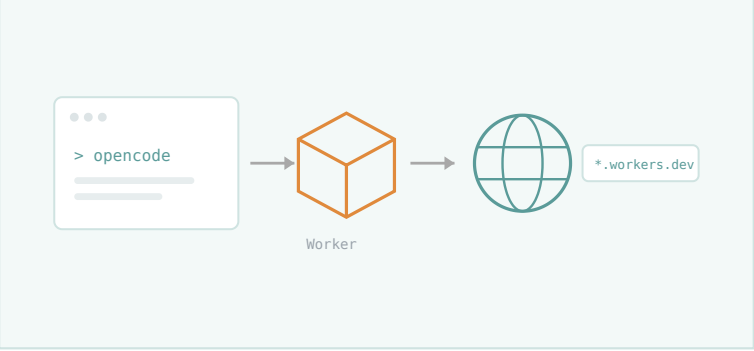
手順書

研修Day3
Cloudflare Workers ハンズオンマニュアル

「消えるURL」から「消えないURL」へ

OpenCode で Worker を作り、世界に公開する



このハンズオンのゴール

OpenCode で **Cloudflare Worker** を作り、自分の **URL** で世界に公開する

Day2 では OpenCode に日本語で指示してダッシュボードを作り、Quick Tunnel の**一時 URL** で隣の人に見せました。Day3 はその続きです。同じ Codespace の中で、今度は Cloudflare Workers に `deploy` し、自分の PC を閉じても消えない URL を手に入れます。

① 作る

OpenCode に指示して、Hello, World を返す Worker のひな形を生成する。

② 動かす

wrangler dev でローカル確認。Day2 と同じポート 8787 を使う。

③ 公開する

wrangler deploy で workers.dev の URL を取得し、隣の人と交換する。

Day2（OpenCode）との対応

Day2（OpenCode）	Day3（Cloudflare Workers）
日本語プロンプトで実装させる	同じやり方で Worker を作る
wrangler dev（ポート 8787）で確認	同じ。これが Workers 本来の使い方
cloudflared の一時 URL で共有	wrangler deploy の恒常 URL で共有
Codespace を止めると URL は消える	Codespace を止めても URL は生きている

この資料の位置づけ

本書は配布書籍『Web開発者のための［入門］Cloudflare Workers』（技術評論社）の**第1章にあたる範囲を、60分で手を動かして通す**ための手順書です。概念の説明は最小限にとどめています。各 Step に対応する章を記載しているので、当日わからなかったところ・もっと知りたいところは本を開いてください。

> **OpenCode が止まったときは、手打ちに切り替えてください**
>
> Step 2 以降は、**OpenCode に任せるルート**と**自分でコマンドを打つルート**を併記しています。どちらでも同じ結果になります。コピペ用のコードは全文を載せているので、詰まったら迷わず切り替えてください。

> **Day2 の「後日回し」が、今日の本編です**
>
> Day2 手順書の最後で後日メニューとした「OpenCode Skills（Hono / Cloudflare / Wrangler）の呼び出し」と「`npm create cloudflare@latest` からのプロジェクト作成」が、この Day3 の内容です。

概要

所要時間	約 60 分（事前課題の時間は含みません）
前提	① Cloudflare アカウントを作成済みであること（**事前課題**・Step 0-A） ② Day2 と同じリポジトリ（`kzkski/cloudflare-dev-container`）にアクセスできること
起動手順	Day2 と同じ手順で GitHub Codespaces を作成します（Day2 手順書の準備編を参照）。Day2 の Codespace が残っていれば、それを再開しても構いません。
料金	**クレジットカードの登録は不要**。Cloudflare の無料プランで今日の内容はすべて動きます（Workers は 10 万リクエスト／日まで無料）。
後日回し	Hono でのルーティング、KV / D1 でのデータ保存、Cron Triggers での定期実行、独自ドメインの割り当て

