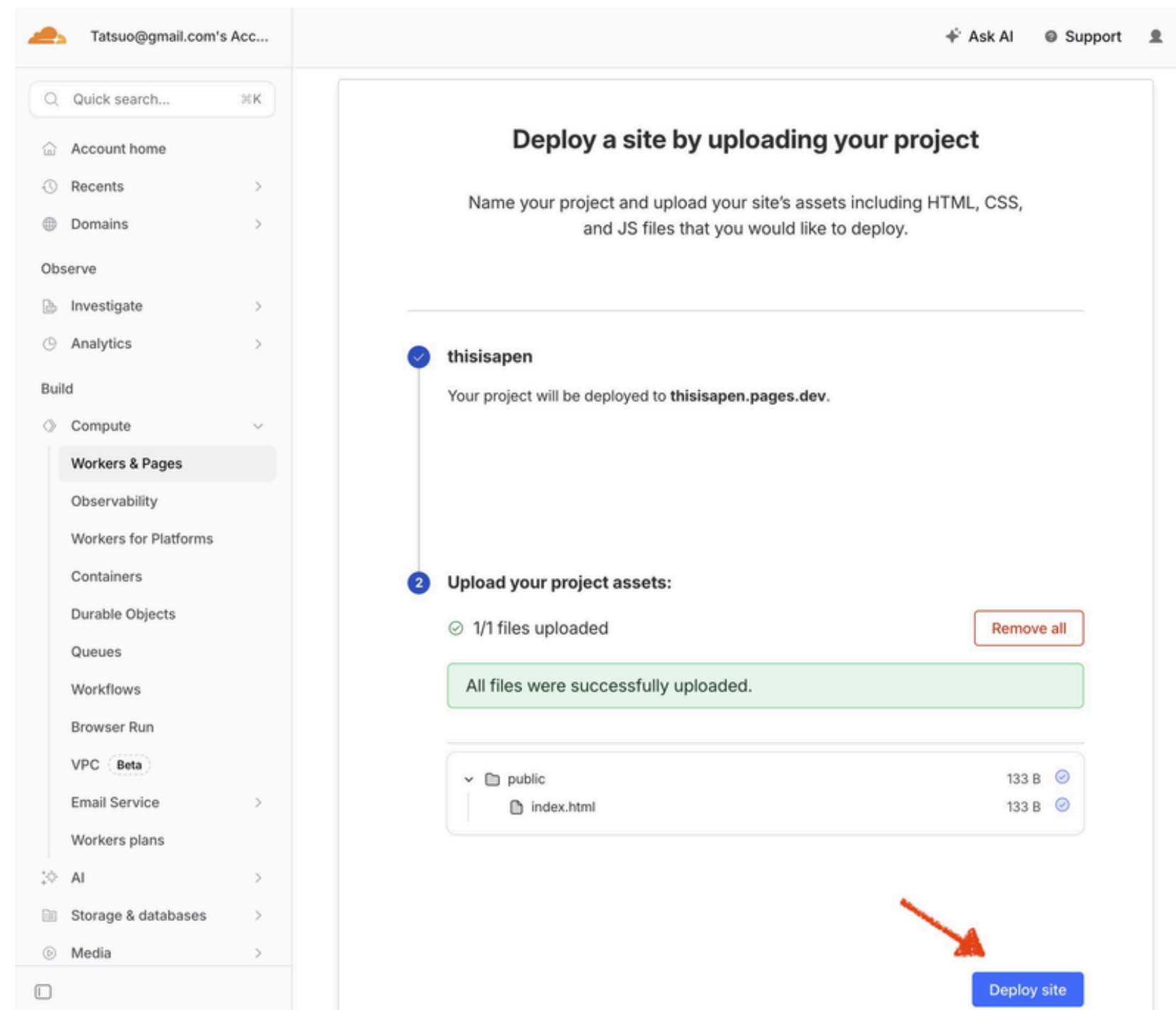
<プロジェクト名>-XXX.pages.dev

Webサイトのデプロイ①

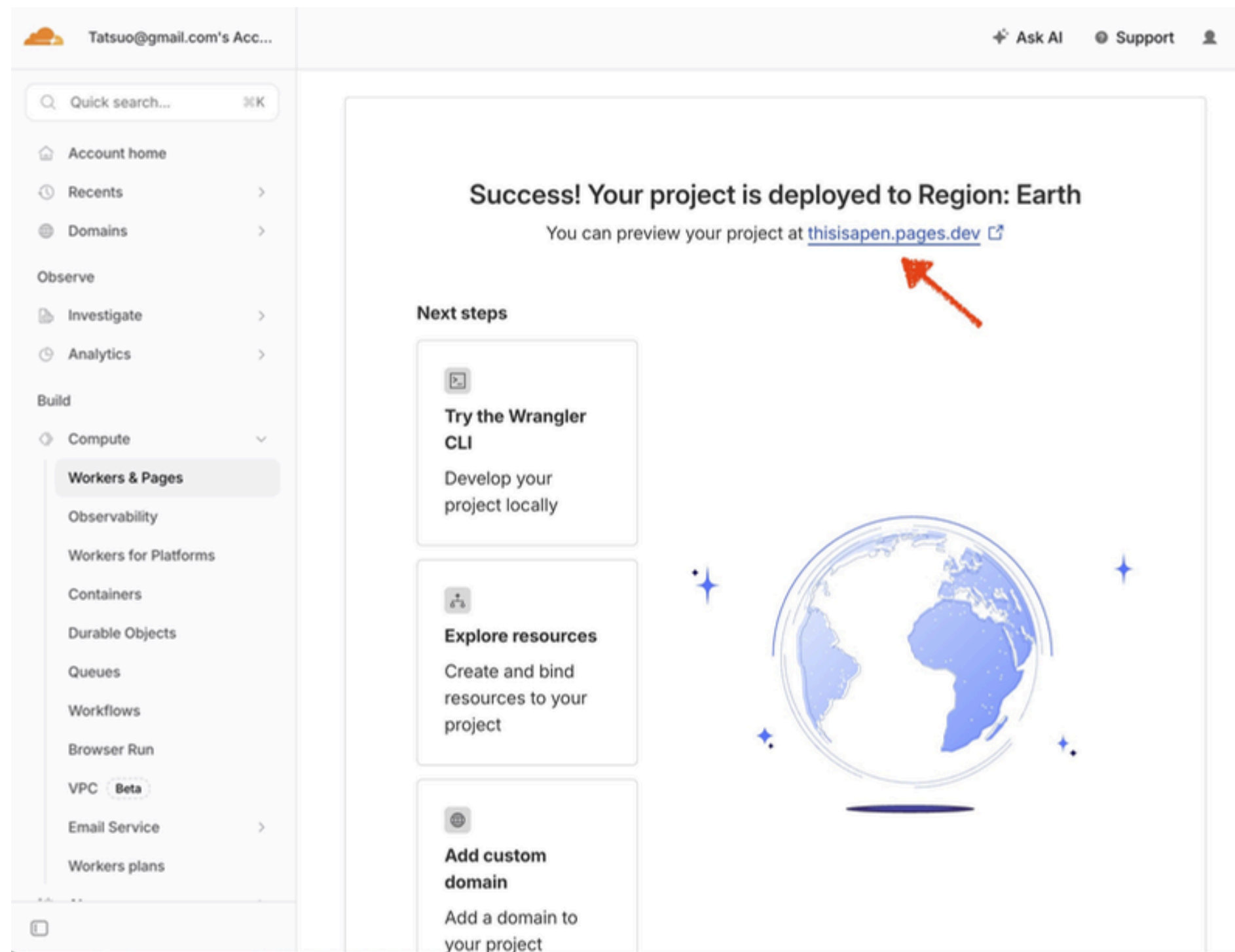


supフォルダの下のpublicフォルダを
ドラッグ&ドロップするか
folderをクリックして選択

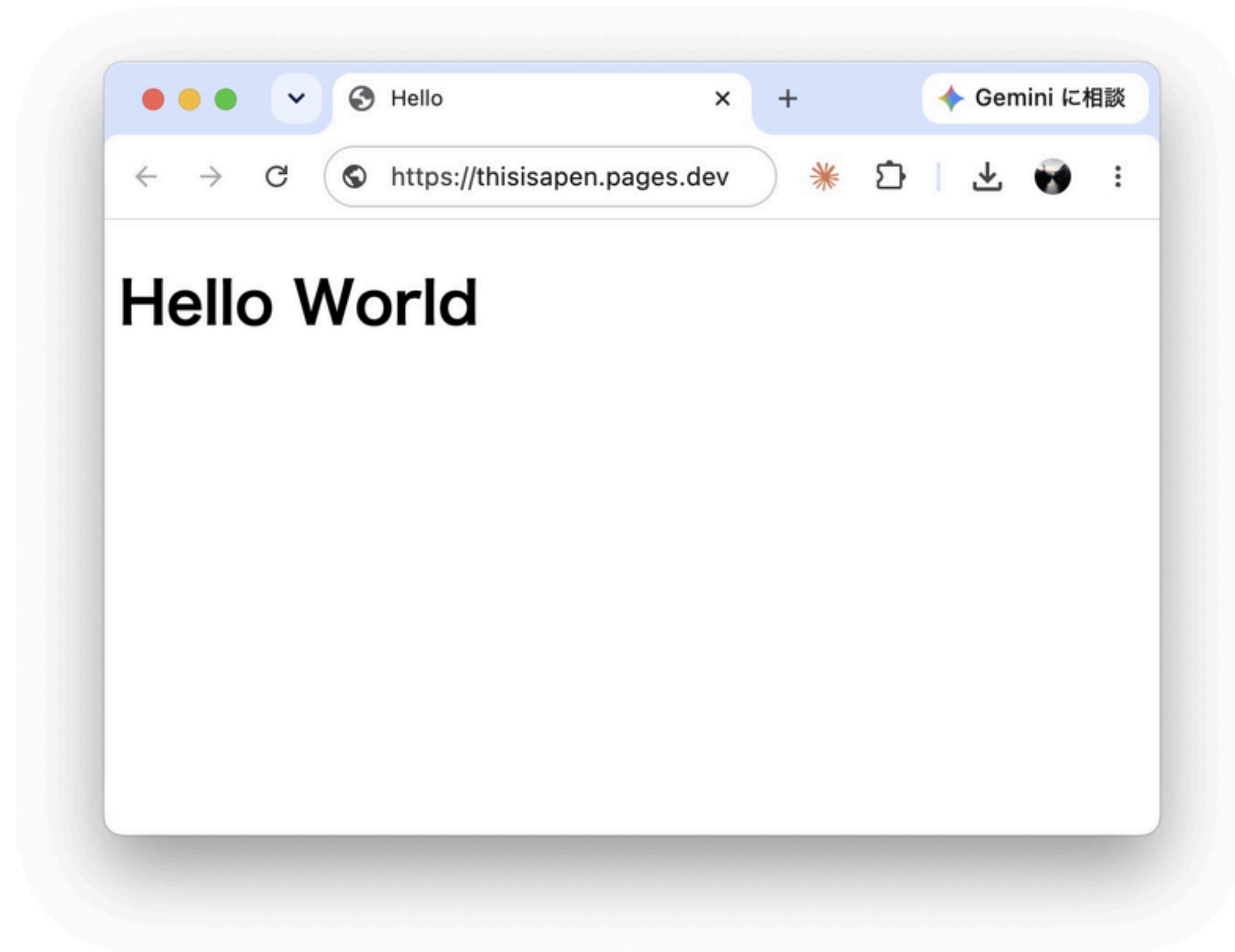
Webサイトのデプロイ②



Webサイトのデプロイ③



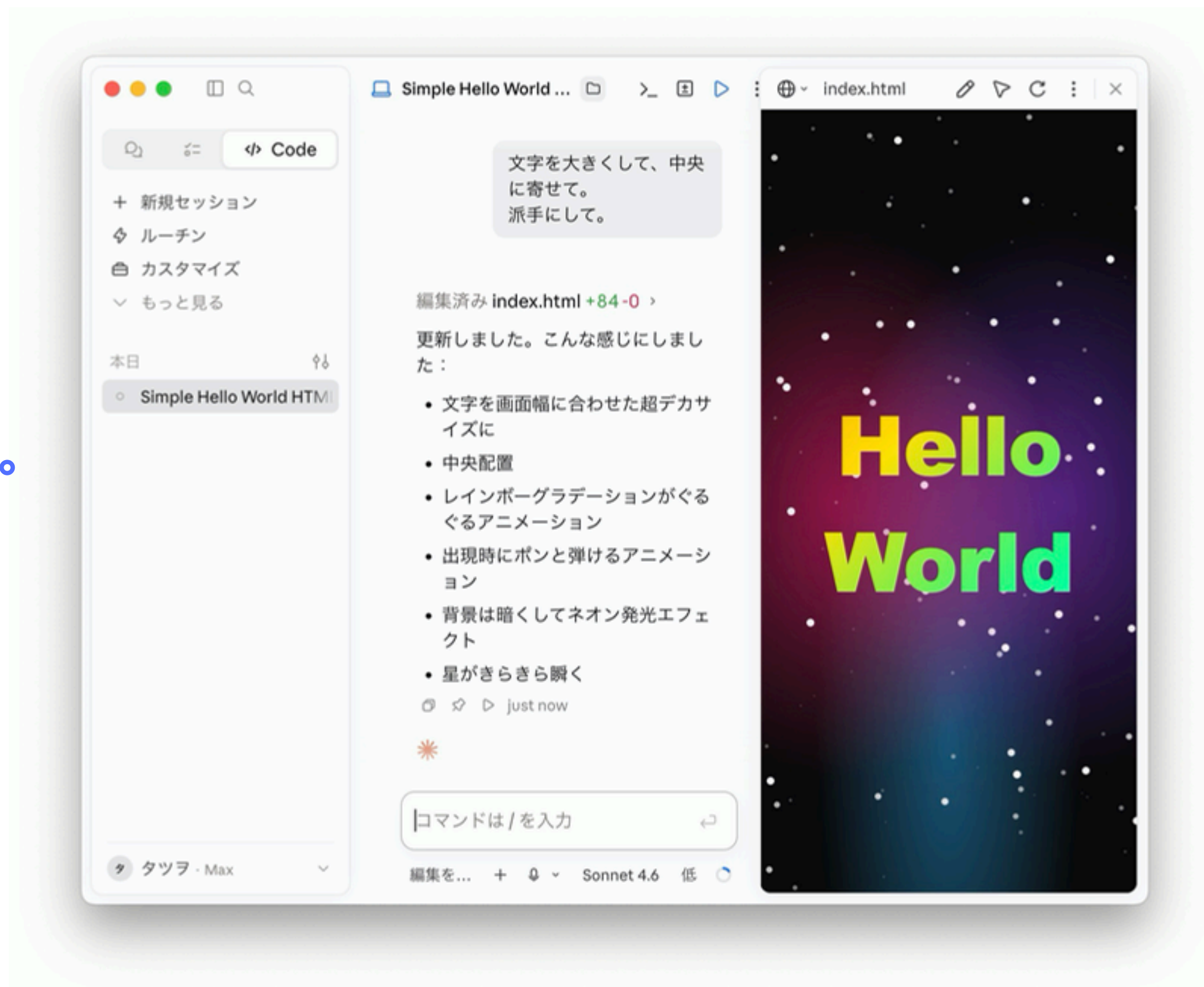
Webサイトのデプロイ④



修正と再デプロイ①

プロンプト：

文字を大きくして、中央に寄せて。
派手にして。



修正と再デプロイ②

The screenshot shows the Cloudflare Workers & Pages dashboard for the account 'Tatsuo@gmail.com's Acc...'. The left sidebar contains a navigation menu with items: Account home, Recents, Domains, Observe, Investigate, Analytics, Build, Compute, Workers & Pages (highlighted), Observability, Workers for Platforms, Containers, Durable Objects, Queues, Workflows, Browser Run, and VPC (Beta). Two red arrows point to the 'Workers & Pages' menu item and the 'thisisapen' application in the list. The main content area displays the 'Workers & Pages' header with 'Documentation' and 'Create application' buttons. Below the header are four summary cards: Requests (4.95k), CPU time (5,104 ms), Observability events (0), and Workers build mins (6). A search bar and filters are present above the application list. The application list shows three entries: 'thisisapen' (thisisapen.pages.dev, 2m ago), 'helloyto' (helloyto.pages.dev, 7m ago), and 'vibecoding' (vibecoding-1zg.pages.dev + 1 other domain, 27m ago). Each entry has a 'No Git connection' status.

Application	Domain	Last Modified
thisisapen	thisisapen.pages.dev	2m ago
helloyto	helloyto.pages.dev	7m ago
vibecoding	vibecoding-1zg.pages.dev + 1 other domain	27m ago

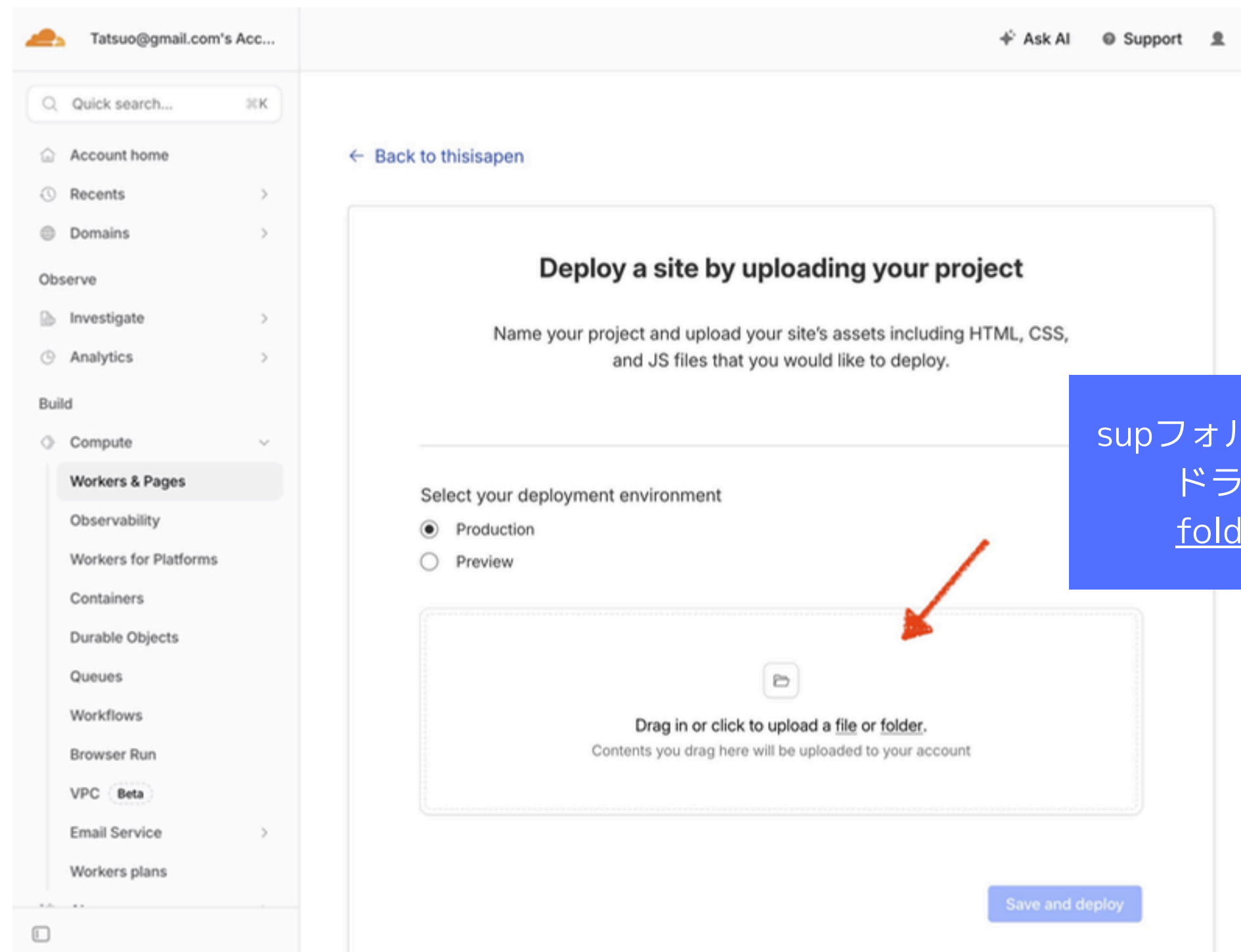
修正と再デプロイ③

The screenshot shows the Cloudflare Workers & Pages interface. The left sidebar contains navigation links: Account home, Recents, Domains, Observe, Investigate, Analytics, Build, Compute, Workers & Pages (selected), Observability, and Workers for Platforms. The main content area is titled 'Workers & Pages > thisisapen'. It features tabs for Deployments, Metrics, Custom domains, and Settings. A red arrow points to the 'Create deployment' button. Below the tabs, the 'Production' environment is highlighted, showing the domain 'thisisapen.pages.dev'. A table lists the deployment details:

Environment	Source	Deployment	Status
Production	main	https://61c1e5fc.thisisapen.pages.dev	✓ 15 hours ago

Below this, the 'All deployments' section shows the same deployment details.

修正と再デプロイ④



あとは初回デプロイと同じ

修正と再デプロイ⑤

修正とデプロイを繰り返してみよう！

プロンプト：

Hello Worldの下に日本語訳「こんにちは世界」を足して

→デプロイ

プロンプト：

文字をクリックすると拡大するようにして

→デプロイ

過去のデプロイにロールバックする

サイトを更新して不具合が出たときなどは、過去のデプロイに本番を戻せる（ロールバック）
Deployments の一覧で戻したいデプロイの行の右端の「…」 → Rollback to this deployment

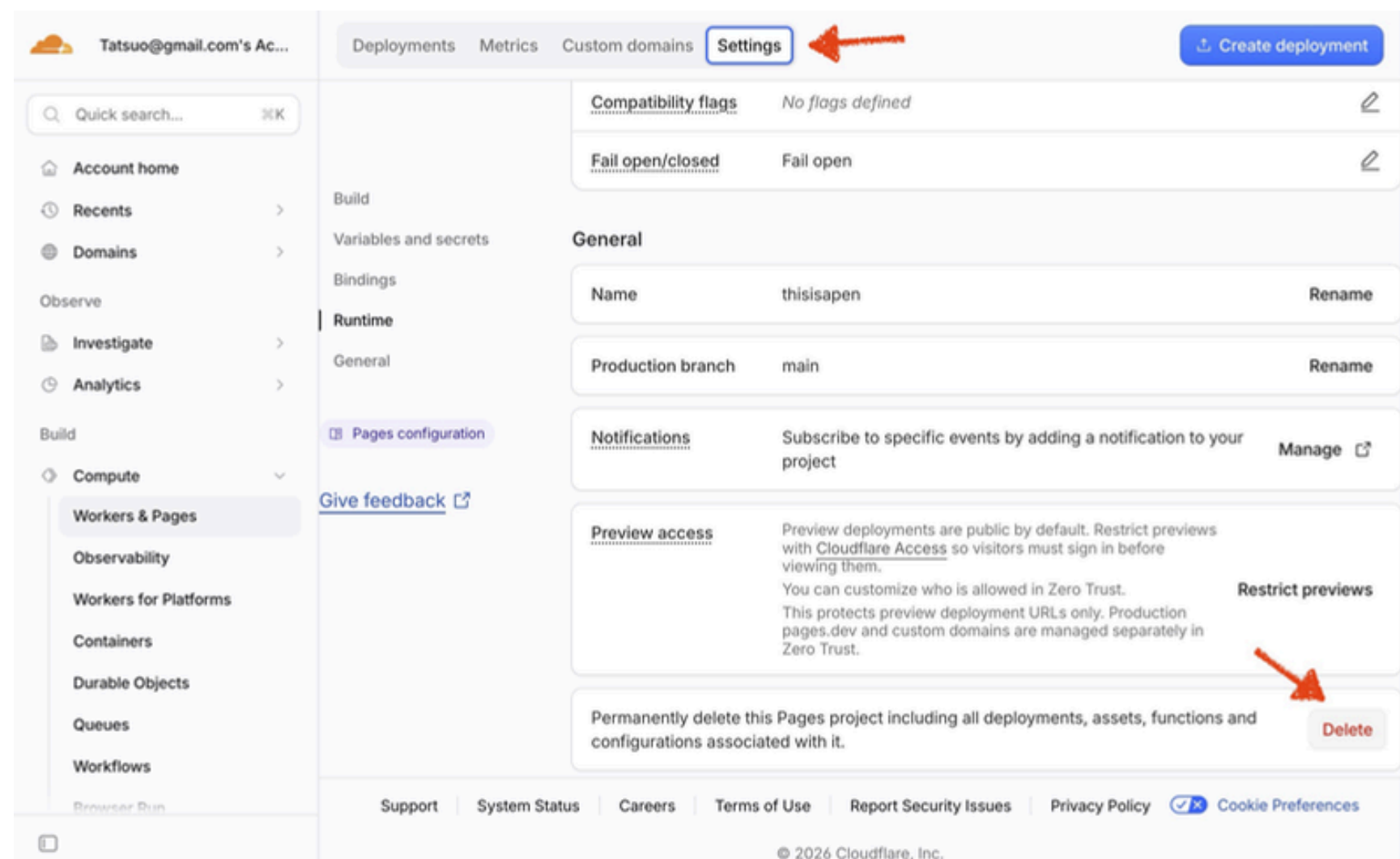
All deployments						
Environment	Source	Deployment	Status			
Production	main	https://635c3404.tokyorain.pages.dev	✓ 5 days ago	Details		...
Production	main	https://716141e8.tokyorain.pages.dev	✓ 5 days ago	Details		...
Production	main	https://73fc31b6.tokyorain.pages.dev	✓ 5 days ago	Details		...
Production	main	https://9c3fa779.tokyorain.pages.dev	✓ 5 days ago	Details		...

Rollback to this deployment

Delete deployment

プロジェクトを削除する方法

プロジェクトページの Settings を開き、一番下までスクロールして Delete（削除）
削除すると プロジェクト名.pages.dev のサイトも公開停止になる



いろいろ作れます



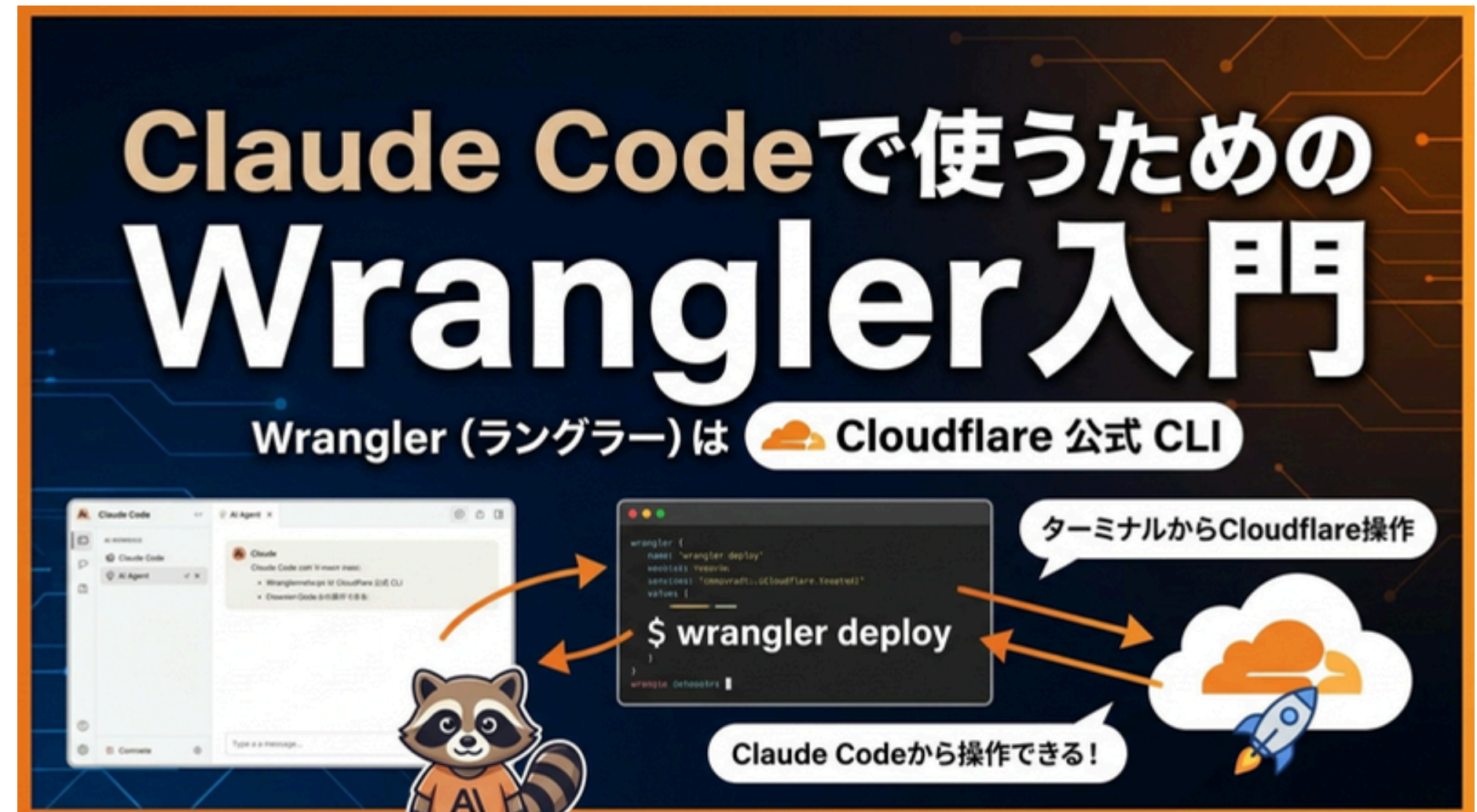
詳細は：[CloudflareにHTMLファイルなどをブラウザでアップロードしてWebサイトとして公開する | Vibecoding Notes](#)

初級 Wrangler でデプロイ

[WRG] <https://vibecodingnotes.com/handson-cloudflare-wrangler/>

Wrangler とは

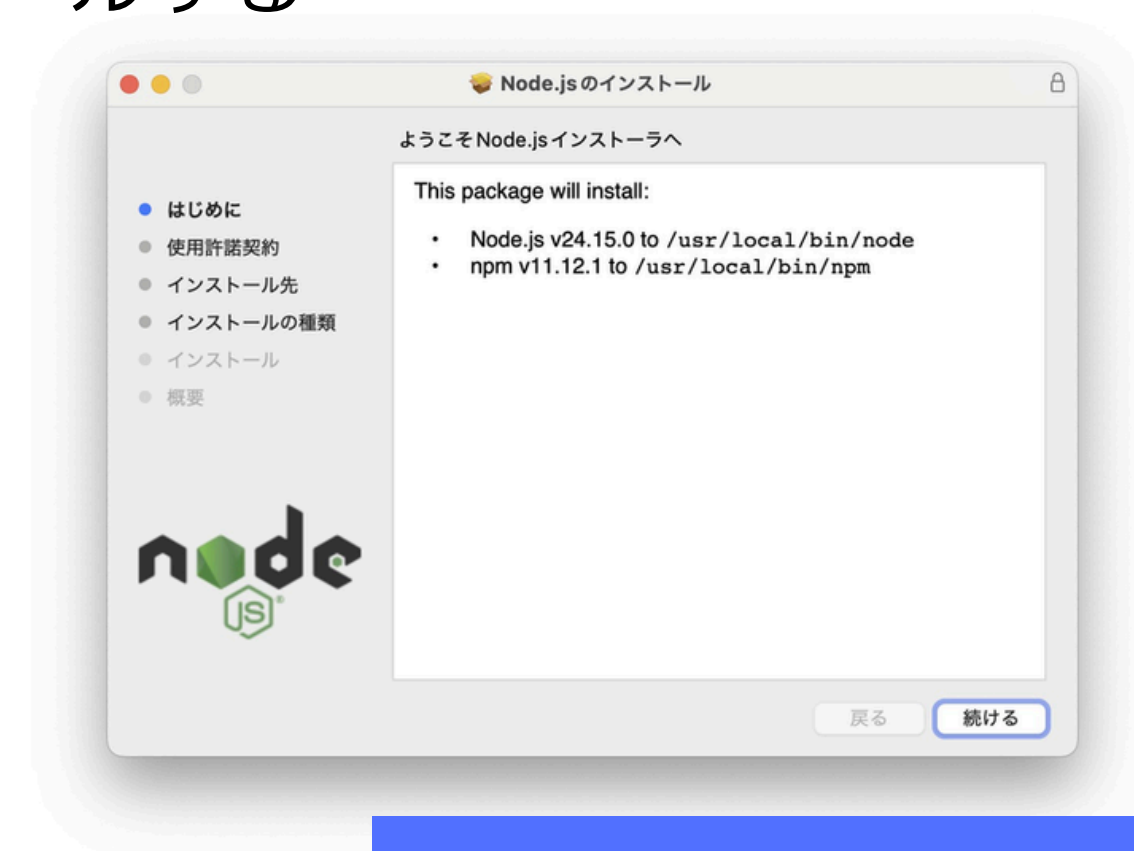
- Wrangler（ラングラー）とは Cloudflare が公式に提供する コマンドラインツール（CLI）
- ターミナルから Cloudflare の 操作（デプロイなど）ができる
- ターミナルから操作できるということは、Claude Codeから操作できるということ



Node.js のインストール

Wrangler を使うためには Node.js が必要！

Node.js の公式サイト（<https://nodejs.org/ja>）からインストーラー（Mac/Windows）をダウンロードしてインストールする



インストール後、Claude デスクトップアプリが起動している状態だったら再起動すること

Wrangler で Cloudflare にログイン

Wrangler から Cloudflare を操作するには、Wrangler でのログインが必要

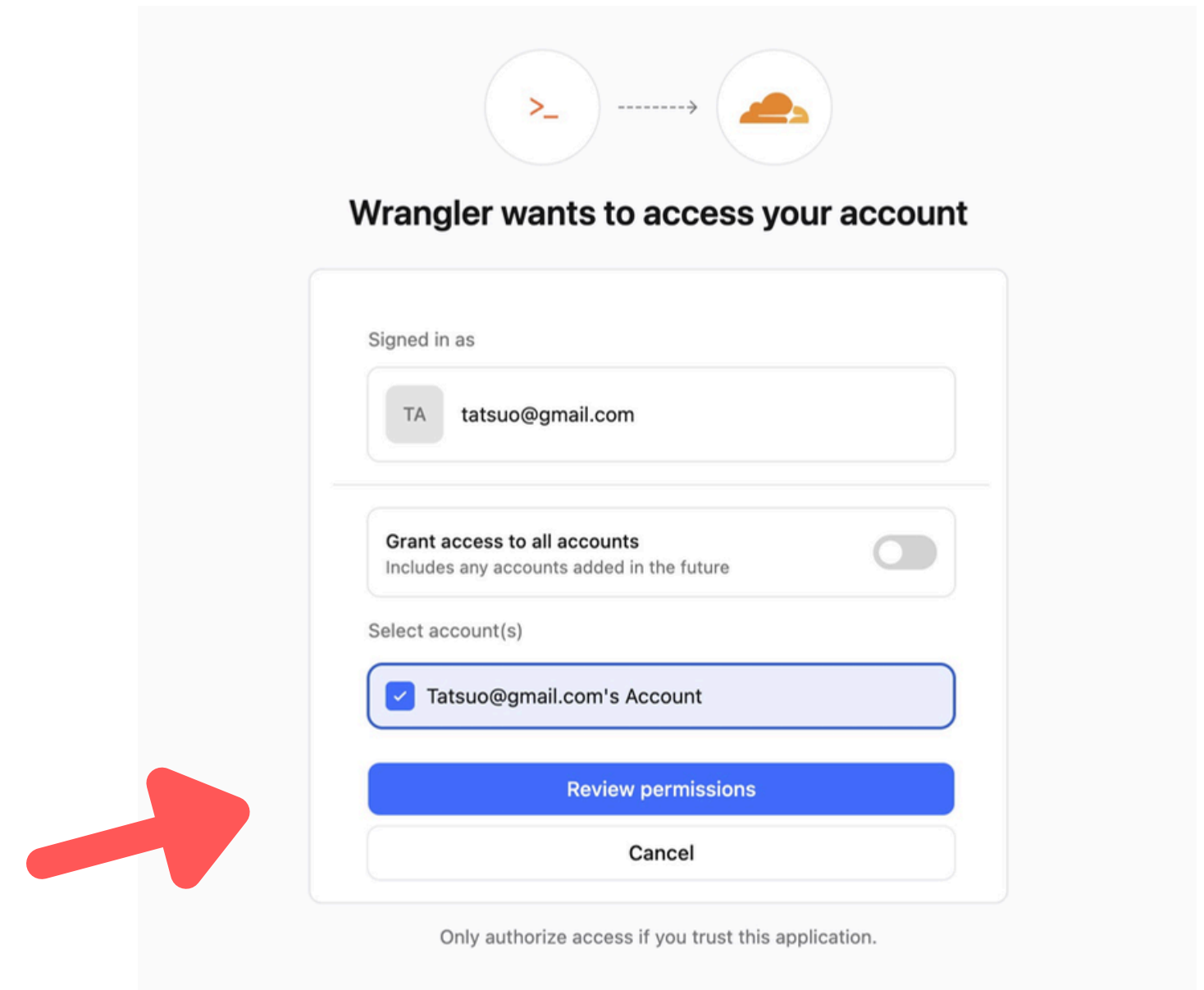
1. ターミナルを開く

- Mac: Cmd + スペース → terminal と検索して起動
- Win: スタートメニューで cmd と検索して起動

2. ログインコマンド (↓) をそこで実行

- **npx wrangler login**

3. ブラウザで承認画面が開くので進めていく



デプロイするWebサイトデータ

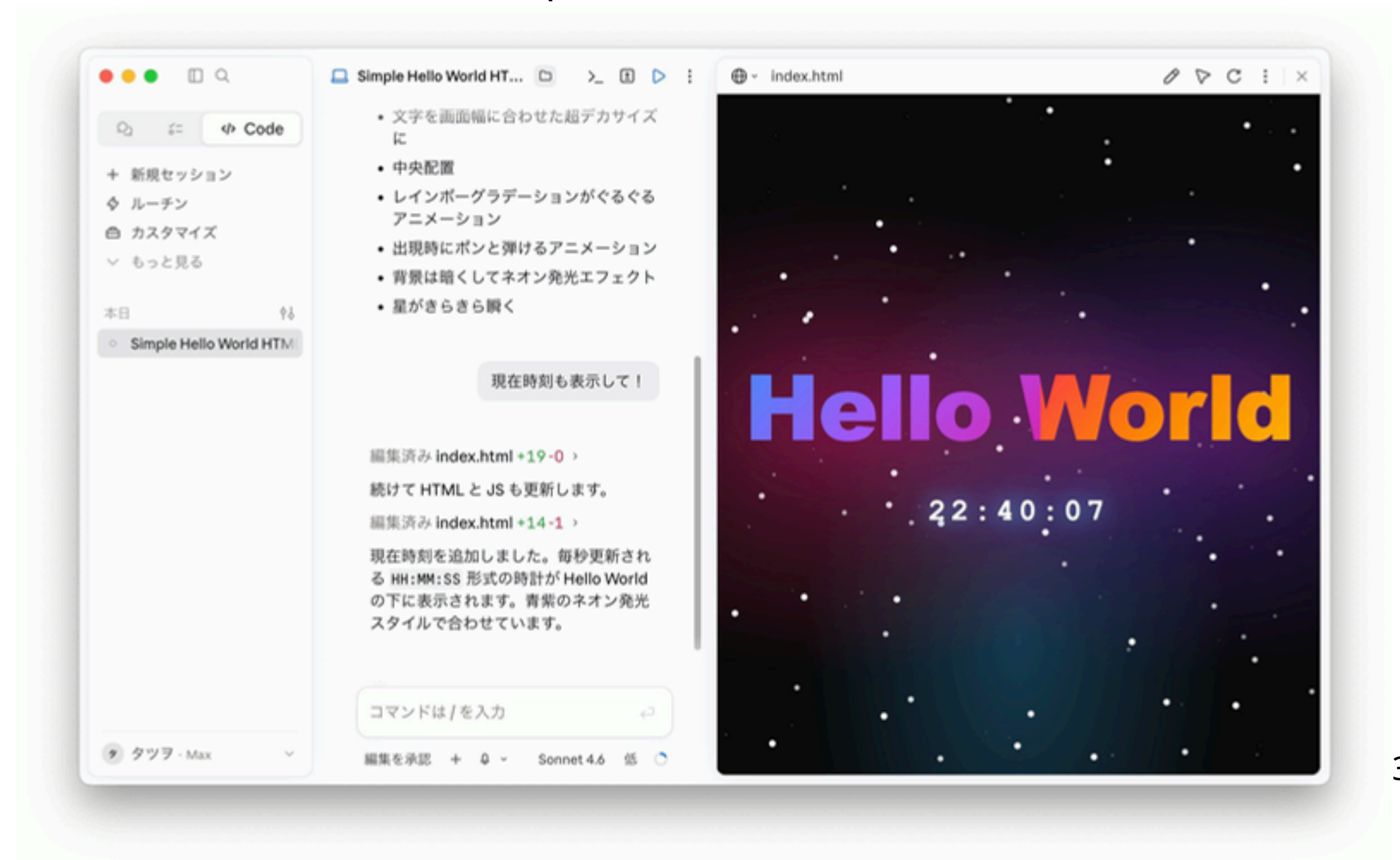
Claudeデスクトップアプリのさっきのセッション（supフォルダ）。

派手な Hello World が
表示されるWebサイト。
これを修正して、
Wrangler で再デプロイする。

まず修正する。

プロンプト：

現在時刻も表示して！



デプロイ先のプロジェクト名は？

さっきブラウザ経由でデプロイしたときのプロジェクト名を覚えていますか？
それが分かれば、Wrangler からデプロイできます。

忘れてたら Cloudflare のダッシュボードで確認しましょう
<プロジェクト名>.pages.dev / <プロジェクト名>-XXX.pages.devのように
URL の一部になってます。

手作業でブラウザからデプロイしたプロジェクトなので、Claude Code も
Wrangler もプロジェクト名を知りようがないのですよ。

Wrangler でデプロイ（初）

プロンプト：

public フォルダの中身を、Cloudflare Pages のプロジェクト「プロジェクト名」に、本番（main ブランチ）として wrangler でデプロイして

プロジェクト名は自分のに置き換える：

public フォルダの中身を、Cloudflare Pages のプロジェクト「thisisapen」に、本番（main ブランチ）として wrangler でデプロイして

（裏で動くコマンド：npx wrangler pages deploy ./public --branch=main --project-name=プロジェクト名）

デプロイ完了したらブラウザで確認してみよう！

デプロイが禁止されている場合

すでに ~/.claude/CLAUDE.md にコミュニティで配られた開発ルールを入れている人は、ここで止まることがあります。

「エンジニアに相談してください」のように言われて、先に進めなかったり。

不具合ではありません。自分で入れたルールが効いているだけです。

参考：[公開するWebアプリの地雷を避ける：Claude に安全ルールを守らせるスキルを入れる | Vibecoding Notes](#)



心当たりのある人は
私に聞いてね！

Mac: `open -e ~/.claude/CLAUDE.md`

Windows: `notepad %USERPROFILE%\.claude\CLAUDE.md`

修正して再デプロイ

プロンプト：
流れ星も出して

プロンプト：
デプロイして

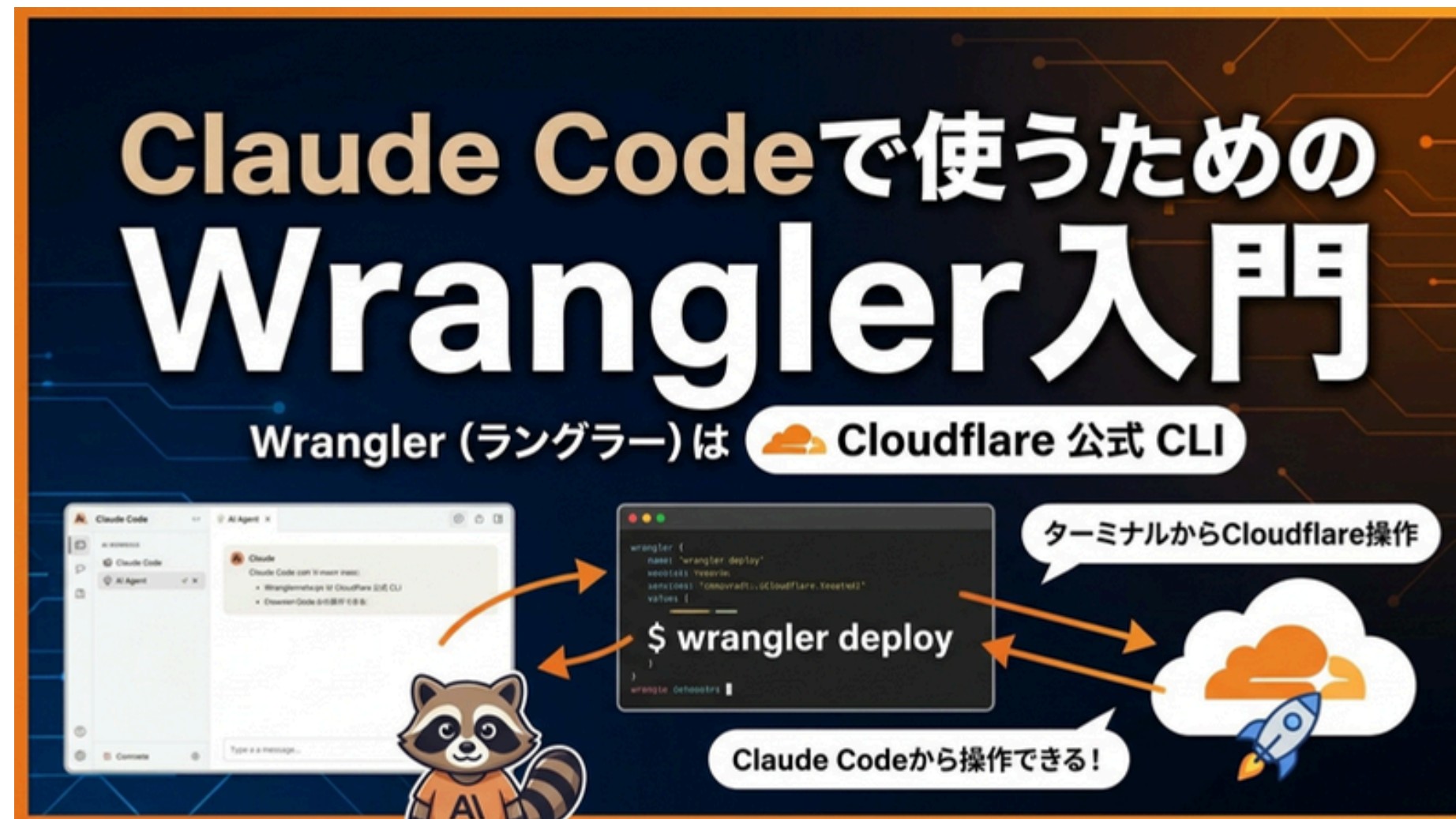
↑これでさっきのWranglerコマンドを動かしてくれる

デプロイ完了したらブラウザで確認してみよう！



祝 クロコだけでデプロイできた！

Wrangler を使って、Claude Code だけで Web サイトを Cloudflare にデプロイすることができました。おめでとうございます🎉



クロコだけで作成から公開まで

ここまでは、ブラウザで作ったプロジェクトを Wrangler で更新してきた

Wrangler はプロジェクトの作成もできる → クロコだけで、サイト作成から公開まで全部できる

- (1) 作業フォルダを作って、Claude デスクトップアプリで開く
- (2) 「〇〇を作って」でサイトを作ってもらう
- (3) 「公開して」で公開してもらう