当日の流れ（目安）

時刻	Step	内容
0–5 分	0-B	動作確認と Cloudflare ログイン
5–13 分	1	Cloudflare Workers とは（座学）
13–27 分	2	Hello, World を作る
27–40 分	3	デプロイして公開 URL を得る（本丸）
40–52 分	4	JSON を返す
52–60 分	—	料金・後片付け・振り返り

> **30 分で区切る場合**
>
> Step 3（デプロイ）まで到達すれば、このハンズオンの目的は達成です。Step 4 は各自あとから。止まった Step のプロンプトから再開できます。

進捗チェックリスト

- [ ] Step 0-A　Cloudflare アカウント作成（事前）
- [ ] Step 0-B　wrangler login
- [ ] Step 1　座学（Workers とは）
- [ ] Step 2-1　プロジェクト作成
- [ ] Step 2-2　wrangler dev で表示
- [ ] Step 3-1　wrangler deploy
- [ ] Step 3-2　URL を交換する
- [ ] Step 4-1　JSON を返す

0

準備編 | 事前課題と Cloudflare ログイン

60 分でいちばん時間が溶けるのは「アカウント作成」です。
ここだけは前日までに済ませてきてください。
当日は **`wrangler login`** から始めます。

0-A　事前課題：Cloudflare アカウントを作る（前日まで・約 5 分）

目的：当日 `wrangler login` だけで始められる状態にしておく。

1　サインアップページを開く

> サインアップ URL
> `https://dash.cloudflare.com/sign-up`

メールアドレスとパスワードだけで作成できます。**クレジットカードの登録は求められません**（無料プランのまま今日の内容はすべて動きます）。

2　確認メールのリンクを開く

登録したアドレスに確認メールが届きます。本文のリンクを開いてアカウントを有効化してください。届かないときは迷惑メールフォルダも確認します。

3　ダッシュボードにログインできることを確認する

左のメニューに **Compute (Workers)** の項目が見えていれば準備完了です。ここまでで前日の作業は終わりです。

> ✓ 完了条件
> - Cloudflare のダッシュボードにログインできる
> - 左メニューに Compute (Workers) がある

> ⚠ よくあるつまずき
> - 確認メールが届かない → 迷惑メール、または組織ドメインのフィルタ。個人の Gmail 等で作っても構いません。
> - すでに Cloudflare アカウントを持っている → そのまま使えます。新規作成は不要です。
> - プラン選択を求められた → **Free** を選びます。

0-B　当日：Codespace を開いて wrangler にログインする（5 分）

目的：Codespace のターミナルから Cloudflare を操作できる状態にする。

terminal

```
$ npx wrangler --version
$ npx wrangler login --device
```

なぜ --device を付けるのか

Codespaces のようなリモート環境では、通常のログイン（ブラウザが開いて戻ってくる方式）のコールバックが届かず固まることがあります。`--device` を付けると「画面に出たコードを手元のブラウザに入力する」方式になり、確実です。

1. **ターミナルに出た URL とコードを控える**
   確認用のコード（英数字）と、開くべき URL が表示されます。
2. **手元のブラウザでその URL を開き、コードを入力する**
   Cloudflare にログインした状態で開き、内容を確認して **Allow** を押します。
3. **ターミナルに戻り、ログインを確認する**

   terminal

   ```
   $ npx wrangler whoami
   ```

✓ 完了条件

- `npx wrangler whoami` で自分のメールアドレスと Account ID が表示される

⚠ うまくいかないとき

- ブラウザが勝手に開いて固まる → `npx wrangler login --device --browser false`
- それでもログインできない → **API トークン方式**に切り替えます。ダッシュボード右上のアカウントアイコン → My Profile → API Tokens → Create Token → テンプレート「Edit Cloudflare Workers」を選択して発行。

```
$ export CLOUDFLARE_API_TOKEN=（発行したトークン）
$ npx wrangler whoami
```