<(3) の依頼文>

public フォルダの中身を Cloudflare Workers に「XXXXXXXXXX」という名前で公開して。

wrangler の設定ファイル (wrangler.jsonc) も作ってほしい。

- ・ 公開ディレクトリは public
- ・ compatibility_date は UTC (協定世界時) での今日の日付にする

Pages と Workers

2026年9月上旬から、Wrangler で新しく作ると Workers になる。
どちらでも同じように公開できる。無料なのも同じ。
今あるプロジェクトの更新は、これまでどおり Pages のまま。

Pages

- これまでにブラウザにドラッグ＆ドロップして作ったやつ
- URL が短い <名前>.pages.dev

Workers

- もともとプログラムを動かす場所。いまは静的サイトも置ける
- URL は少し長い <名前>.<アカウント名>.workers.dev
- Cloudflare は「新しく作るならこっち」と案内している



やってみよう！ Webアプリ作成

HTML+JavaScriptでポモドーロタイマーを作って！ public/index.html 1ファイルで作って。

複利計算機をHTML+JavaScriptで作って！ 構成はHTMLファイル(public/index.html)が1つ。

マウスかタッチパッドでボタンを10回連打するのに何秒かかるかを競うゲーム。
public/index.html 1ファイルで作って。

ボタンを押すと効果音が出るWebアプリをHTMLファイル1つ（public/index.html）で作って。
3つのボタンで三つの効果音。

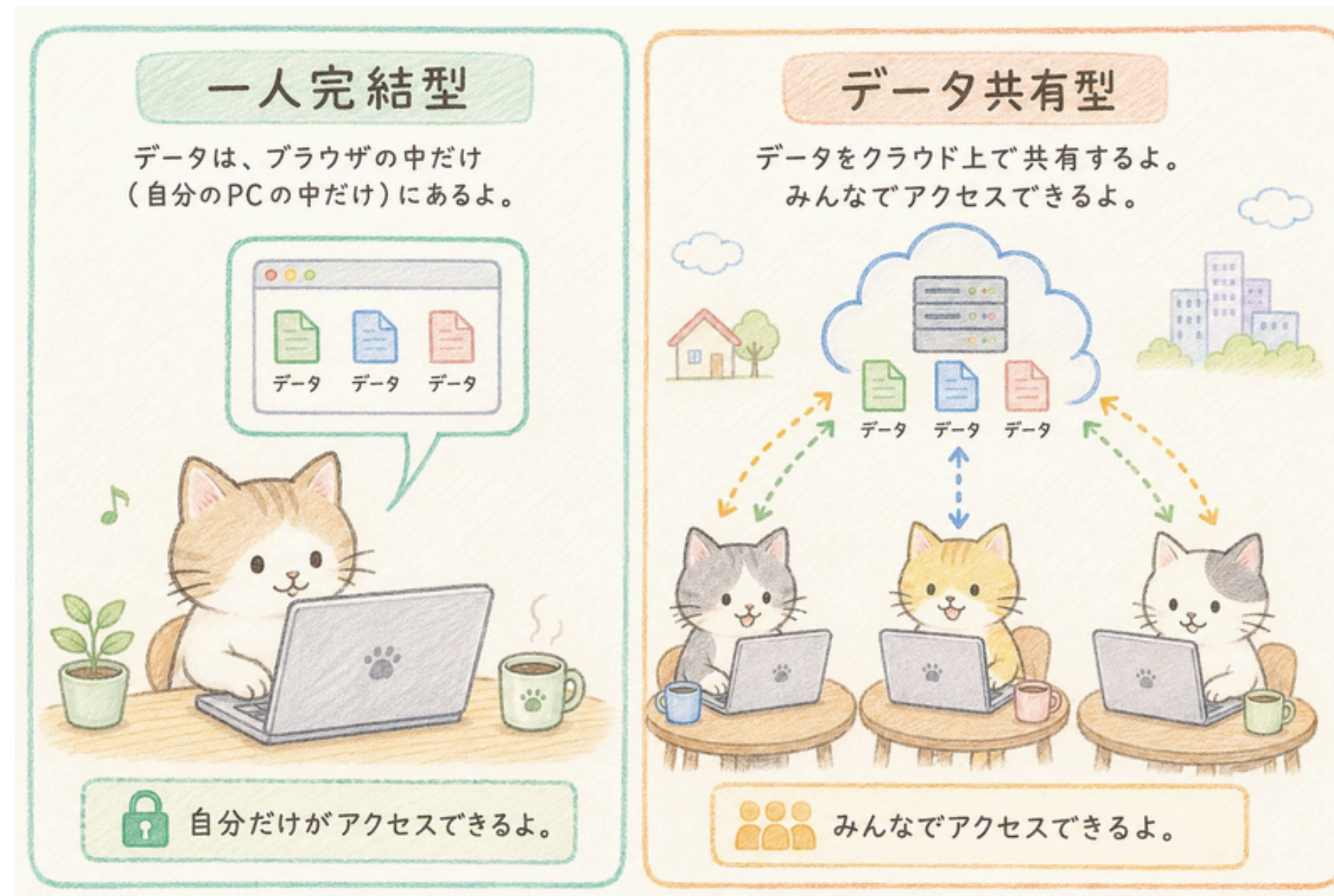
TODOリストをHTML+JavaScriptで作って！ 項目の追加・削除ができて、ページを閉じてでも消えないようにしたい。
public/index.htmlに書いて。ファイルは1つで。

最大60秒のカウントダウンタイマーをHTML+JavaScriptで作って！
秒数を入力してスタート・リセットできるようにしたい。HTMLファイル1つ（public/index.html）にまとめて。

初級 Webアプリの2つのタイプ (前篇)

[TWA] <https://vibecodingnotes.com/guide-webapp-types/>

Webアプリの2つのタイプ



一人完結型とデータ共有型

一人完結型アプリ

自己紹介サイト・ポートフォリオ・LP
スクリーンセーバー
複利計算・年金計算・投資推移予測
タイマー・ストップウォッチ・時報
QRコード生成
マインドマップ
...

データ共有型アプリ

掲示板・チャット
カレンダー・予約・シフト管理
作品投稿・レビュー
ランキングゲーム
アンケート・投票
リアルタイム対戦
...

一人完結型からデータ共有型へ

同じ「共有する」でも、行き先は違う。

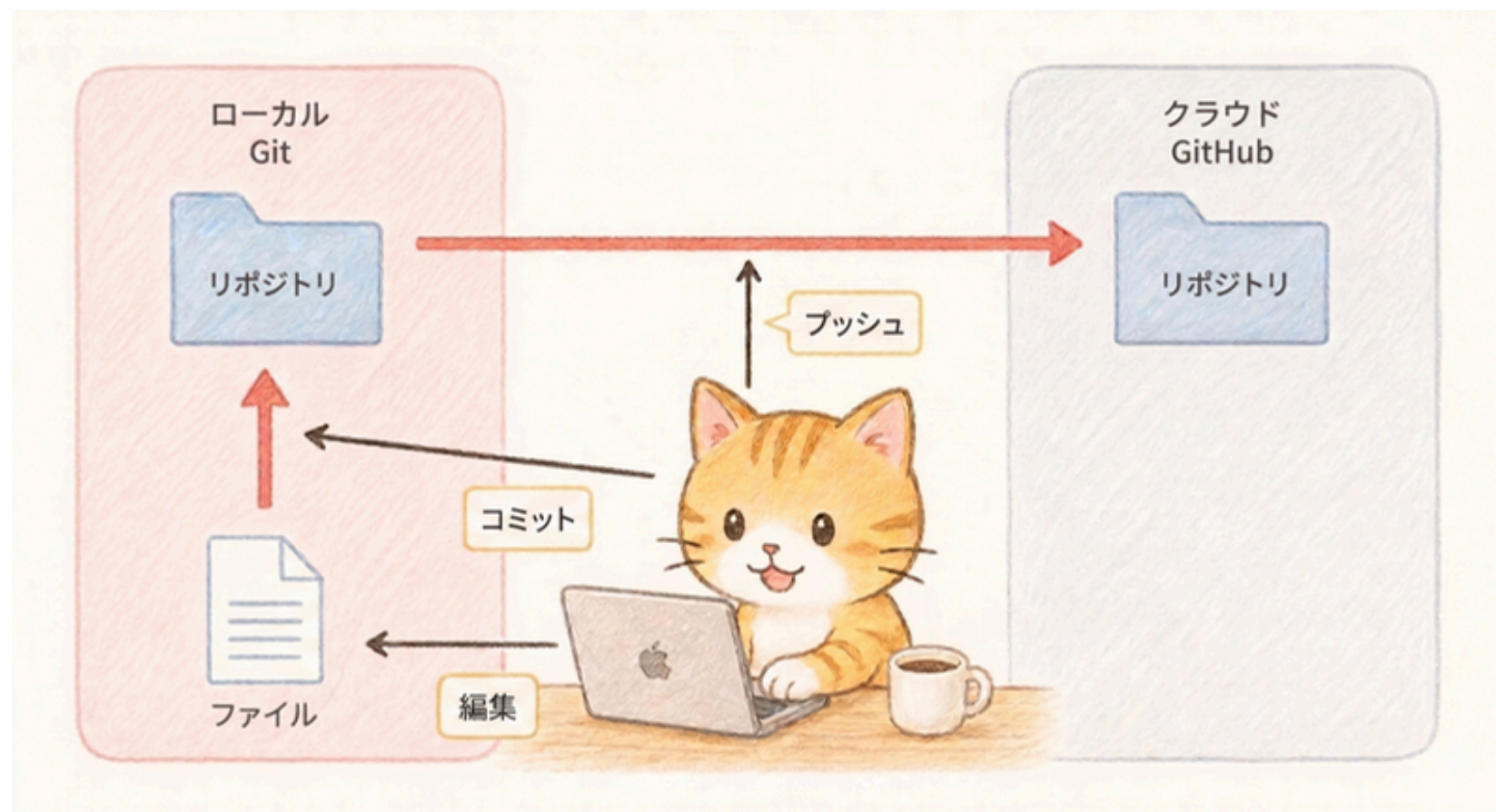
- TODO → チームTODO
- 家計簿 → 共有家計簿
- オセロ（コンピュータと対戦） → ネット対戦のオセロ
- スコアアタック → ランキング
- ホワイトボード → 共同ホワイトボード

掲示板・アンケート・リアルタイム対戦は、相手がいて初めて成立する。
一人完結型の形がなく、最初からデータ共有型で作ることになる。

開発基礎

Git と GitHub

[GBA] <https://vibecodingnotes.com/guide-git-basic/>



Git (コマンド)

- ローカルPCで履歴管理
- 管理用フォルダを**リポジトリ**と呼ぶ
- 履歴の一区切り（スナップショット）を**コミット**（commit）と呼ぶ
 - 動詞用法あり「コミットする」
- コミットはリポジトリに保存
- ローカルで使う場合はアカウント不要

GitHub (Webサービス)

- コミットをネット上に保存する場所
- ローカルのリポジトリと同期
- ローカルのコミットをGitHubにアップすることを**プッシュ**（push）と呼ぶ
- 利用にはGitHubのアカウントが必要

※ざっくりな説明なので厳密にはちょっと違うこともあり

Claude Code と Cloudflare で Git/GitHub を 使う理由 4選



思い切って変更を頼める

Git で変更履歴を残しておけば、失敗しても元に戻せる。
Claude Code に大きめの修正を頼みやすくなる。

何を変えたか追える

どのファイルを、いつ、どう変更したかを確認できる。
AIに任せた変更でも、あとから内容を見直せる。

バックアップ・移行・引き継ぎに強い

GitHub がコードの保管場所になる。
別のMacへの移行や、誰かへの引き継ぎもしやすくなる。

Cloudflare との相性がいい

GitHub と Cloudflare を連携すると、GitHub にプッシュ
するだけで本番サイトに反映できる。

Git for Windows のインストール (Windows)

Mac には git が最初から入っているが
Windows には入っていないので、ここに入れる (過去に入れた人は不要)

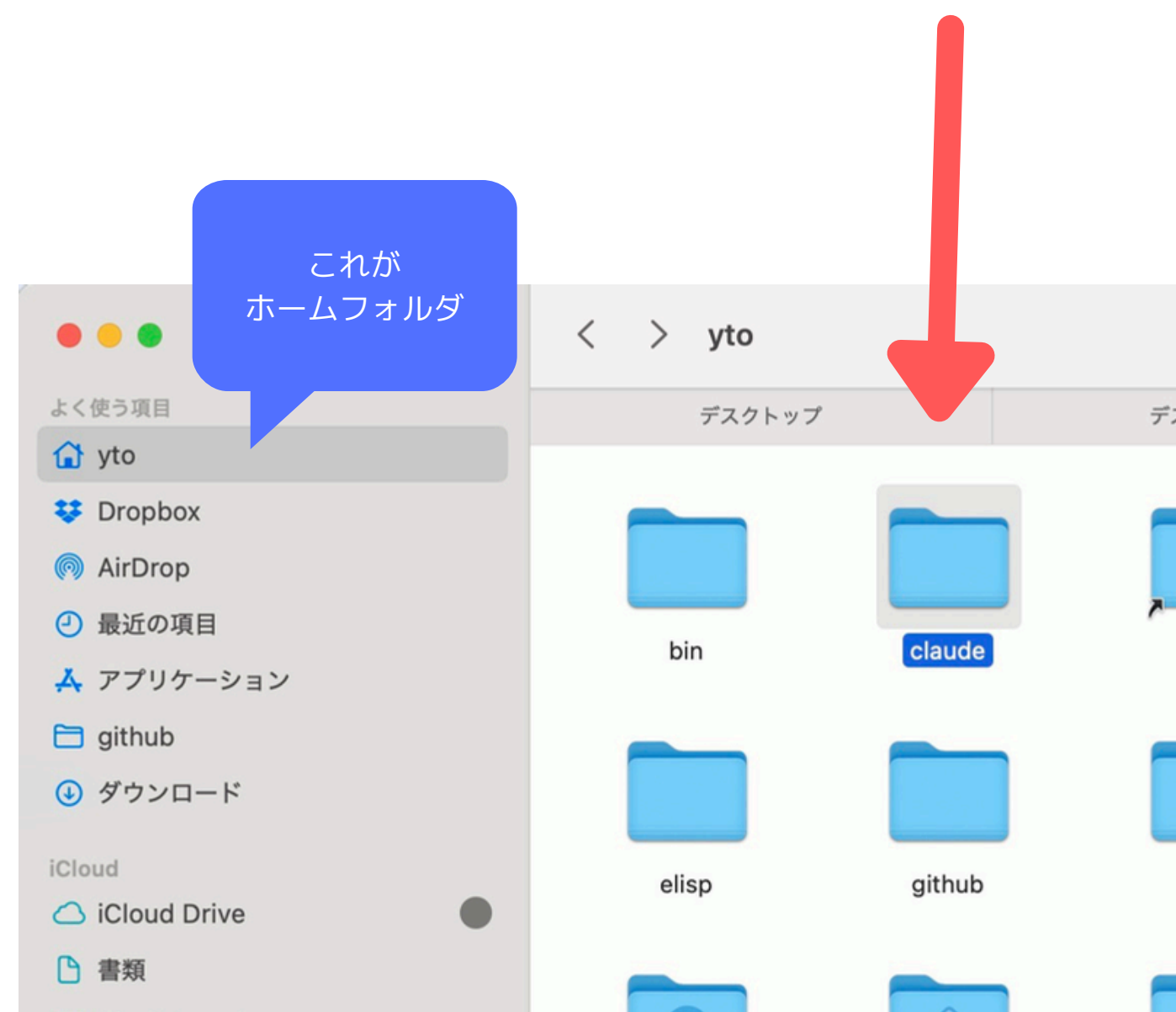
<https://git-scm.com/downloads/win>

インストーラーではすべて既定のまま Next でよい

入れ終わったら Claude デスクトップアプリを完全に終了して起動し直す
(Windows 本体の再起動は不要)

練習用フォルダ作成 & クロコ起動

Finderで**ホームフォルダ**の **claude** フォルダに、Webサイト用フォルダ **git-test** を作る



- Claudeデスクトップアプリを起動
- 左上の“Code”を選択
- 新規
- 「ローカル」と書かれたボタンの横のフォルダアイコン → フォルダを開く
- 作業フォルダ（ホーム→claude→git-test）を選択

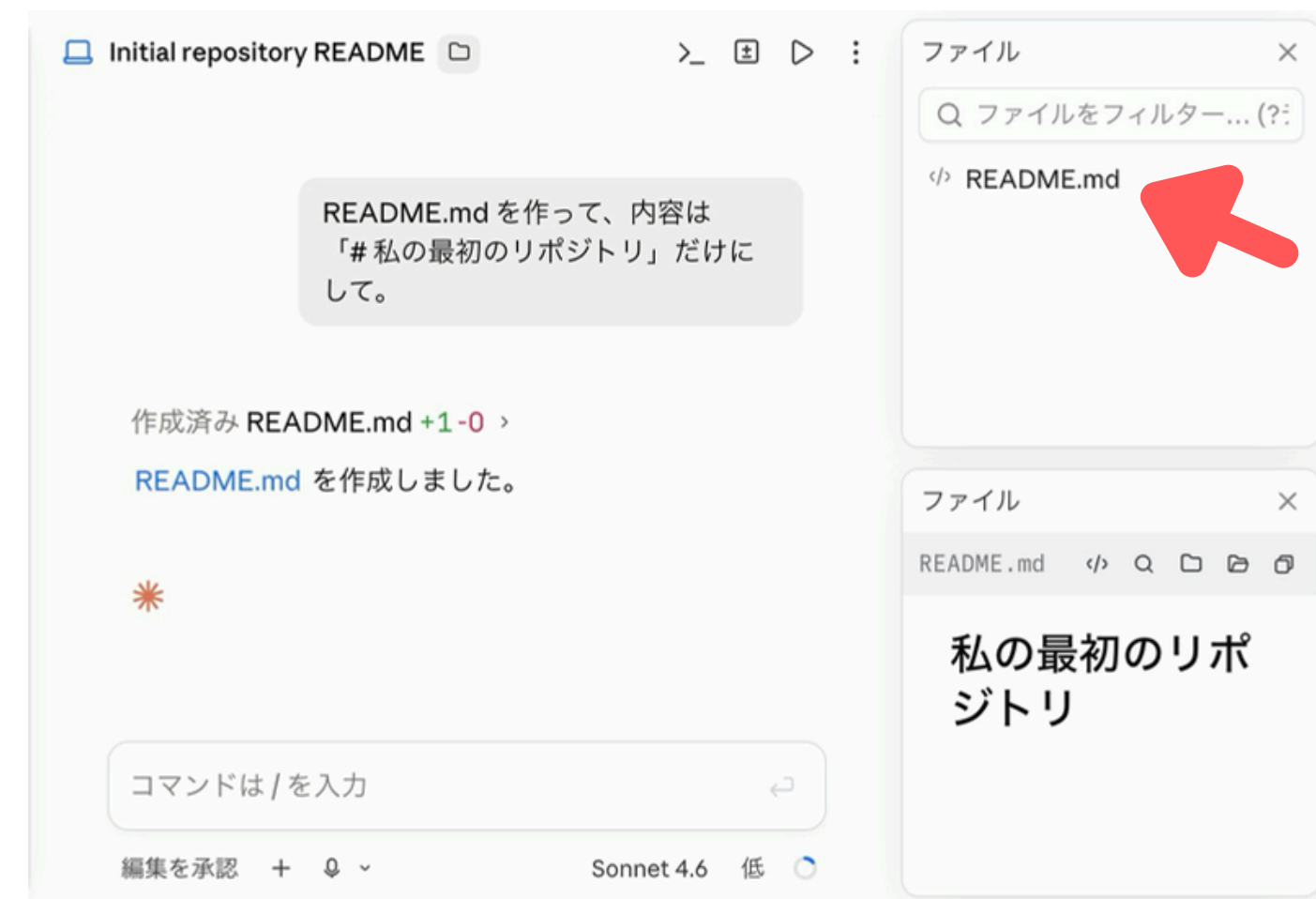
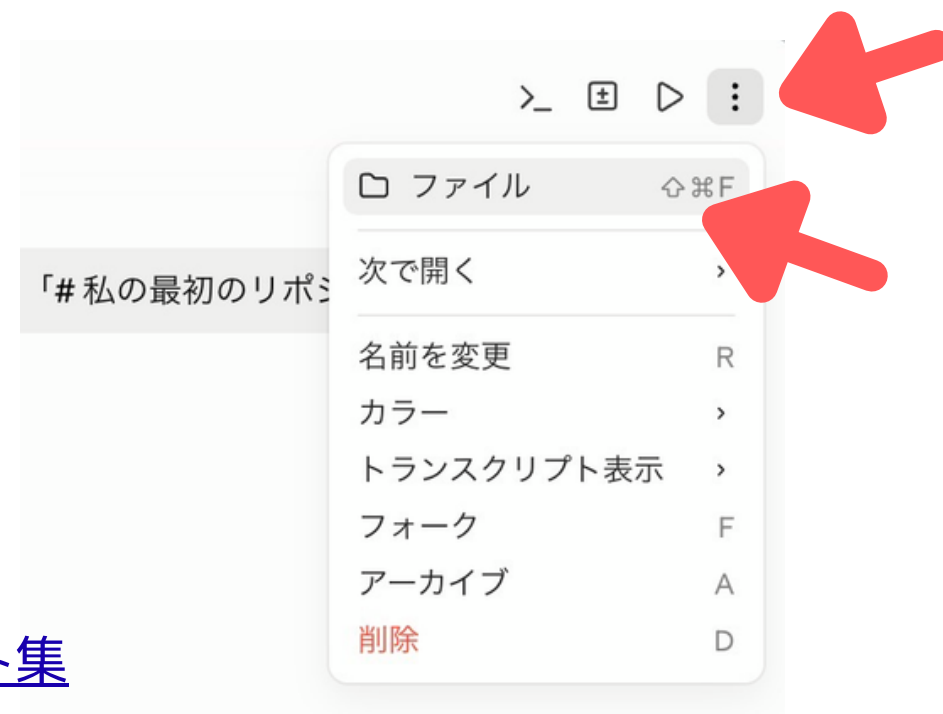


練習用のファイルを作る

プロンプト：

README.md を作って、内容は「# 私の最初のリポジトリ」だけにして。

右上の「⋮」→「ファイル」で確認！



名前とメールアドレスを登録

Windowsではコミットの際に必要となることがある。
本名じゃなくてよい。

プロンプト：

```
git config --global user.name "Your Name"
```

```
git config --global user.email "you@example.com"
```


リポジトリの準備と初回コミット

リポジトリの初期化（これからGitを使うよという宣言のようなもの）と
最初のコミット（スナップショット作成&格納）

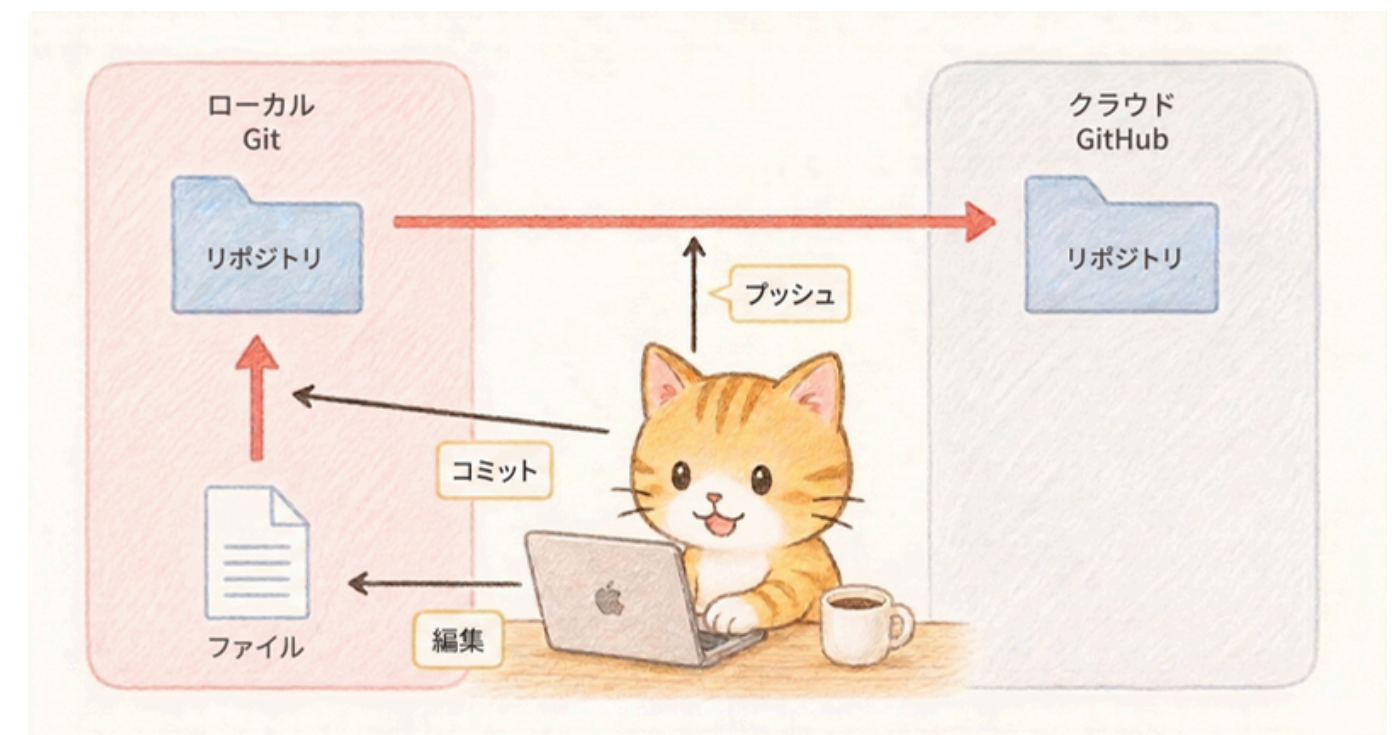
プロンプト：

Git管理始める。

「最初のコミット」としてコミットして。

裏で動くコマンド（リポ作成→ステージング→コミット）

- `git init`
- `git add README.md`
- `git commit -m "最初のコミット"`



まずはローカル Git から

練習なのでどんどんコミットする①

プロンプト：

public/index.html として Hello World と表示するだけのシンプルなHTMLを作って。

README.md にもそのことを書いておいて

プロンプト：

コミットして

プロンプト：

public/index.html を派手にして

プロンプト：

コミットして

練習なのでどんどんコミットする②

プロンプト：

Claude Code を題材にした川柳を1つ作って、README.md に足して

プロンプト：

コミット

プロンプト：

README.md の1行目のタイトルに、今日の日付を足して

プロンプト：

commit

変更履歴を見る

プロンプト：

変更履歴を表で見せて

または

`git log`

変更履歴を表で見せて		
   1m ago		
コミット	日時	メッセージ
e2e1816	2026-06-24 09:25:23	README.md のタイトルに日付を追加
5aef81a	2026-06-24 09:25:07	README.md に川柳を追加

過去のことも覚えているアシスタント

プロンプト：

今日やったことを100文字ほどでまとめて

プロンプト：

間違って消しちゃった。戻して

プロンプト：

README.md に川柳を足す前のコミットに戻して

プロンプト：

戻したのを取り消して

プロンプト：

派手にする前と今で、何が変わった？

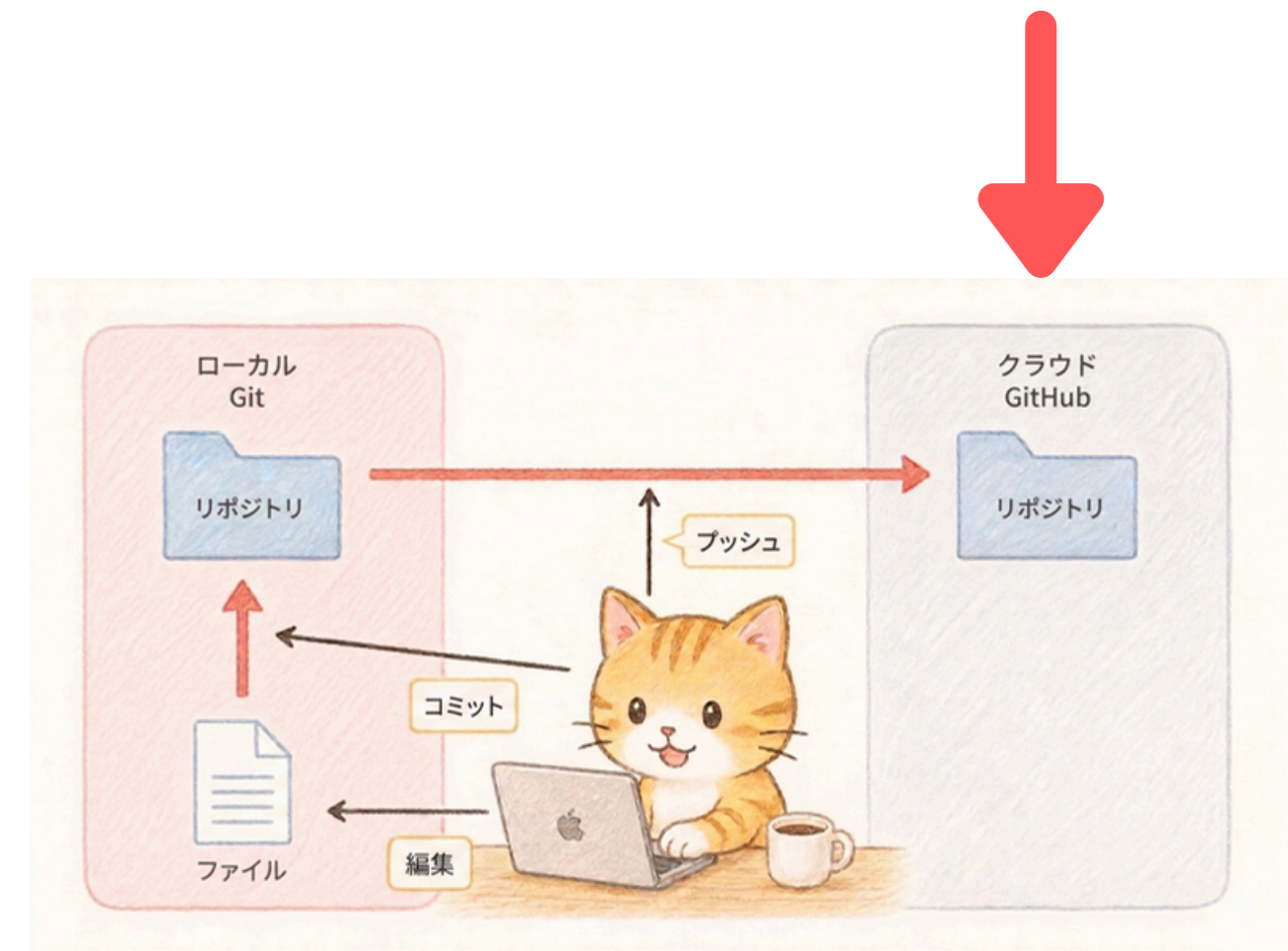
GitHub につなげる

Git（ローカル）が使えるようになったところで次のステップ！

これからやること

ローカルで作ったリポジトリを GitHub に置いて同期する

- GitHub アカウントを作成
- gh インストール
- gh 認証
- GitHub 側のリポジトリ作成
- プッシュ（push）

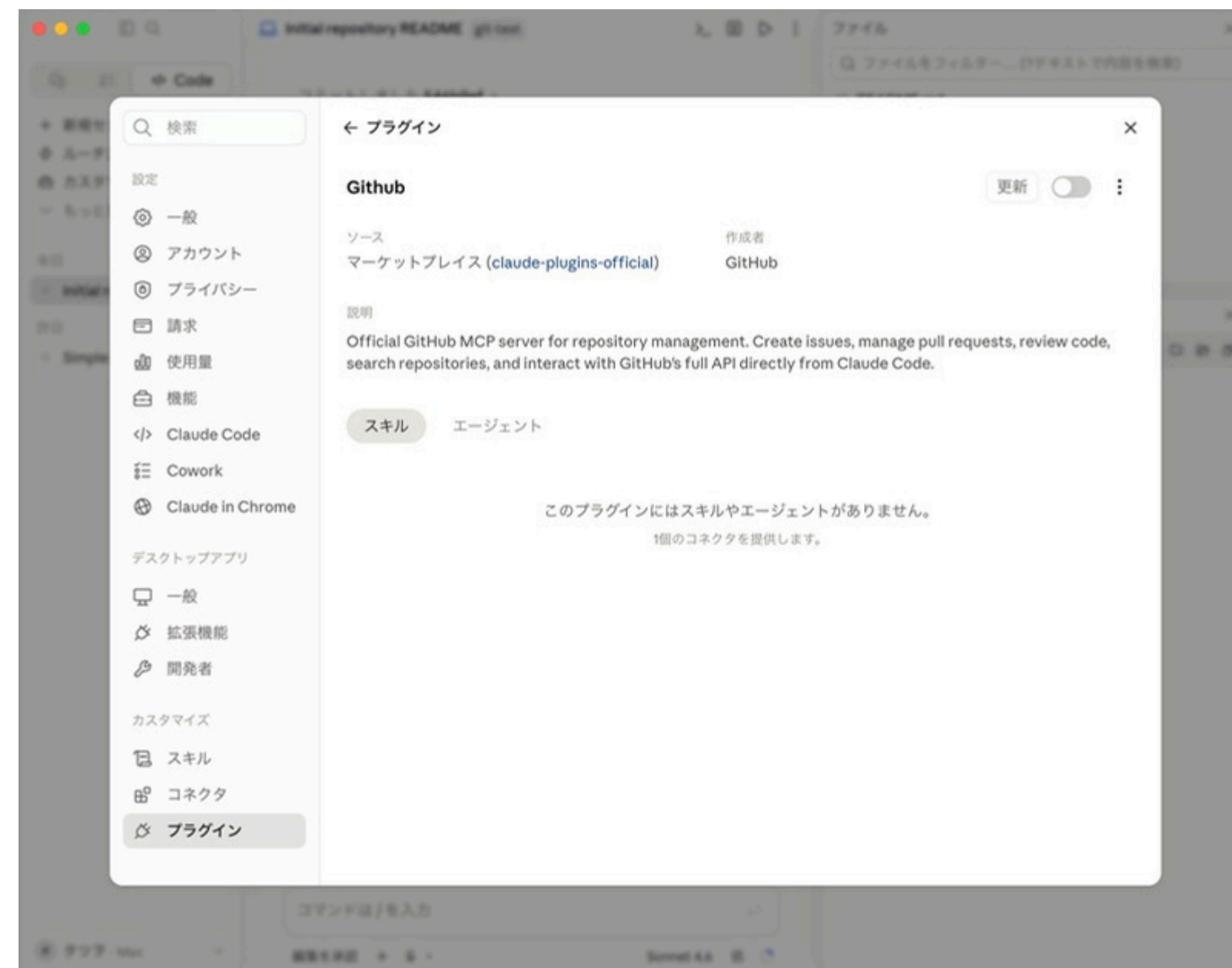


ローカルからクラウド（GitHub）へ

GitHub プラグインは無効にする

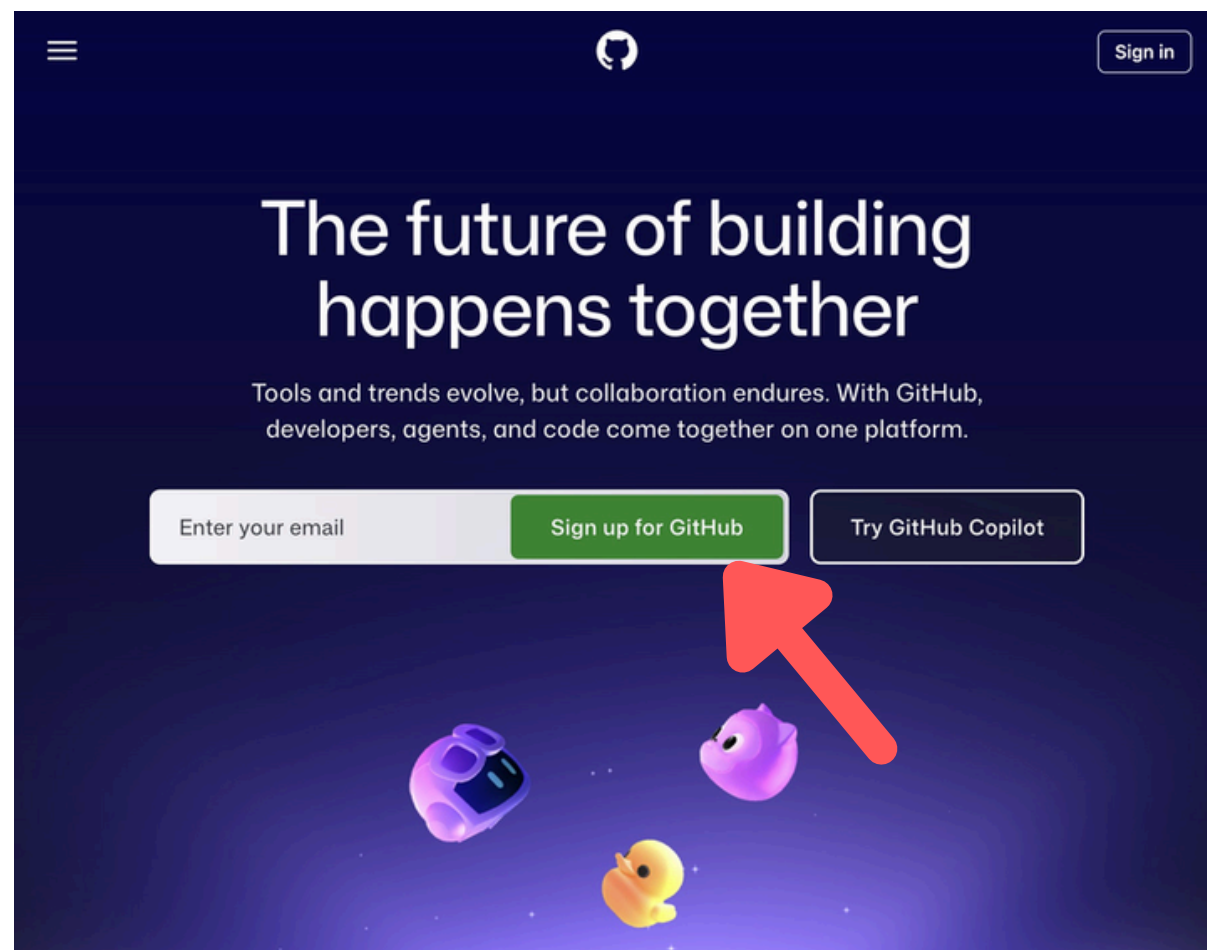
ククデ会では、
GitHub への操作（リポジトリ作成・push など）を
すべて git コマンド と gh コマンド（GitHub CLI）
で行う方針。
ということで…

- プラグイン「Github」をオフにする
 - Claudeデスクトップアプリの左下アイコン
→ 設定 → カスタマイズ → プラグイン
- コネクタ「GitHub連携」はそのままok

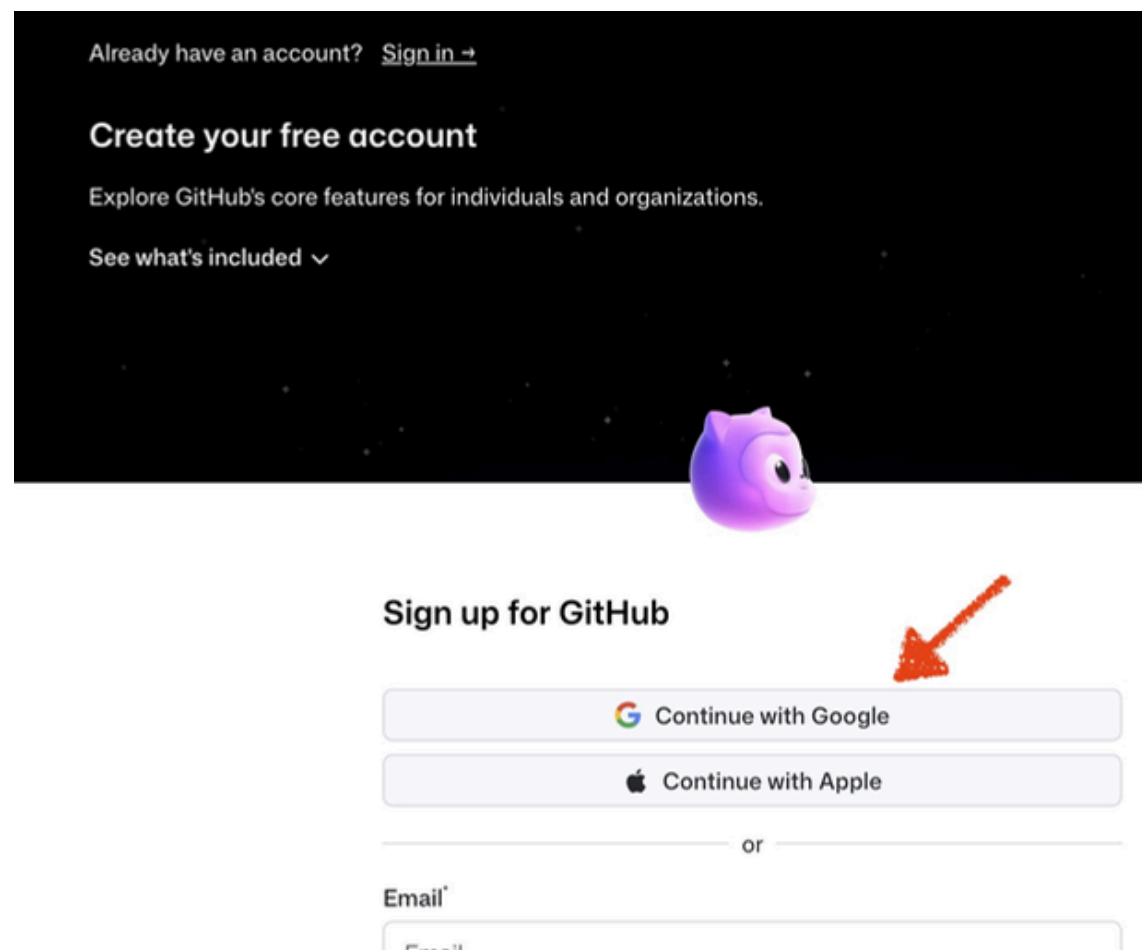


GitHub アカウントの作成（ない人）

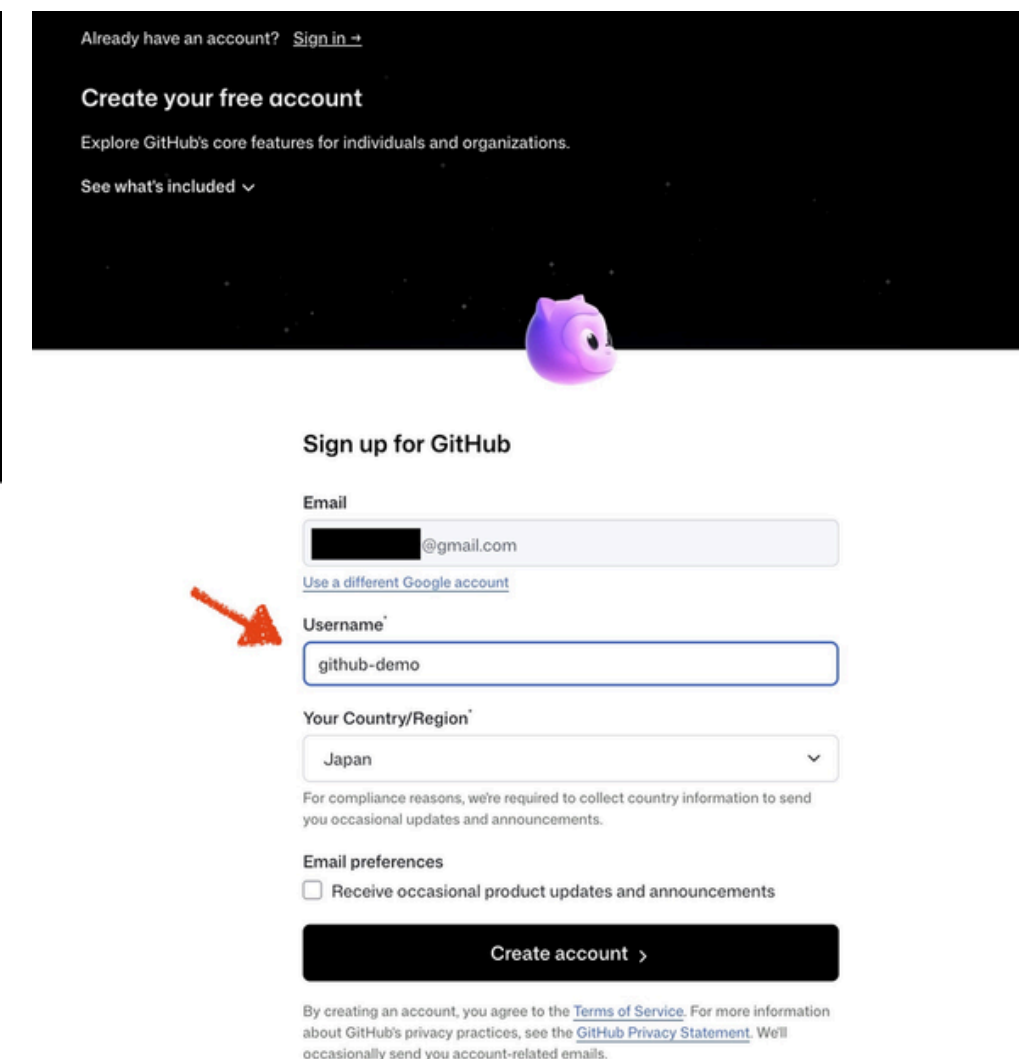
github.com にアクセスして Sign up for GitHub をクリック → Google アカウントで作成