それでも解決しないときは、付録のプロンプト早見表「Step 0-B | ログインで詰まったとき」を OpenCode に貼ってください。

1

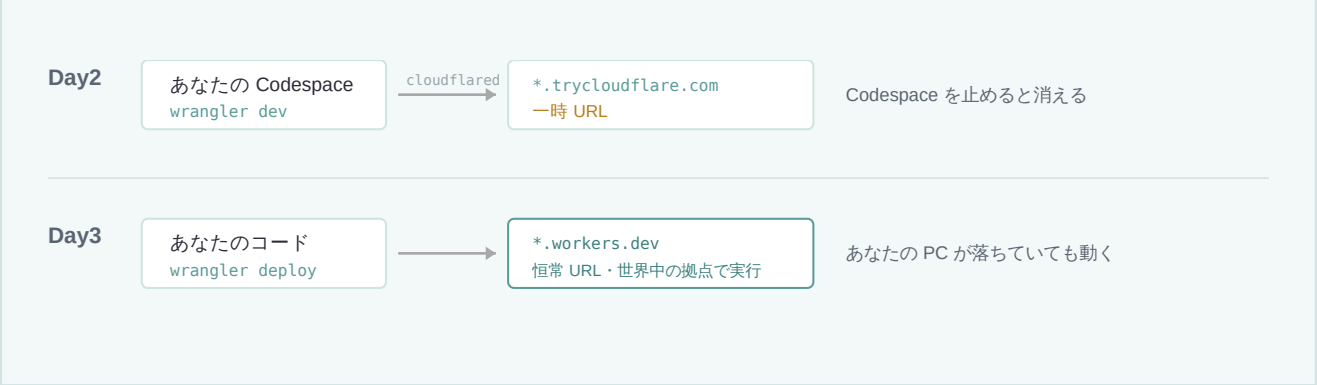
座学 | Cloudflare Workers とは

手を動かす前に **8** 分だけ。
詳しい説明は配布書籍の第 **1** 章にあります。
ここでは「昨日の **Quick Tunnel** と何が違うのか」だけ掴んでください。

1-1 一言でいうと（4 分）

自分でサーバを持たずに、世界中の Cloudflare の拠点で JavaScript / TypeScript を動かす仕組みです。URL にアクセスがあったときだけコードが起動し、レスポンスを返します。動いていない間の費用も、サーバの管理も発生しません。

Day2（一時 URL）と Day3（恒常 URL）の違い



押さえるのは 3 つだけ

起動が速い	サーバの起動待ちがありません。アクセスが来た瞬間に実行され、終わったら消えます。
世界中で動く	デプロイすると自動的に世界中の拠点に配られ、アクセスした人にいちばん近い場所で実行されます。
無料で始まる	クレジットカード不要。10 万リクエスト／日まで無料。個人の実験や試作には十分です。

> 📖 本のどこ
>
> 1.1 Cloudflare Workers とは ／ 1.2 Cloudflare Workers をはじめよう

1-2 何に使えるのか（4 分）

午後のプロトタイプ開発で使える形は、おおむね次の 3 つです。

1. **API の中継（プロキシ）**
   ブラウザから直接叩けない API を、Worker 越しに叩きます。CORS が許可されていない API、API キーを画面に出したくない場合に効きます。データ連携基盤（DSPF）の API やオープンデータの取得は、たいていこの形になります。**今日はここの入口までやります。**

2. **作った画面の置き場**
   Day2 で作った `handson-viz` のような画面ごと Worker に載せて、URL ひとつで配れます。「見せたいときにノート PC を開いておく」必要がなくなります。

3. **定期実行**
   Cron Triggers を使うと「毎朝 6 時にデータを取りに行って保存する」といった処理を、サーバを立てずに回せます。

> **今日 Hono をやらない理由**
> 60 分では素の Workers（`fetch` ハンドラひとつ）までにとどめます。URL のパターンが増えて分岐が面倒になってきたら Hono を入れる、というのが本の第 2 章です。順番を逆にすると「何が Workers で何が Hono か」が分からなくなります。