①



②



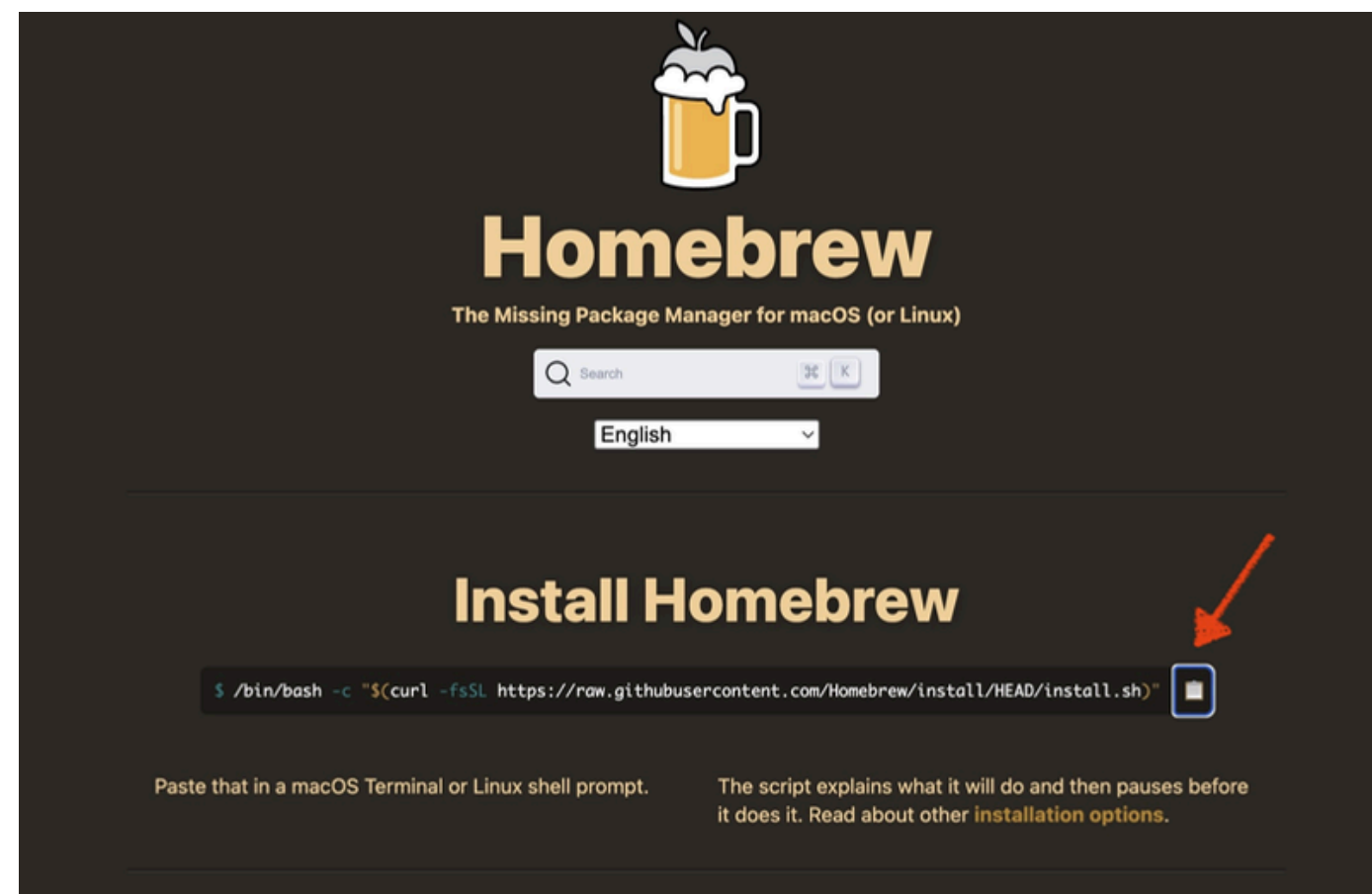
③

Homebrew のインストール (Mac)

Homebrew 公式サイト を開いて、ページに表示されているインストールコマンドをコピーボタンでコピーして、ターミナルにペーストしてリターン

最初にパスワードを聞かれるがそれはMacに入るときのパスワード。

パスワード入力文字は画面に表示されないので注意！
終わったらターミナルを開き直す（重要）



```
~/github/vibecoding% /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
=> Checking for `sudo` access (which may request your password)...
Password:
=> This script will install:
/opt/homebrew/bin/brew
/opt/homebrew/share/doc/homebrew
/opt/homebrew/share/man/man1/brew.1
/opt/homebrew/share/zsh/site-functions/_brew
/opt/homebrew/etc/bash_completion.d/brew
/opt/homebrew
/etc/paths.d/homebrew

Press RETURN/ENTER to continue or any other key to abort:
=> /usr/bin/sudo /usr/sbin/chown -R yto:admin /opt/homebrew
=> Downloading and installing Homebrew...
remote: Enumerating objects: 835, done.
remote: Counting objects: 100% (316/316), done.
remote: Compressing objects: 100% (78/78), done.
remote: Total 835 (delta 257), reused 238 (delta 238), pack-reused 519 (from 3)
=> /usr/bin/sudo /bin/mkdir -p /etc/paths.d
=> /usr/bin/sudo tee /etc/paths.d/homebrew
/opt/homebrew/bin
=> /usr/bin/sudo /usr/sbin/chown root:wheel /etc/paths.d/homebrew
=> /usr/bin/sudo /bin/chmod a+r /etc/paths.d/homebrew
=> Updating Homebrew...
Updated 3 taps (d12frosted/emacs-plus, homebrew/core and homebrew/cask).
=> Installation successful!

=> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics
No analytics data has been sent yet (nor will any be during this install run).

=> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brew#donations

=> Next steps:
- Run brew help to get started
- Further documentation:
  https://docs.brew.sh

~/github/vibecoding% █
```

gh のインストール (Mac)

Homebrew インストールしたばかりの人は開き直したターミナルで
そうでない人はターミナルを新たに開いて

brew install gh

を実行して、gh をインストールする。その後、

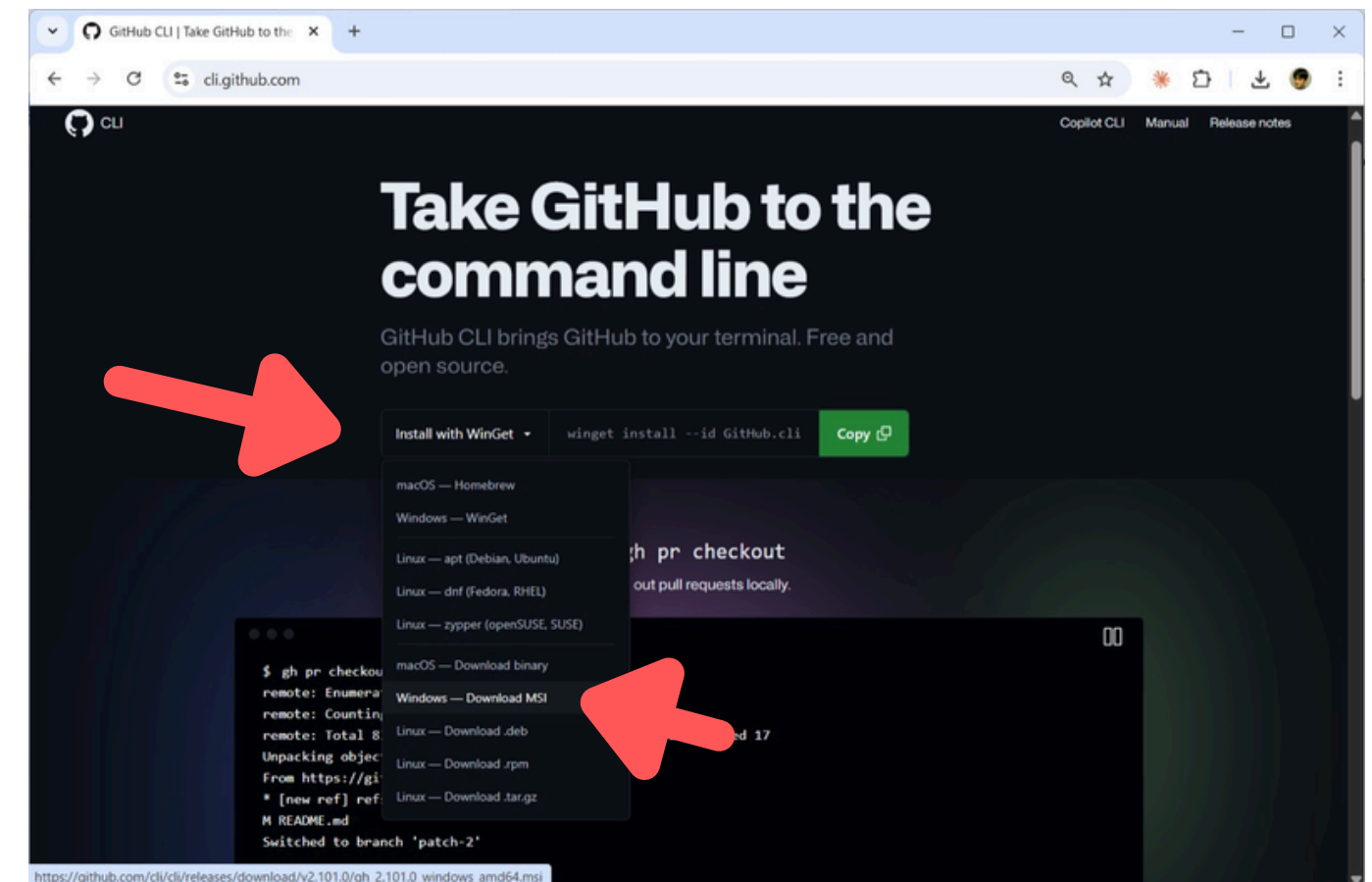
gh --version

を実行して、“gh version 2.x” などと表示されればインストール成功
Claude デスクトップアプリを完全に終了して起動し直す

gh のインストール (Windows)

<https://cli.github.com> を開く

- 最初は winget のコマンドが表示されている
- 左のボタンのプルダウン (▼) →
「Windows — Download MSI」を選ぶ
- ボタンが「Download for Windows」に変わるので、それを押す



.msi をダブルクリック → Next → Next → Install → Finish (選ぶところはない)
入れ終わったら Claude デスクトップアプリを完全に終了して起動し直す

gh で GitHub にログインする

ターミナルで下記のコマンドを実行する。

gh auth login

質問が順に出るので、全部改行で進む：

- What account do you want to log into? → GitHub.com
- What is your preferred protocol for Git operations on this host? → HTTPS
- Authenticate Git with your GitHub credentials? → Y
- How would you like to authenticate GitHub CLI? → Login with a web browser

ワンタイムコードが表示されるので、自動で開いたブラウザ画面に貼り付ける。GitHub のログイン画面で承認すれば完了。

GitHub 側のリポジトリ作成とpush

Claudeデスクトップアプリ（git-testフォルダ）にて

プロンプト：

GitHubに非公開のリポジトリ git-test を作って、今のフォルダの内容をpushして

または

`gh repo create git-test --private --source=. --remote=origin --push`

GitHub のサイトに行って、リポジトリ git-test ができたのを確認しよう！

README.md の内容も見てね！

日々の運用

プロンプト：

カフェオレについて約200文字の短文をREADME.mdに足して

プロンプト：

プッシュして

プロンプト：

AI使う人でGitHub使っていないとやばい！的な煽りコピーを3つpublic/index.htmlに足して

プロンプト：

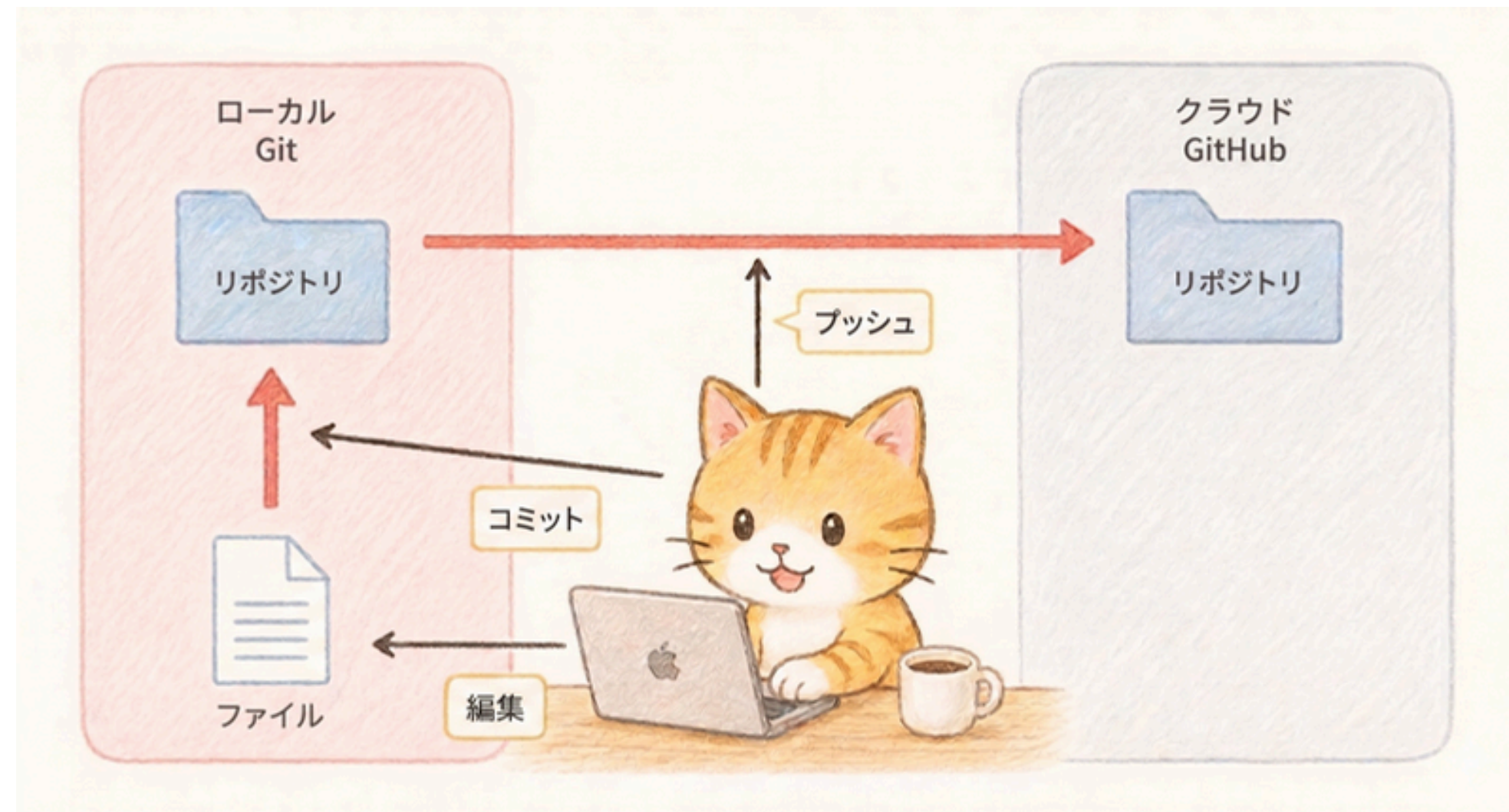
push

プロンプト：

タコライスのレシピを約100文字でREADME.mdに書いてpush

祝 Git と GitHub の初歩を習得

Git / GitHub / コミット / リポジトリ / プッシュ の意味を理解し
一通り使いこなせるようになりました。おめでとうございます 🎉



 やってみよう！

Wrangler でデプロイした Web サイトを Git/GitHub 管理してみよう！

「Git管理したい」と頼めばOK！
(裏で git init が動く)

開発基礎 作業メモの管理

[IRN] <https://vibecodingnotes.com/guide-in-repo-notes/>

「覚えておいて」の行き先

セッションをまたぐと経緯が消える。

新しいクロコは、前回何を検討して何を捨てたかを知らない。

だからといって「これ覚えておいて」と頼むと、行き先をクロコが自分で決めてしまう。

- CLAUDE.md に書かれる
 - 毎回まるごと読み込まれるので重くなる
- ツールのメモリ機能に書かれる
 - 作業フォルダの外。Codex からも自分からも見えない

メモはAIには読んでほしい。ただし目に見えるところで、自分の手で管理したい。

置き場所は作業フォルダの中

毎回読ませるルールは CLAUDE.md、判断の経緯は notes/。
どちらも作業フォルダ（≡リポジトリ）の中に置く。

myproject/ CLAUDE.md notes/ d1-vs-kv-2026-08-01.md login-key-research-2026-08-05.md	作業フォルダ 毎回読ませるルール。短く保つ 判断の経緯などのメモ置き場 AIが書くメモ（長くてよい） AIが書くメモ（長くてよい）
---	---

発明でも独自手法でもない。
単に作業フォルダに
Markdown で書かれたメモ
を置くだけ。

作業フォルダに置く利点

- いつもそこにある：目で確認できる・Codexでもクローコでも読める
- Git管理できる：変更履歴が残る・開発環境を変えても再開できる（GitHub）

ルールを試す

プロンプト：

このフォルダに作業メモの置き場所を作って。

notes/ フォルダを作り、CLAUDE.md に

「調査や検討の経緯は notes/ に

<topic>-<YYYY-MM-DD>.md の形式で置く（topic は英数字とハイフン）。

冒頭に作成日と目的、末尾に参照を書く」

というルールを追記して。

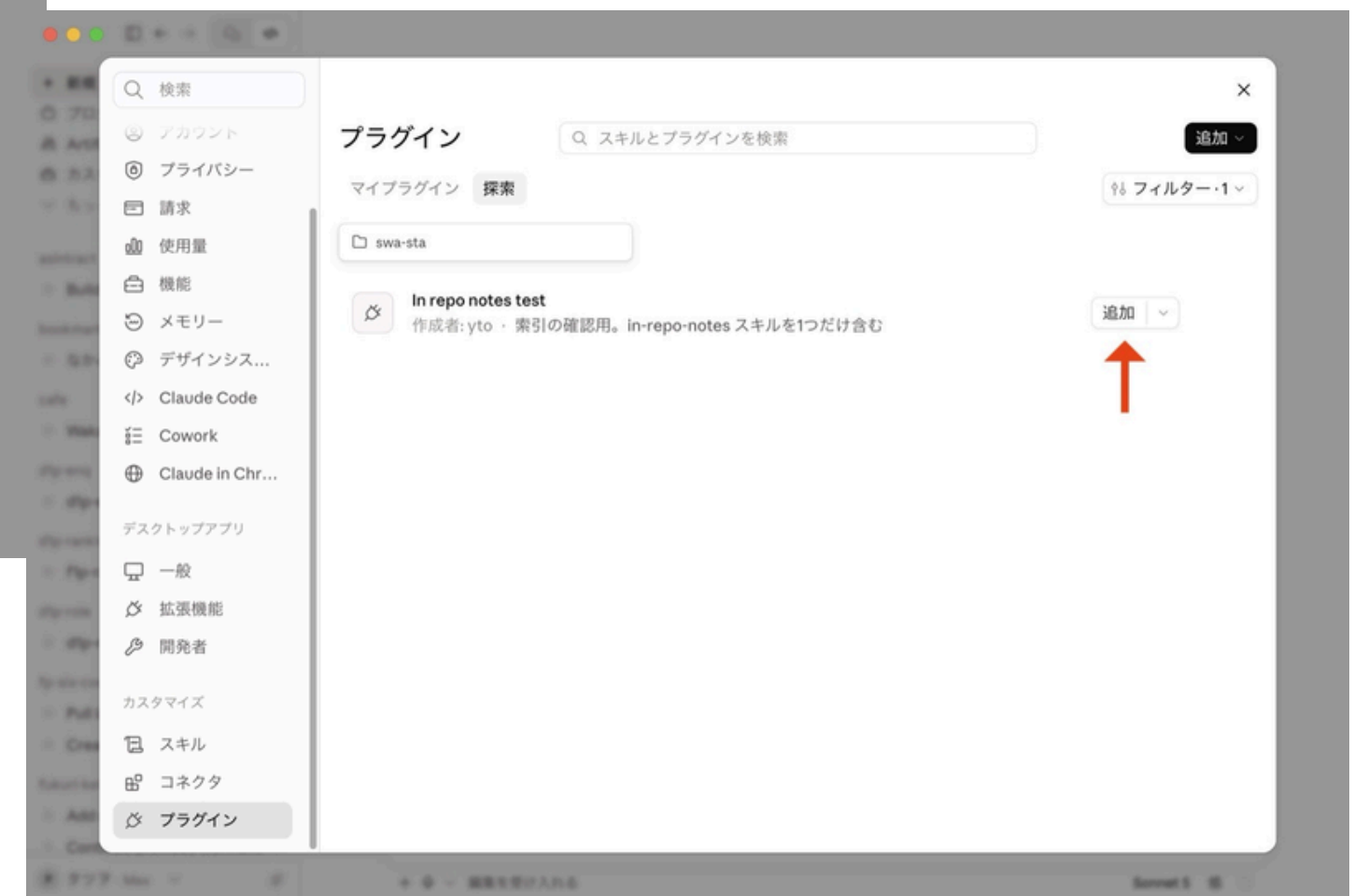
スキルを入れる（任意）

Claudeデスクトップアプリで（次ページに図）

- 設定 → プラグイン
- 右上の 追加 → マーケットプレイスを追加
- URL に yto/yto-skills → 同期
- ダイアログを閉じて 探索 タブ → In repo notes を追加

⚠ 途中で赤い警告が出ます。これは正しい警告です。

スキルの正体は「Claude への指示文」なので、先に中身を読む。



メモを挟みながら開発する（架空の例）

1. 作る前に調べる

Claude

読書記録アプリを作りたい。似たサービスがどんな機能を持っているか調べて、notes にまとめて。

→ notes/competitor-research-2026-07-01.md

2. 調べたメモを見て、作るものを絞る

Claude

さっきのメモを見て、最初に作る機能を3つに絞りたい。相談にのって。決めたことと、今回やらないことを、同じメモに書き足しておいて。

→ 同じファイルに「決めたこと」「今回やらないこと」の節が増える

こちらで説明する



<https://vibecodingnotes.com/guide-in-repo-notes/>

メモは Markdown で書かれる

プレーンテキストに記号を混ぜるだけの記法。

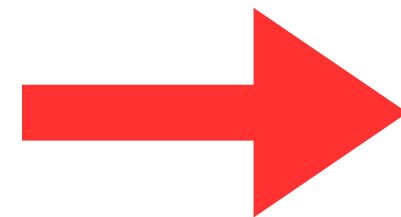
変換すると HTML になり、見出しや箇条書きとしてレイアウトされる。

```
# 見出し1
## 見出し2

- 箇条書き
- もうひとつ

**強調** [リンク](https://example.com)

| 名前 | スコア |
| --- | --- |
| Mari | 210 |
| Tom  | 200 |
```



見出し1

見出し2

- 箇条書き
- もうひとつ

強調 [リンク](https://example.com)