> 📖 **本のどこ**
> 1.3 作って学ぶ Cloudflare Pages ／ 2.1 Hono ではじめる Cloudflare Workers ／ 4.2 Cron Triggers で定期実行

> ⚠ **ここで質問を溜めないでください**
> 概念の疑問は、手を動かしたあとのほうが解けます。分からないまま Step 2 に進んで構いません。手を動かして残った疑問を、本の該当章で拾ってください。

2

Hello, World | Worker を作る

OpenCode に日本語で指示して、**Worker** のひな形を作ります。
Day2 とまったく同じやり方です。
作ったら、まずローカル（**wrangler dev**）で動かして確認します。

2　Hello, World｜Worker を作る

2-1 プロジェクトを作る（8 分）

目的：Worker のひな形を作る。作成先はリポジトリ直下の `hello-worker/` とします。

▌OpenCode に貼るプロンプト

```
Cloudflare Workers の Hello World プロジェクトを作って。

【条件】
- 作成先はリポジトリ直下の hello-worker/
- npm create cloudflare@latest を使う
- テンプレートは「Hello World example」→「Worker only」→ JavaScript
- 対話の質問には次で答えて: git は Yes、デプロイは No（あとで自分でやる）
- 作成が終わったら、生成された wrangler.jsonc と src/index.js の中身を表示して、
  それぞれが何をしているか 3 行以内で説明して
- 最後に、私が次に打つコマンドを 1 つだけ教えて
```

✓ 完了条件

- `hello-worker/` に `wrangler.jsonc` と `src/index.js` ができている
- `fetch` ハンドラが何をしているか、言葉で説明できる

⚠ OpenCode がうまく進まないとき

`npm create cloudflare@latest` は矢印キーで選ぶ**対話形式**のため、OpenCode 経由だと途中で止まることがあります。そのときは待たずに、**次のページの「手打ちで進める場合」に切り替えてください**。ここで粘ると時間が足りなくなります。

この資料の読み方

Step 2 以降は、**OpenCode に任せるルート**と**自分でコマンドを打つルート**を併記しています。どちらか一方をやれば同じ結果になります。詰まったら遠慮なく切り替えてください。

2-1　〈別ルート〉手打ちで進める場合

OpenCode を使わず、自分でコマンドを打って同じものを作ります。

terminal（リポジトリ直下で実行）

```
$ npm create cloudflare@latest -- hello-worker
```

▌聞かれる質問と、答え方

矢印キーで選んで Enter です。上から順に次のとおり答えます。

質問	選ぶもの
What would you like to start with?	Hello World example
Which template would you like to use?	Worker only
Which language do you want to use?	JavaScript
Do you want to use git for version control?	Yes
Do you want to deploy your application?	**No**（デプロイは Step 3 で自分でやります）

✓ 完了条件

- `hello-worker/` に `wrangler.jsonc` と `src/index.js` ができている

生成されるコードはこれだけ

```
// hello-worker/src/index.js
export default {
  async fetch(request, env, ctx) {
    return new Response("Hello World!");
  },
};
```

「アクセスが来たら `fetch` が呼ばれ、`Response` を返す」。それだけです。Workers のコードは、規模が大きくなっても原理的にはずっとこの形のままです。

create-cloudflare は更新が速く、選択肢の文言が変わることがあります。迷ったら「いちばん単純そうなもの」を選んでください。`wrangler.jsonc` と `src/index.js` ができていれば先に進めます。

📖 本のどこ

1.2 Cloudflare Workers をはじめよう

2-2　ローカルで動かす（4 分）

terminal

```
$ cd hello-worker
$ npx wrangler dev
```

Day2 と同じく、Codespaces の **Ports** タブに **8787** が現れます。地球儀のアイコンから開くと、ブラウザに `Hello World!` と表示されます。