名前	スコア
Mari	210
Tom	200

参考：[MDB] <https://vibecodingnotes.com/guide-markdown/>

Markdownで書かれた文書を見る

- Claudeデスクトップアプリ
 - 右上の「⋮」→「ファイル」
- GitHub.com
 - push すればブラウザで整形されて読める
- github.dev
 - リポジトリのページで「.» キー。ブラウザ版 VS Code
- QLMarkdown（おすすめ）
 - macos Finder 用

作業中はクロコの「⋮ → ファイル」がいちばん早い。



やってみよう！

自分のサイトのフォルダにも `notes/` を作って、
ここまでの経緯を1本書いてみよう！

「このフォルダに作業メモの置き場所を作って」と頼めばOK！

増えてきたら足すもの（最初はやらない）

`notes/README.md` 索引。ノートが20本を超えたあたりから

`DECISIONS.md` 決定だけを追えるようにする

`notes/INBOX.md` 思いつきを1行で捕まえる

開発基礎

公開後のセキュリティ点検

[CHK] <https://vibecodingnotes.com/guide-security-check/>

このプロンプトで点検する

新しいセッション（別の会話）を開いて貼る。

プロンプト：

このURLのセキュリティをチェックして: <https://...>

まず、何でできているか調べて（ホスティング／フロント／データ置き場／外部API）。

前回の結果は使わず、いま実際にリクエストを送って確かめて。推測では答えない:

- ・ 認証なしで DB が読める / 書けるか
- ・ 認証なしで叩ける API があるか（とくに AI を呼ぶもの）
- ・ ログインなしで管理画面に入れるか
- ・ 課金が無防備に置かれていないか

報告は「実際に確認した」と「コードから推測した」を分けて。

頼み方と、直す順

- 頼むのは、まっさらな別セッションで
 - Webサイトを作ったときのセッションは自分を疑わない傾向
- 見つかったら直す順は お金 → 復旧 → 信用
 - 今日の止血（例：金額上限）と、あとでしっかり（例：認証追加）を分ける

中級 アクセス制限

[BAU] <https://vibecodingnotes.com/handson-basic-auth/>

Basic認証とは

ブラウザに昔から備わっている、いちばん手軽な鍵のかけ方
ログイン画面を自分で作らない。ダイアログはブラウザが出す

1. 鍵のかかったページを開く
2. サーバーが「401（合言葉が要る）」と返す
3. ブラウザが合言葉を尋ねるダイアログを出す
4. 入力した合言葉を付けて、同じURLを開き直す
5. 合っていればページが返る



鍵をかけたくなる場面

1. 作りかけを関係者だけに見せたい

- まだ人に見せられる状態ではないが、URLは渡して見てもらいたい

2. 身内向けのページを作った

- 家族・サークル・社内など、特定の人にだけ届けばよいページ

3. 管理用の画面を自分だけが開けるようにしたい

- 投稿の削除や設定変更ができる画面は、自分以外に開かれると困る

4. 公開前の確認用に見せたい

- 本番と同じURL・同じ環境で、限られた人に確認してもらいたい

画面を作らなくていい

【作らなくていいもの】

- ログイン画面（ブラウザがダイアログを出してくれる）
- パスワードを保存するデータベース
- 「ログイン中かどうか」を覚えておく仕組み

→ HTML 側の実装はゼロ。合言葉が合っているか照合するだけ



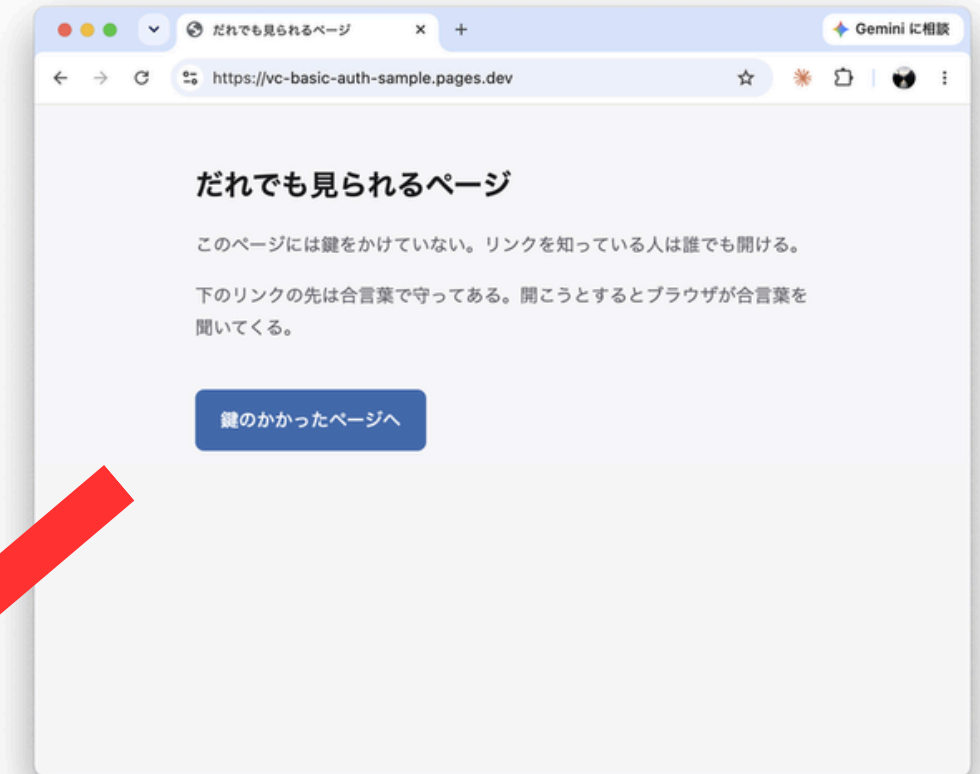
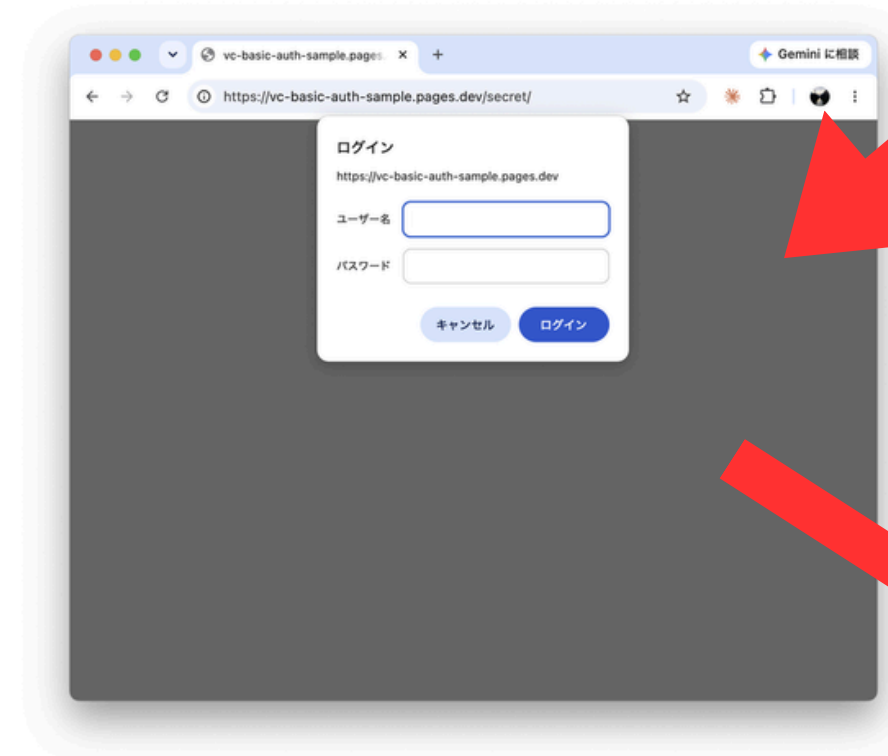
【できないこと】

- 誰がアクセスしたかは区別できない
- ログアウトできない

それでも「関係者だけに見せる」用途にはこれで足りることが多い

作るもの：2ページのサイト

- public/index.html
 - 誰でも見られるトップ
 - public/secret/index.html
 - 合言葉を知っている人だけ
-
- トップページは誰でも開ける
 - /secret/ を開こうとすると、ブラウザが合言葉を尋ねてくる
 - 正しく入れると中身が見える



作成からデプロイまでの流れ

1. Webサイト（2ページ構成）を作る
2. まず鍵なしで公開する
3. 合言葉を確かめる処理（ミドルウェア）を作る
4. 合言葉を Cloudflare に預ける
5. もう一度デプロイして反映する

① プロジェクト名を決める

- プロジェクト名を決める（例: **my-secret-site**）
 - 例のままの名前を使うと衝突するので注意
- プロジェクト名はそのまま公開URLになる（あとでCloudflareに設定する）
 - **https://<プロジェクト名>.pages.dev**
 - 例: **https://my-secret-site.pages.dev**
 -
- スライドでは **<プロジェクト名>** として表記する

① Webサイトを作る

Finderでホームフォルダの claude フォルダに、新フォルダ <プロジェクト名> を作る
ククロを起動 → 新規 → いま作ったフォルダを選択

プロンプト：

このフォルダに public フォルダを作って、2ページだけの静的サイトを作ってください。

- public/index.html は誰でも見られるトップページ
- public/secret/index.html は、あとで合言葉で守るページ
- 中身は見分けがつけばよく、凝らなくていい
- トップページから secret のページへのリンクを置いてください

② ブラウザで公開する

鍵をかける前に、普通のサイトとして公開する

この段階では2ページとも誰でも見られる

そこから鍵をかけるので、何が変わったのかがはっきり分かる

初級（ブラウザ経由でデプロイ）と同じ手順：

- Cloudflare → **Compute** → **Workers & Pages** → **Create application** → Pages の **Get started**
→ Drag and drop your files の **Get started**
- Project name に <プロジェクト名> を入れて Create project
- public フォルダをドラッグ&ドロップして Deploy site

<https://<プロジェクト名>.pages.dev> を開いて確認する

この時点では /secret/ も誰でも見られる。ここに鍵をかける

② ブラウザから作る理由

Wrangler でも新しいプロジェクトを作れる。

でも、現在は Pages ではなく Workers のように作られる（→ 初級で説明した話）

この章は Pages の functions/ を使うので、Pages である必要がある。
だからプロジェクトだけブラウザで作る。

作ったあとは Wrangler からそのまま更新できる。
次の④⑤で Wrangler を使う。

③ 合言葉を確かめる処理を作る

プロンプト：

Basic 認証で特定のパス以下を守るミドルウェアを作って。守るパスは /secret/。

- ミドルウェアはルート直下（functions/_middleware.js）に置いてサイト全体で動かし、
守るパスは1つの定数 PROTECTED_PREFIX にまとめて、そのプレフィックス以下かどうかで判定する
- 先頭スラッシュを増やした抜け道も塞ぎたいので、連続したスラッシュを正規化してから判定する
- 合言葉は環境変数 SITE_PASSWORD から読む。ユーザー名は guest 固定でよい
- タイミング攻撃を避けるため、比較は crypto.subtle.timingSafeEqual を使う

できあがるのは functions/_middleware.js の1ファイルだけ

守りたいパスはプロンプトに1か所だけ書く → 範囲を変えたくなくても直すのは1か所

④ 合言葉を預ける

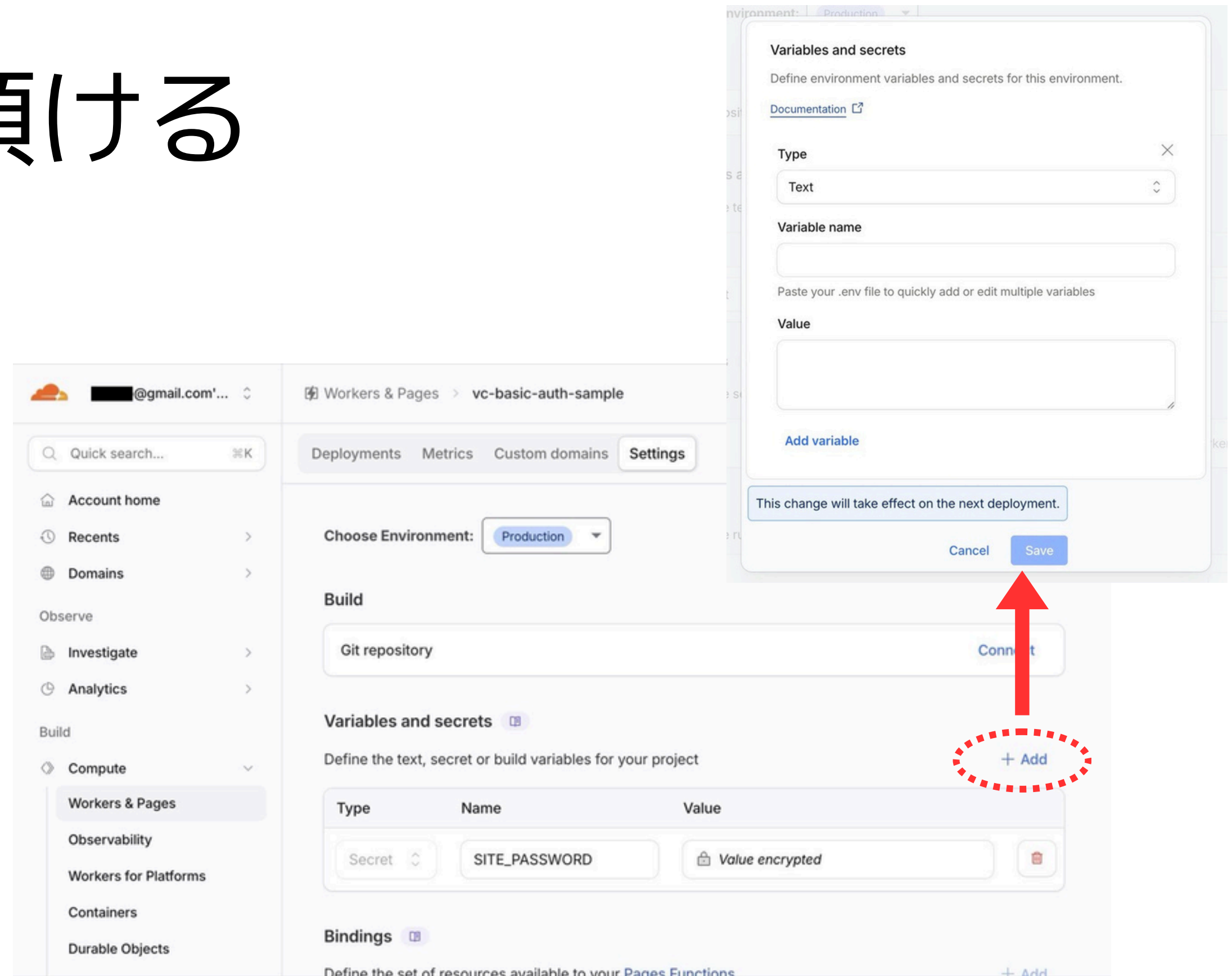
Cloudflare → **Workers & Pages** →
自分のプロジェクト → **Settings** →
Variables and secrets → **+ Add**

Type = Secret

Variable name = SITE_PASSWORD

Value = (合言葉)

⚠ 上の Choose Environment が
Production になっているか確かめる



⑤ もう一度デプロイして確かめる

ここから先は Wrangler を使う。

Cloudflare にログインしていなければ済ませておく（⚠ 自分でターミナルに入力して実行）

npx wrangler login

合言葉を預けただけでは反映されない。もう一度デプロイして、はじめて有効になる。

プロンプト：

public フォルダの中身を、Cloudflare Pages のプロジェクト「<プロジェクト名>」に、
本番（main ブランチ）としてデプロイして

【確認】

トップ <https://<プロジェクト名>.pages.dev/> は誰でも見られる

鍵付き <https://<プロジェクト名>.pages.dev/secret/> は合言葉が要る

合言葉を増やす

プロンプト：

Basic認証のユーザーを複数にしたい。

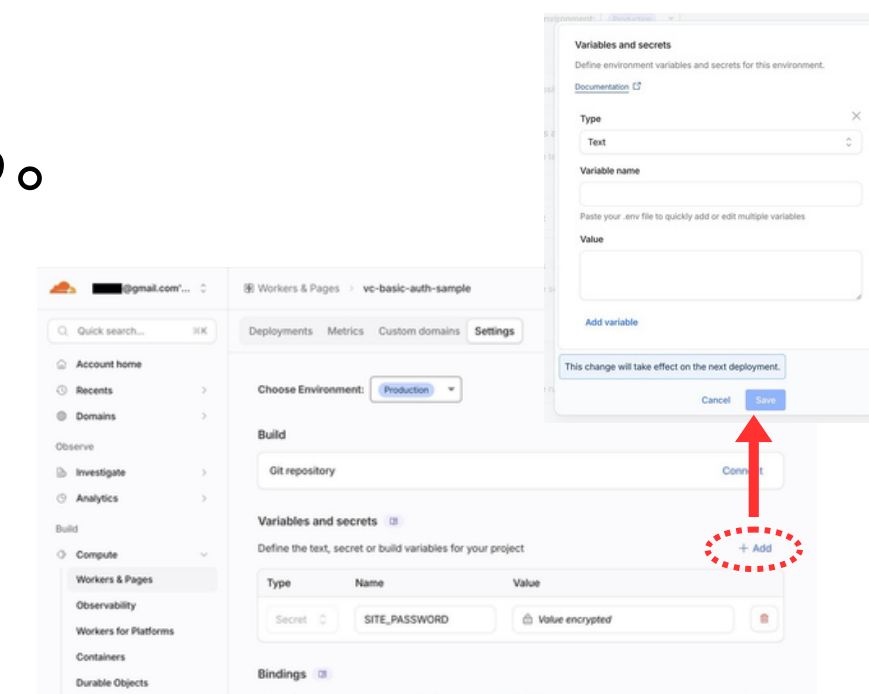
環境変数 SITE_USERS を "ユーザー名:合言葉,ユーザー名:合言葉" の形式として読み、
どれかに一致したら通すようにして。

値は私が自分で Cloudflare に入れるので、コードにも設定ファイルにも書かないで。

Cloudflare のダッシュボードで SITE_USERS を Secret で登録する。

プロンプト：

デプロイして



鍵をかける範囲を変える・増やす

鍵をかける範囲を変更する！

プロンプト：

Basic認証をサイト全体にかけたい。

パスの範囲判定を外して、どのページでも合言葉を求めるようにして

プロンプト：

デプロイして

鍵をかける範囲を増やす！

プロンプト：

Basic認証をかける範囲を /admin/ と /api/admin/ の2つにして

プロンプト：

デプロイして

できること・できないこと

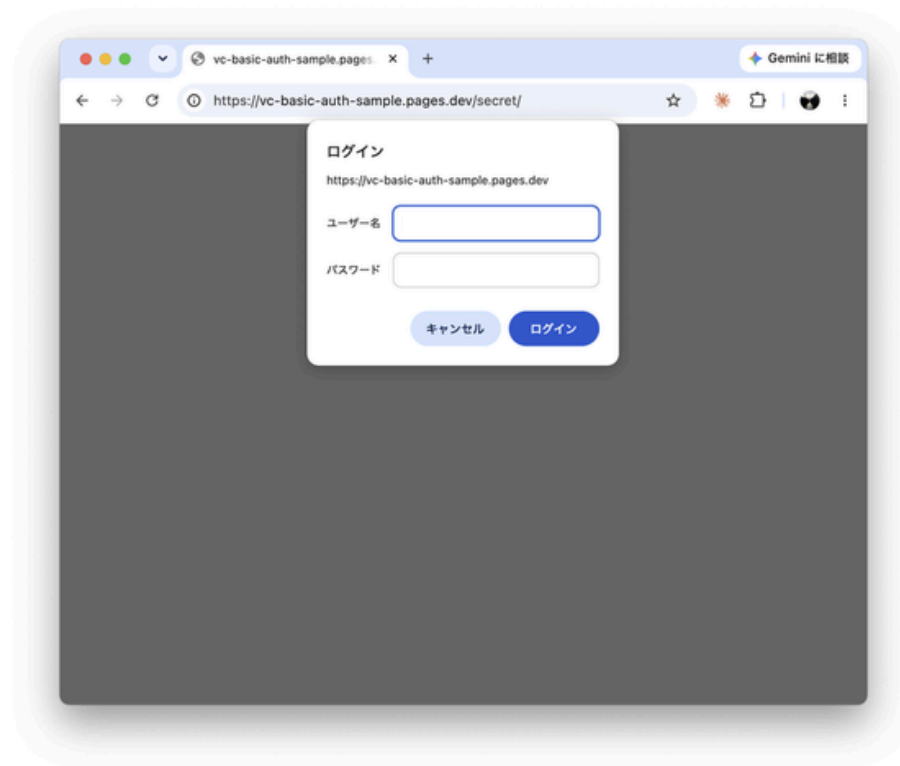
- できる
 - 合言葉を知らない人を通さない
 - 関係者だけに見せる用途なら、これで十分
- できない
 - ログアウトができない
 - 一度通れば、ブラウザを完全に終了するまで認証されたまま
 - ログイン画面（ダイアログ）の見た目を変えられない
 - 誰がアクセスしたかを区別できない
 - 合言葉を変えるときは、全員に配り直すことになる

補足：用途が変わってきたら

- Basic認証（今回）
 - 手間は最小／画面はブラウザ固定／ログアウト不可
- トークン認証
 - 手間は少ない／画面は自前／ログアウトできる
- Cloudflare Access
 - 手間は多い（初回のみ）／画面はGoogle／ログアウトできる
- ログイン画面を自前で作りたい・ログアウトを付けたい
 - トークン認証
- メールアドレス単位で許可したい・Googleの2段階認証で守りたい
 - Cloudflare Access
- 一般ユーザーそれぞれにアカウントを持たせたい
 - ログインキー／ソーシャルログイン

祝 アクセス制限を習得

公開したサイトの一部（またはサイト全体）を
合言葉を知っている人だけに見せられるようになりました。
おめでとうございます 🎉





やってみよう！

自分で作ったサイトに Basic認証をかけてみよう！

作りかけのサイト、身内向けのページ、Webアプリの管理ページなど

例えば

「/admin/ 以下に Basic認証をかけて」

「サイト全体に Basic認証をかけて」

など

（合言葉を預けるコマンドだけ、自分でターミナルに入力する）

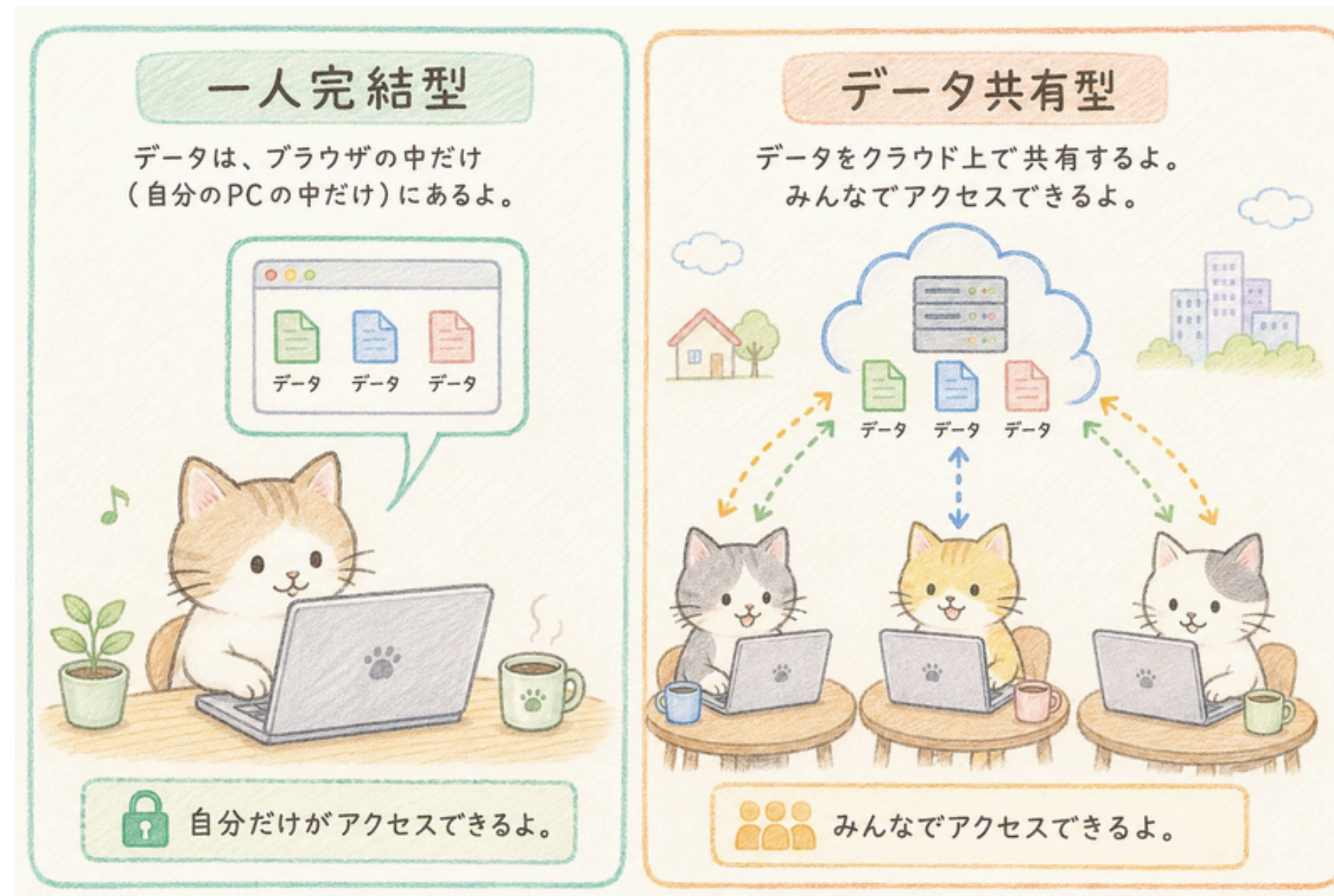
中級

Webアプリの2つのタイプ

(後編)

[TWA] <https://vibecodingnotes.com/guide-webapp-types/>

Webアプリの2つのタイプ



データの置き方は3つある

データ共有型といっても、置き方はひとつではない。

「データをどう読み書きするか」で、必要な技術要素が変わる。

- 集計
 - 集めて数えるだけ（イベントログ・Workers Analytics Engine）
- 出し入れ
 - 1件ずつ出し入れする。あとから直す・消す（データベース・D1）
- リアルタイム
 - その場で見える、その場で決まる（常駐サーバー・Durable Objects）

どんなアプリがどこに入るか

- 集計
 - アンケート／投票／アクセス解析／ランキング（上位だけ）
- 出し入れ
 - 掲示板／寄せ書き／チャット／コメント／いいね／ターン制の対戦／予約／出欠管理／会員名簿／在庫管理／自分の順位が出るランキング
- リアルタイム
 - 共同ホワイトボード／共同編集／リアルタイム対戦

汎用的なのは「出し入れ」。作りたいものの多くはここに入る。

どれになるかの判断基準

1. 全員が同じものを見る？ 自分のぶんだけ？
 - 自分のぶんだけ = 一人完結型（ブラウザの中で完結）
2. あとから直す・消す？ 1件ずつ見せる？
 - 数えるだけ = 集計
 - 直す・1件ずつ = 出し入れ
3. 相手の操作が、自分の画面にその場で映る必要がある？
 - リロードで十分 = 出し入れ
 - 勝手に変わってほしい = リアルタイム

「リアルタイム」と思いがちなもの

- 対戦ゲーム → ターン制（将棋・オセロ）なら D1 で作れる
- 予約・先着枠 → D1 で作れる。二重予約も防げる
- ランキング → 上位だけなら集計。自分の順位まで出すなら出し入れ

リアルタイム要素が必要なのは、
他の人が動かしたものが、自分の画面にその瞬間に映ってほしいときだけ。

作る前に確かめること

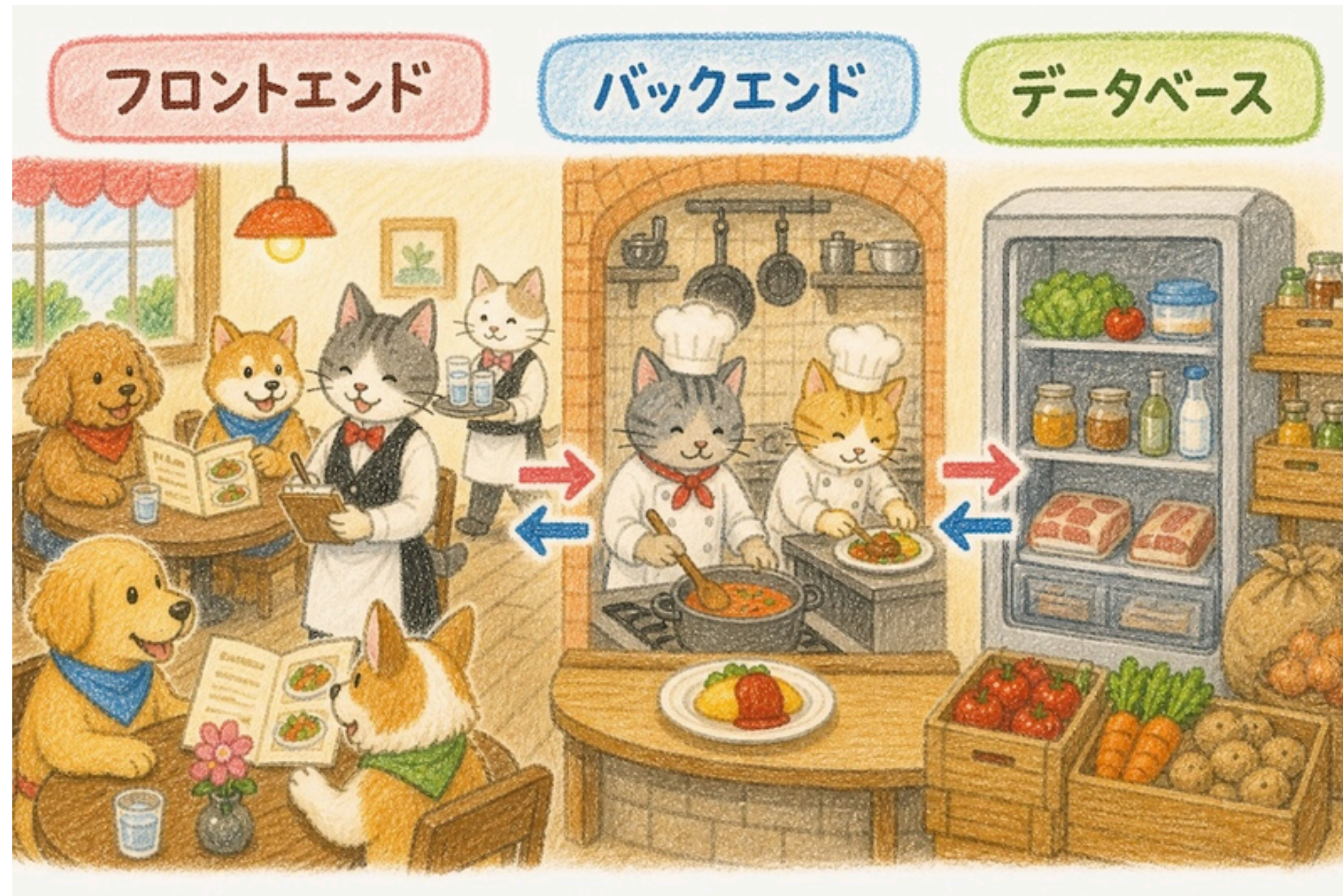
- そもそもデータをサーバで保存・処理しないで済まないか
 - 問い合わせフォームは、保存せずにGoogleフォームなどを利用すればよい
 - 一人完結型なら、ブラウザの中だけで完結する

持たなければ漏れない。外に出すのは、他の人と共有するときだけ

中級 データベース

[D1W] <https://vibecodingnotes.com/handson-cloudflare-d1-webapp/>

データ共有型（出し入れタイプ）



- フロントエンド(FE)
 - ブラウザで表示されるページまわり
- バックエンド(API)
 - 表面(FE)とDBの仲介役
- データベース(DB)
 - FEとは独立
 - 設計・作成・連携・マイグレーション

作るもの：匿名一行掲示板

- 名前の入力不要。書き込むだけで投稿できる
- 書き込んだ人には自動的に「会員1号」「会員2号」と番号が割り振られる
- 同じブラウザから書き込むと、次回以降も同じ会員番号が使われる
- みんなの投稿を一覧表示する

The screenshot shows a web interface for an 'Anonymous One-Line Bulletin Board'. At the top, it says '匿名一行掲示板'. Below this, there is a form area where the user is logged in as '会員1号' (Member 1). There is a text input field with the placeholder '一行どうぞ' (One line, please) and a '投稿' (Post) button. Below the form, it shows '5 件' (5 items) and a '更新' (Refresh) button. The list of posts is as follows:

会員番号	投稿内容	投稿日時
会員1号	いいですね	2026/05/03 04:23
会員3号	やっほー	2026/05/03 04:23
会員1号	ういうい	2026/05/02 12:41
会員2号	あああ	2026/05/02 12:02
会員1号	あ	2026/05/02 12:01

デモ: <https://bbs-sample.vibecodingnotes.com/>

開発からデプロイまでの流れ

1. ベース名と作るものを決めて作業フォルダを準備
2. データベース（D1）の作成
3. プロジェクト設定ファイルの作成
4. バックエンド（API）の作成
5. フロントエンド（UI）の作成
6. ローカルでの動作確認
7. 本番反映（デプロイ＆マイグレーション）
8. 修正と反映（再デプロイ）

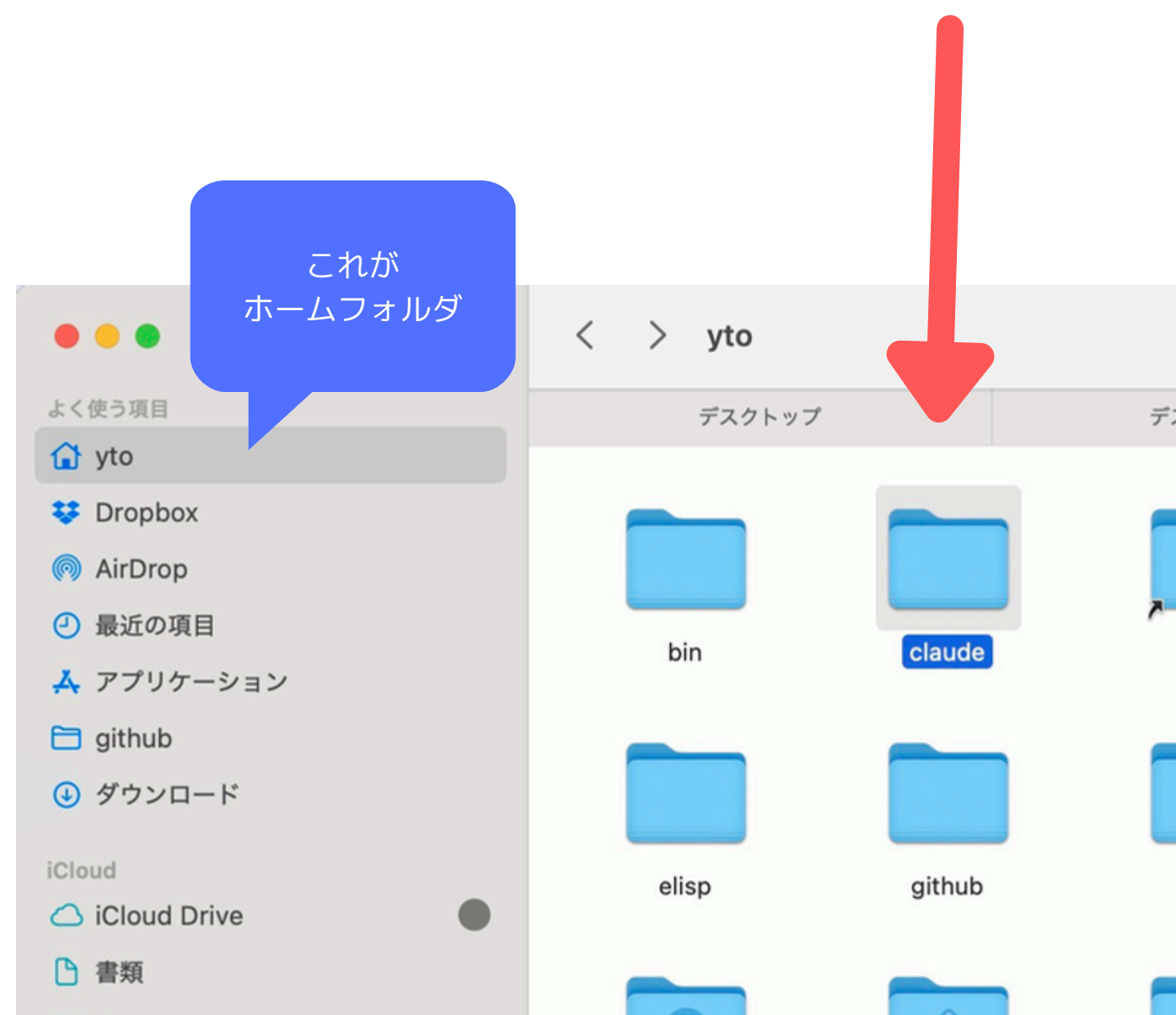
① ベース名を決める

- 自分だけのベース名を決める（例: **my-bbs**）
 - 例のままの名前を使うと衝突するので注意
- ベース名はそのまま公開URLになる（あとでCloudflareに設定する）
 - **https://<ベース名>.pages.dev**
 - 例: **https://my-bbs.pages.dev**
 - ブラウザで開いて空き確認する
 - 「このサイトにアクセスできません」と表示されたらok
- スライドでは **<ベース名>** として表記する
 - **<ベース名>-db** を **<D1データベース>** として表記する



① 作業フォルダ作成 & クロコ起動

Finderで**ホームフォルダ**の **claude** フォルダに、ベース名のフォルダ（例：my-bbs）を作る



- Claudeデスクトップアプリを起動
- 左上の“Code”を選択
- 新規
- 「ローカル」と書かれたボタンの横のフォルダアイコン → フォルダを開く
- 作業フォルダを選択
 - ホーム→claude→<ベース名>（例：my-bbs）

① クロコにベース名を覚えさせる

プロンプト：

このプロジェクトのベース名は **my-bbs** です。

これを CLAUDE.md に記録してください。

- アプリ名・プロジェクト名・GitHubリポジトリ名 = ベース名
- D1データベース名 = ベース名のうしろに -db を付けた名前

これ以降、私のプロンプトの <ベース名> はベース名に、

<D1データベース> は「ベース名-db」に読み替えて作業してください。

① クロコに作るものを覚えさせる

プロンプト：

これから作るアプリの仕様を CLAUDE.md に書いてください。

作るもの：匿名一行掲示板

- 名前登録なし、初めて投稿したときに会員番号が払い出される（1, 2, 3...の連番）
- 同じブラウザから投稿した人には毎回同じ会員番号が使われるようにする
- 画面には「会員1号」「会員2号」と表示する
- 投稿内容は1行のテキスト

② データベースの作成（D1作成）

D1 を作り、掲示板に必要なテーブルを用意する。

(2-1) D1 データベースを作成する（表示される database_id は次で使う）。

プロンプト：

D1データベース <D1データベース> を作って

または

npx wrangler d1 create <D1データベース> を実行してください。

ここまでで、データベースの「枠」ができた！

データベースの作成（テーブル設計）

(2-2) Claude Code に相談してテーブル設計（スキーマ）を相談する。

プロンプト：

CLAUDE.md に書いた仕様のとおりに、このアプリに必要なデータベースのテーブル設計を提案してください。

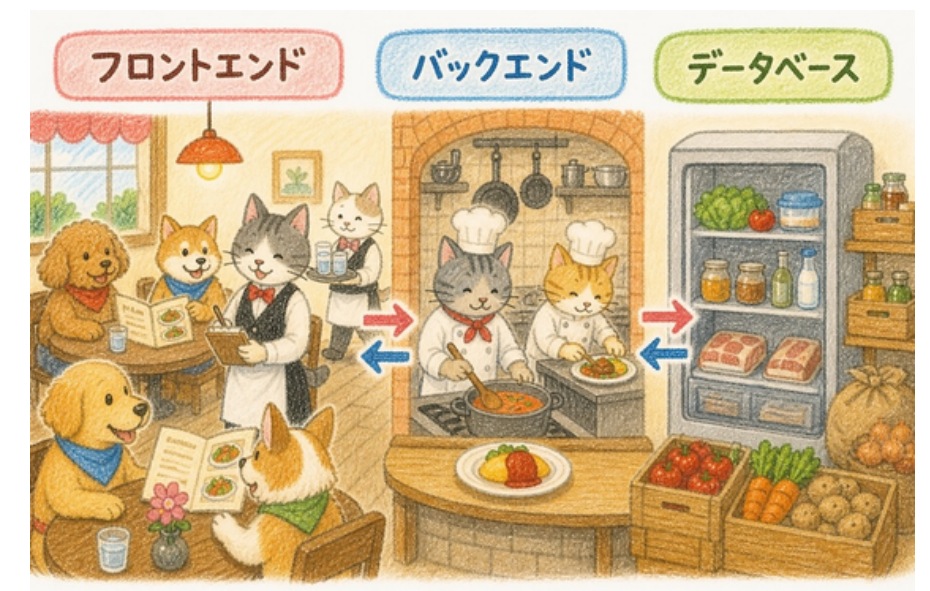
データベースの作成（ファイル作成）

(2-3) マイグレーションファイルを作る（データベース変更指令ファイル）。

プロンプト：

このスキーマでマイグレーションファイルを作成してください。
ファイルは migrations/0001_init.sql に保存してください。

ここまでで、データベースの「表（テーブル）」の
設計図（スキーマ）と
施工手順書（マイグレーションファイル）ができた！



③ プロジェクト設定ファイルの作成

ファイル wrangler.jsonc を作成する。

(プロジェクトの基本設定、「DB ↔ プロジェクト」の結びつけ設定)

プロンプト：

このプロジェクト用の wrangler.jsonc を作って。

Cloudflare Pages プロジェクトで、公開ディレクトリは public。

D1データベース <D1データベース> を binding 名 DB で使う。

マイグレーションは migrations フォルダ。

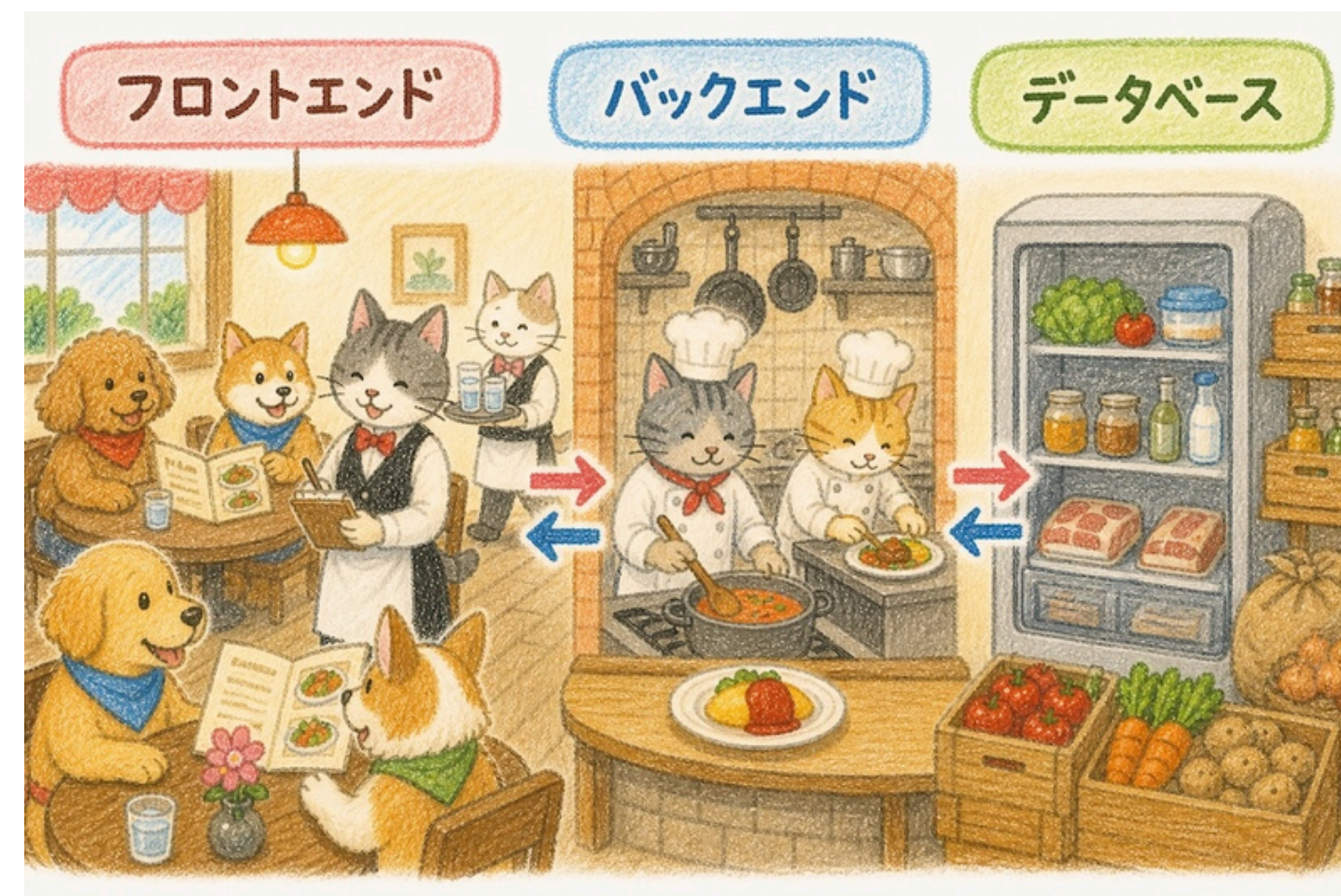
compatibility_date は UTC (協定世界時) での今日の日付にする

database_id は d1 create で表示された値を入れる。

④ バックエンド（API）の作成

プロンプト：

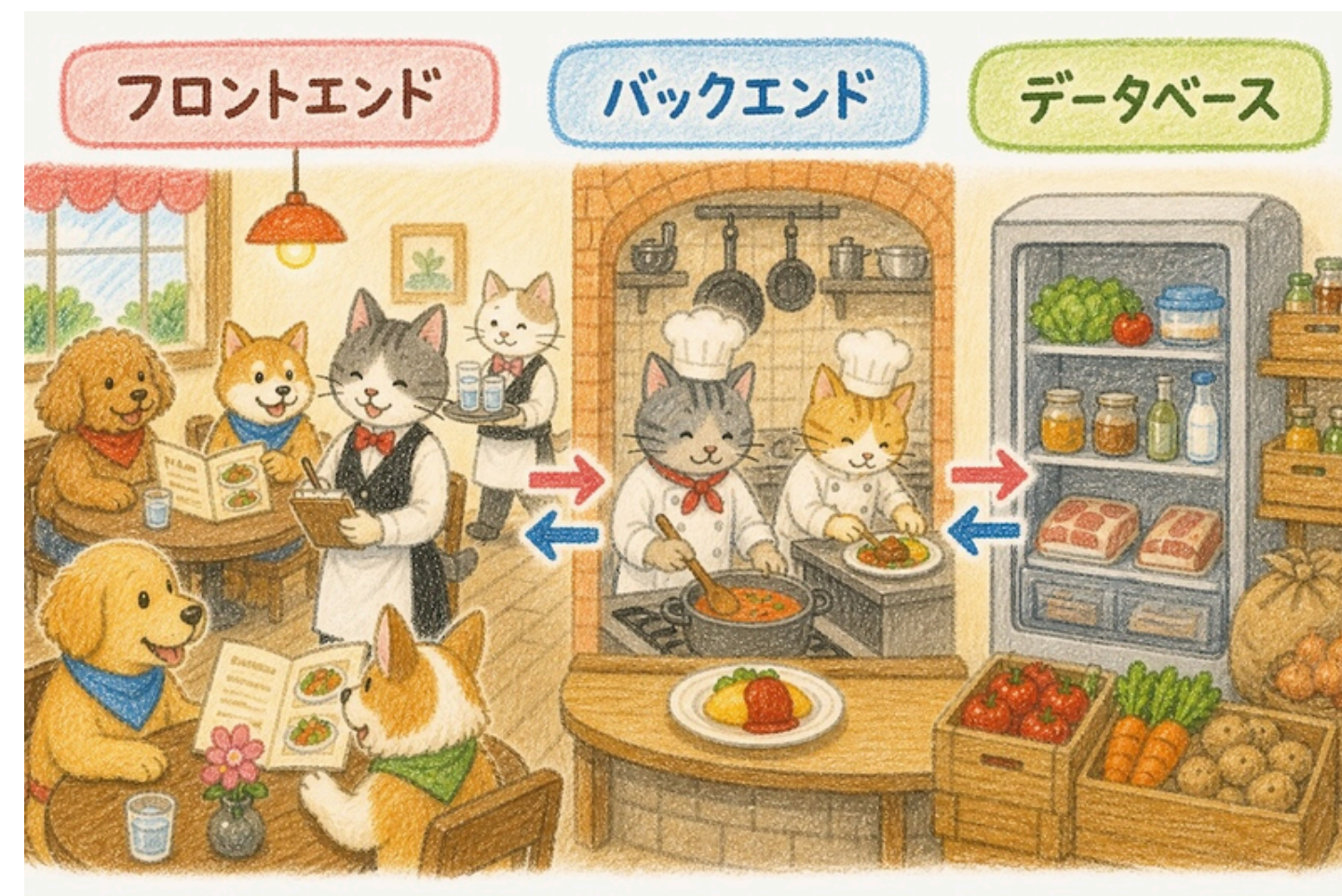
スキーマ作成時に提示した仕様と、
先ほど設計したスキーマを使って、
Pages Functions で API を作って。
D1 の binding 名は DB。