✓ 完了条件

- ブラウザに Hello World! が表示される

2-3　表示を変えてみる（2 分）

▌OpenCode に貼るプロンプト

```
src/index.js のレスポンスを「こんにちは、仙台！」に変えて。
日本語が文字化けしないように、必要ならヘッダも直して。
wrangler dev は動かしたままでよいか、再起動が必要かも教えて。
```

▌手打ちで進める場合

エディタで `hello-worker/src/index.js` を開き、次の内容に書き換えて保存します。

```
export default {
  async fetch(request, env, ctx) {
    return new Response("こんにちは、仙台！", {
      headers: { "Content-Type": "text/plain; charset=utf-8" },
    });
  },
};
```

`wrangler dev` は保存すると自動で反映されます。ブラウザを再読み込みするだけで表示が変わります。

✓ 完了条件

- ブラウザの表示が「こんにちは、仙台！」に変わる（文字化けしていない）

⚠ うまくいかないとき

- 8787 が開かない → Ports タブで地球儀アイコンをクリック。Day2 の `wrangler dev` が残っていると別ポートになります。
- 日本語が文字化けする → `charset=utf-8` の指定漏れです。

3

本丸 | 世界に公開する

ここが今日の山場です。
コマンド **1** つで、自分だけの **URL** が世界に公開されます。
Codespace を止めても、その **URL** は生き続けます。

3-1 デプロイする（8 分）

terminal（別ターミナル。wrangler dev は動かしたままでよい）

```
$ cd hello-worker
$ npx wrangler deploy
```

初回は **workers.dev のサブドメイン名**を聞かれることがあります（例：`kazuki-sendai`）。一度決めると、以後あなたが作る Worker はすべて次の形の URL になります。

```
https://<Worker 名>.<あなたのサブドメイン>.workers.dev

例）https://hello-worker.kazuki-sendai.workers.dev
```

✓ 完了条件

- Codespaces ではなく**手元のブラウザ**で、表示された URL が開ける

⚠ デプロイ直後に 523 エラーが出たら

初回デプロイでは、DNS が行き渡るまでの間だけ 523 エラーになることがあります。故障ではありません。1〜2 分待って再読み込みしてください。

3-2 URL を交換する（4 分）

1. 表示された URL をコピーする
2. 隣の人と URL を交換して、お互いのページを開く
3. **Codespace を停止してから**、もう一度その URL を開く（スマホからでも構いません）

昨日との違いを、ここで体感してください

Day2 の Quick Tunnel は「自分の Codespace が動いている間だけ」見えるものでした。いま手元にある URL は、Codespace を止めても、PC を閉じても、そのまま生きています。**これが「サーバを持たずに公開する」ということです。**

▌OpenCode に貼るプロンプト（詰まったとき）

```
wrangler deploy が失敗する。実行したコマンドと出力は次の通り。
（ここに貼る）
原因と、次に打つコマンドを 1 つだけ教えて。
```

4

あと一歩 | **JSON** を返す

文字列の次は **JSON**。
ここまでやると、午後のプロトタイプ開発で
そのまま使える「部品」になります。

4　あと一歩｜JSON を返す

4-1 JSON を返すように変える（8 分）

▌OpenCode に貼るプロンプト

```
hello-worker の Worker を、JSON を返すように変えて。

【要件】
1. / にアクセスすると次の JSON を返す
   { "message": "こんにちは、仙台！", "time": "（現在時刻の ISO8601）" }
2. Content-Type は application/json; charset=utf-8
3. ブラウザの JavaScript から呼べるように CORS ヘッダを付ける
   （Access-Control-Allow-Origin: *）
4. /health にアクセスすると { "status": "ok" } を返す
5. それ以外のパスは 404 と { "error": "not found" } を返す

変更後、wrangler dev で確認する手順と、デプロイのコマンドを教えて。
```

OpenCode を使わない場合は、次のページのコードをそのまま `src/index.js` に貼り付けてください。

terminal

```
$ curl http://localhost:8787/
$ curl http://localhost:8787/health
$ npx wrangler deploy
```

✓ 完了条件

- 公開 URL の末尾に `/health` を付けると `{"status":"ok"}` が返る

なぜ CORS ヘッダを付けるのか

Day2 で作ったようなブラウザ上の画面から、この Worker を `fetch()` で呼べるようにするためです。ブラウザから直接叩けない外部 API を、この Worker を間に挟んで叩く——それが Step 1-2 で挙げた「① API の中継」の正体です。

4　あと一歩｜JSON を返す

4-1　〈別ルート〉手打ちで進める場合

`hello-worker/src/index.js` を、次の内容にまるごと置き換えて保存します。

▌src/index.js（全文・コピペ用）

```
const HEADERS = {
  "Content-Type": "application/json; charset=utf-8",
  "Access-Control-Allow-Origin": "*",
};

function json(data, status = 200) {
  return new Response(JSON.stringify(data, null, 2), { status, headers: HEADERS });
}

export default {
  async fetch(request, env, ctx) {
    const { pathname } = new URL(request.url);

    if (pathname === "/") {
      return json({
        message: "こんにちは、仙台！",
        time: new Date().toISOString(),
      });
    }

    if (pathname === "/health") {
      return json({ status: "ok" });
    }

    return json({ error: "not found" }, 404);
  },
};
```

読み方のヒント

増えたのは「**URL のパスを見て、返すものを変える**」という分岐だけです。`fetch` が呼ばれて `Response` を返す、という骨格は Step 2 のときから何も変わっていません。

⚠ 貼り付けたのに反映されない

保存できているか（タブに ● が残っていないか）を確認してください。それでも変わらないときは `wrangler dev` を Ctrl+C で止めて、もう一度起動します。

4-2 任意：オープンデータを中継してみる（4 分）

目的：Worker を「データの取り口」にする。ここまで来ると、午後そのまま使えます。

▌OpenCode に貼るプロンプト（任意）

```
/opendata というパスを追加して、CKAN のオープンデータ API の
データセット検索結果を JSON で返すようにして。
検索キーワードはクエリパラメータ ?q= で受け取る。
CORS ヘッダは付けたままにして。
まず wrangler dev で動かして、curl で確認する方法を教えて。
```

▌手打ちで進める場合（/health の分岐の下に足す）

```
if (pathname === "/opendata") {
  const q = new URL(request.url).searchParams.get("q") ?? "人口";
  const upstream =
    "https://data.bodik.jp/api/3/action/package_search?rows=5&q=" +
    encodeURIComponent(q);
  const res = await fetch(upstream);
  return json(await res.json());
}
```

terminal

```
$ curl "http://localhost:8787/opendata?q=人口"
```

✓ 完了条件

- 公開 URL に `/opendata?q=人口` を付けると、検索結果の JSON が返る

ここで起きていること

Worker の中から `fetch()` で外部 API を呼び、その結果に自分の CORS ヘッダを付けて返しています。これだけで「ブラウザから直接は叩けない API」が、自分の画面から使えるようになります。（例として BODIK の CKAN API を使っています。別のポータルを使いたいときは URL を差し替えてください。Day2 の ckan-open-data と同じ CKAN 系の API です。）

午後に向けて：今日できた「部品」

この 60 分でできたものを整理すると、午後のプロトタイプ開発に持ち込める形はこうなります。

層	今日までにできたもの	Day
画面	ブラウザで動くダッシュボード	Day2（handson-viz）
API（裏側）	JSON を返す Worker・外部 API の中継	Day3（hello-worker）
データ	データ連携基盤・仙台市オープンデータ	Day1〜Day3
公開手段	workers.dev の恒常 URL	Day3

チームで作るときのコツ

画面担当と API 担当を分けると並行して進められます。API 側を先に「固定の JSON を返すだけ」で公開してしまえば、画面担当はデータが揃う前から作り始められます。