⑤ フロントエンド (UI) の作成

プロンプト：

これまでのDB定義、API定義を参考に
フロントエンドを `public/index.html` と
して作成してください。



⑥ ローカルでの動作確認

公開前に手元で動かして確認する

- 本番DBではなく「ローカルD1」にマイグレーション適用 → ローカルサーバー起動
- `http://localhost:8788`（例）をブラウザで開いて確認

プロンプト：

本番DBではなくローカルのD1で動作確認したい。

ローカルのD1にマイグレーションを適用してから、ローカルサーバーを起動して

⑦ 本番反映

プロンプト：

本番DBにマイグレーションを適用してから、
wrangler で Cloudflare に直接デプロイして

手元（ローカル）の変更 → 本番に反映する（本番反映）

└ コード（画面・API） → デプロイ（今の状態で丸ごと置き換える）

└ DBのスキーマ → 本番DBへのマイグレーション適用（変更を1回だけ足す）

⑧ 修正と反映（再デプロイ）

プロンプト：

ユーザーごとに投稿の背景色を変えてほしい

プロンプト：

デプロイして

匿名一行掲示板

あなたは 会員1号 です

一行どうぞ

5 件

会員1号	いいですね	2026/05/03 04:23
会員3号	やっほー	2026/05/03 04:23
会員1号	ういうい	2026/05/02 12:41
会員2号	あああ	2026/05/02 12:02
会員1号	あ	2026/05/02 12:01

データベースを削除する方法

Cloudflare のダッシュボード左サイドバー

- Storage & databases
- D1
- データベース名
- Delete ...

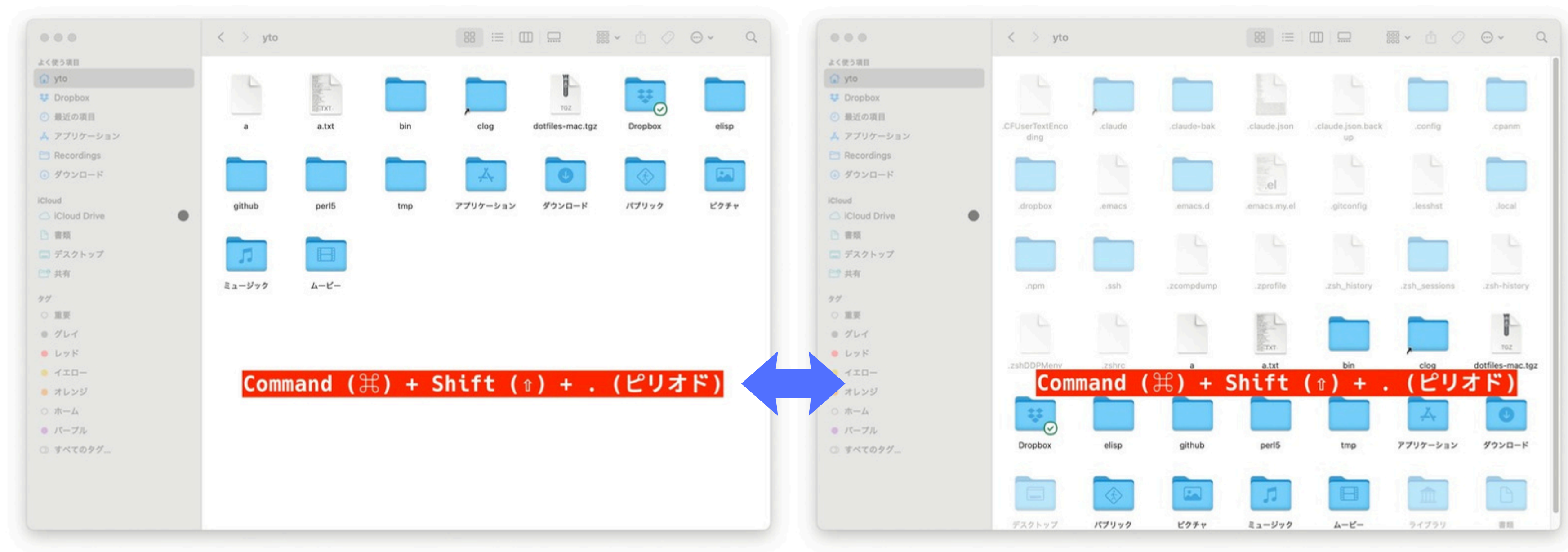
祝 データ共有型Webアプリの作成・公開を習得

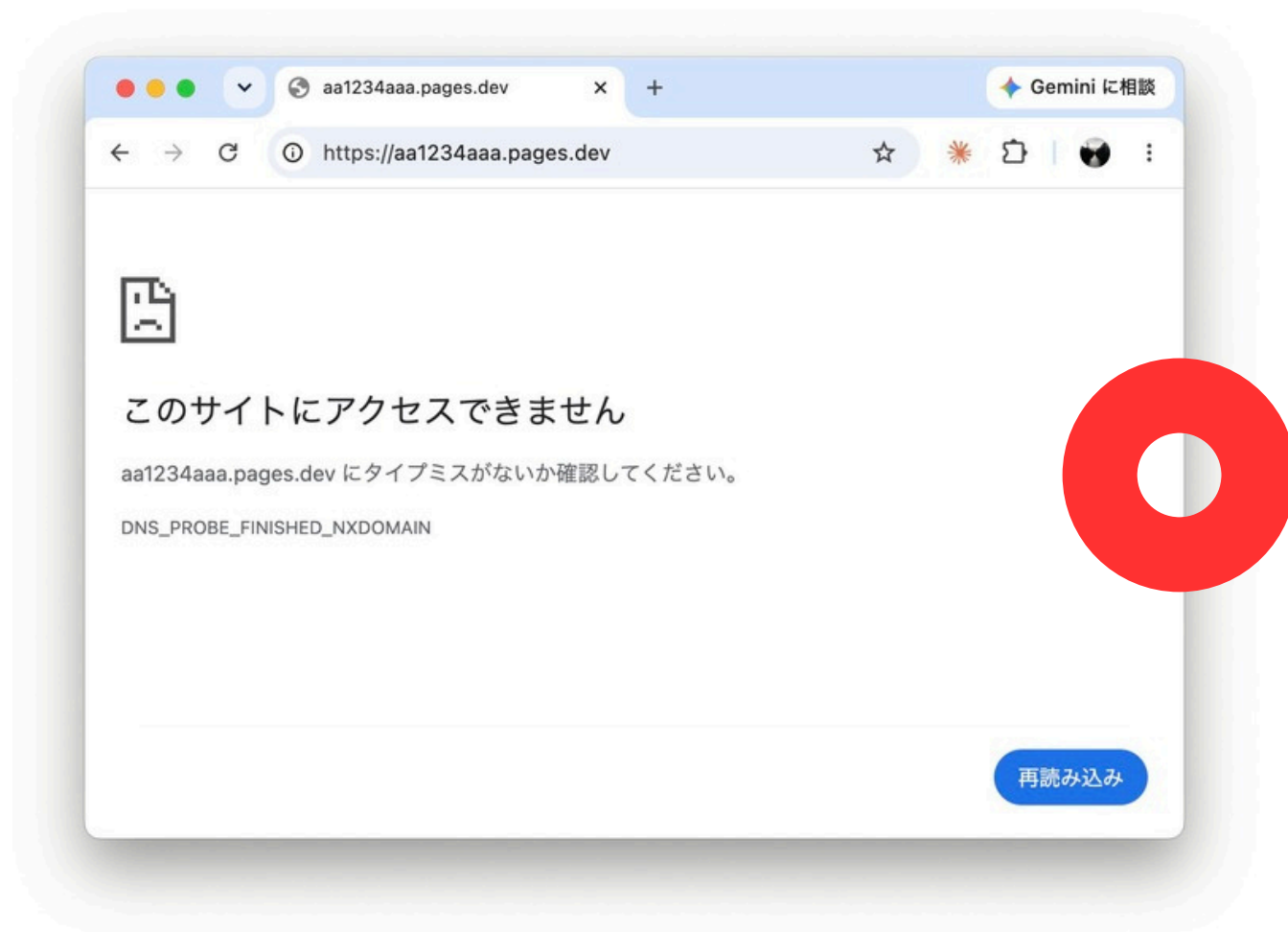
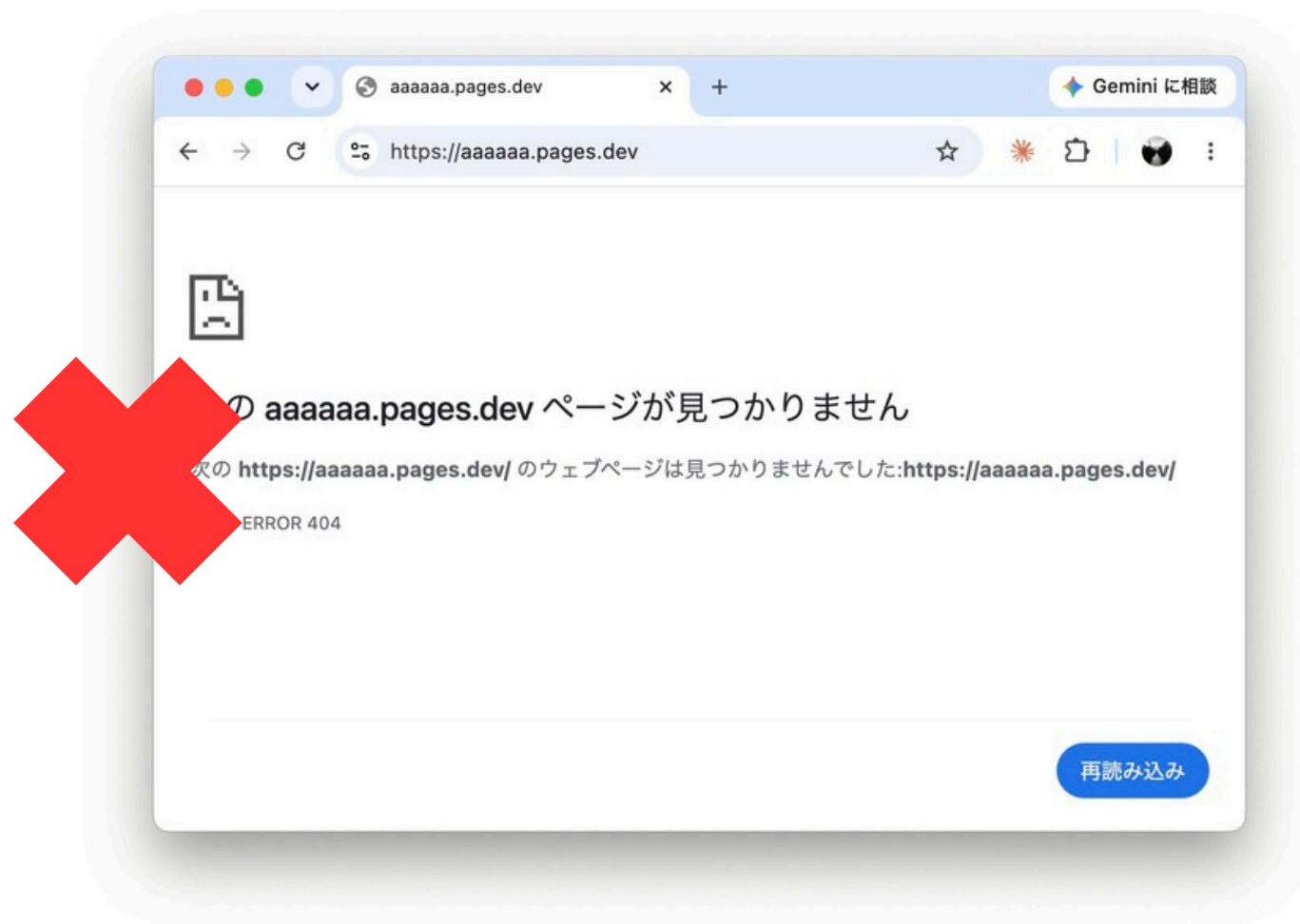
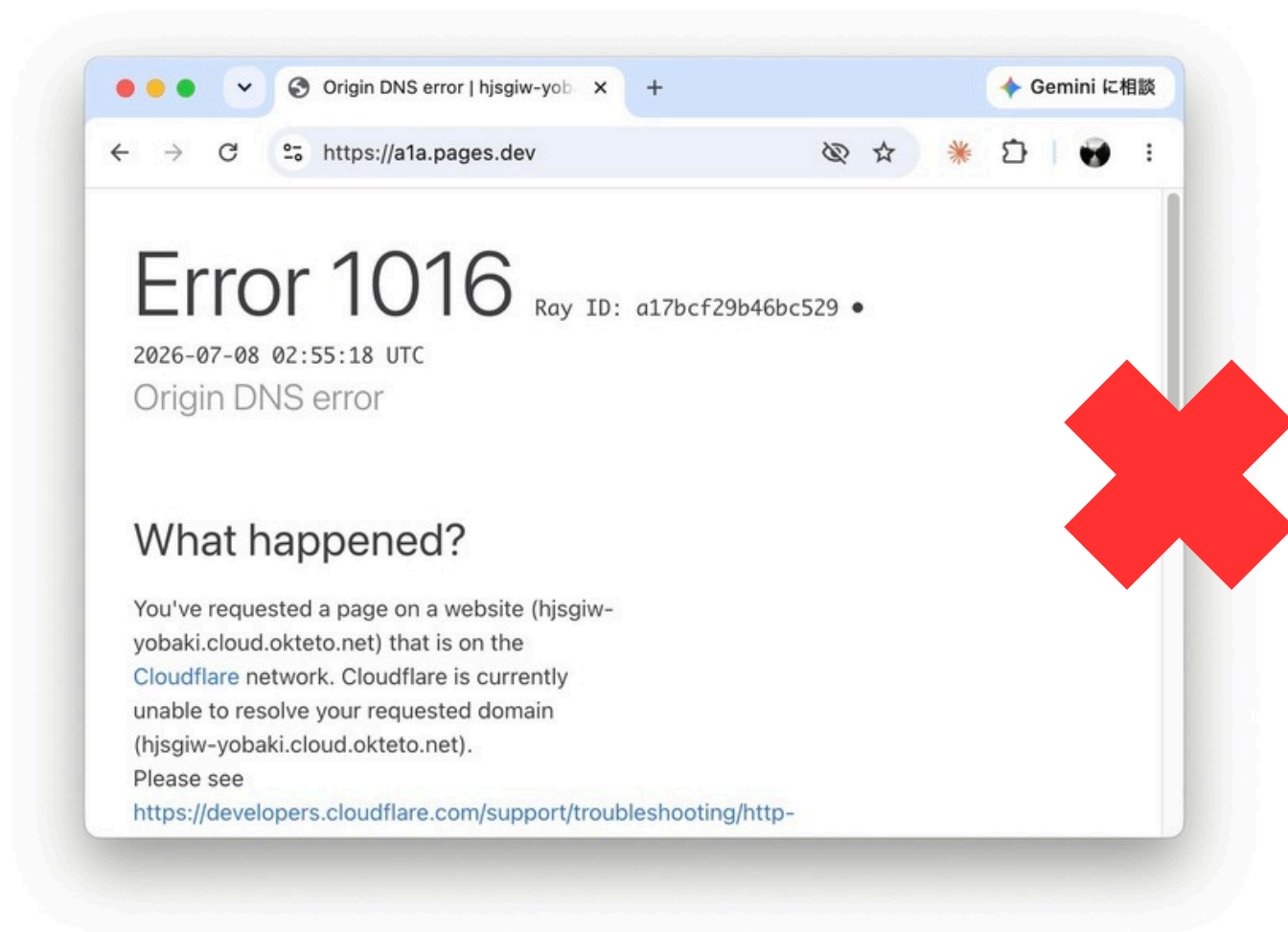
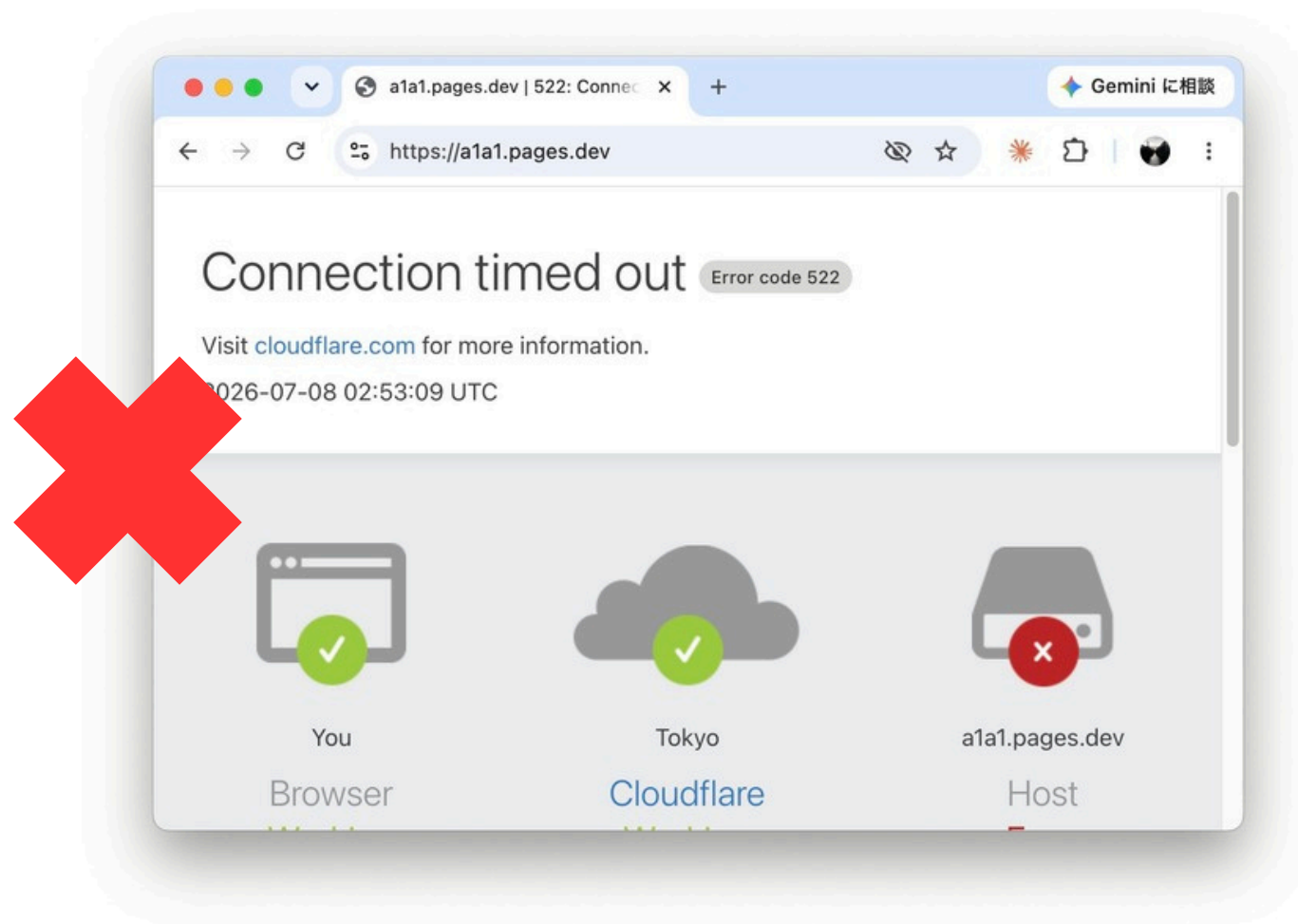
Cloudflare D1 データベースを使ったデータ共有型Webアプリの作成・公開ができるようになりました。おめでとうございます🎉



補足資料

“.”で始まるファイル/フォルダを表示





たぶんOK

前提

- 必要なもの
 - Mac（自力で進められるならWindowsでもok）
 - Claude Pro以上 + Claudeデスクトップアプリ
- 進めていくと必要になるもの
 - インストール: Node.js（Wranglerを使うため） / Homebrew / git / gh
 - Webサービス: Cloudflare / GitHub（どちらもGoogleアカウントがあると便利）
- やること
 - Claude Codeでバイブコーディング（Webサイト/Webアプリを作成）
 - CloudflareでWeb公開（デプロイ）
- 対象
 - 個人・小規模の趣味・ビジネスサイト