今日つまずいた人へ

Step 3（デプロイ）まで行けていれば十分です。Step 4 は付録のコードを貼るだけで再現できるので、午後の作業の合間や、帰ってからでも追いつけます。

> 📖 本のどこ
>
> 2.1 Hono ではじめる Cloudflare Workers（パスの分岐が増えてきたら、ここを読んでください）

安心のための 3 つ

カード登録	**不要です。**無料プラン（Free）のまま、今日の内容はすべて動きます。研修中にカード情報を求められる操作はありません。
無料枠	Workers は **10 万リクエスト／日**まで無料（UTC 0 時にリセット）。研修や個人の試作で使い切ることはまずありません。上限に達しても**自動で課金されることはなく**、その日は一時的にエラー応答になるだけです。
作れる数	1 アカウントあたり Worker は 100 個まで。研修で作った 1 個は、そのまま残しておいて問題ありません。

研修で作ったものを消したいとき

terminal

```
$ cd hello-worker
$ npx wrangler delete
```

ダッシュボードの **Compute (Workers)** から該当の Worker を選んで削除することもできます。

> ⚠ 消す前に
>
> 公開 URL は、削除すると同じものは戻りません。午後のプロトタイプ開発で使う予定があるなら、残しておいてください。

Codespace について

GitHub の無料アカウントでも Codespaces は月 60 時間まで無料です（2 コアマシンの場合）。使い終わったら、GitHub の Codespaces 画面から **Stop** しておくと時間を節約できます。**Codespace を止めても、デプロイ済みの Worker は動き続けます。**

> ここが今日いちばん大事なところ
>
> 「開発環境（Codespace）」と「動いているサービス（Worker）」は別物です。手元の環境を消しても、公開したものは消えない——この分離が分かれば、今日の 60 分は成功です。

完成です！お疲れ様でした！

振り返りチェック

- [ ] Workers は「アクセスが来たら fetch が動く」だけの仕組みだと分かった
- [ ] wrangler dev（ローカル）と wrangler deploy（公開）の違いが分かった
- [ ] 自分の workers.dev の URL を持てた
- [ ] Day2 の一時 URL と Day3 の恒常 URL の違いを、人に説明できる
- [ ] （推奨）JSON を返す API を公開できた
- [ ] 無料枠と、消し方が分かった

時間切れ・あとから再開

止まった場所	再開方法
ログインまで	Step 2-1 のプロンプトから
プロジェクトはできた	Step 2-2 の `npx wrangler dev` から
ローカルは動く	Step 3-1 の `npx wrangler deploy` だけ
デプロイまで完了	Step 4-1 の JSON プロンプトから
JSON まで完了	Step 4-2（オープンデータ中継）を宿題に
全部終わった	本の第 2 章（Hono）へ。午後のプロトタイプで実際に使う

> **⚠ 全体が詰まったとき用プロンプト**
>
> いま `hello-worker` で（やろうとしていること）をしたいが、うまくいかない。
> 実行したコマンドと出力は次の通り。
> （ここに貼る）
> 現在の状態を確認するコマンドと、次に打つコマンドを 1 つだけ教えて。

> **このあと（午後のプロトタイプ開発へ）**
>
> 今日作った Worker は、そのままチームのプロトタイプの「API 側」として使えます。画面（Day2 の可視化）＋ API（Day3 の Worker）＋ データ（DSPF・オープンデータ）——この 3 つが揃いました。

付録 | プロンプト早見表（全文・コピペ用）①

各 Step で OpenCode に貼るプロンプトを省略せずに掲載しています。そのままコピーして使えます。

▌Step 0-B | ログインで詰まったとき

```
wrangler login が完了しない。実行したコマンドと出力は次の通り。
（ここに貼る）
Codespaces から Cloudflare にログインする方法を、
次に打つコマンド 1 つだけ教えて。
```

▌Step 2-1 | プロジェクト作成

```
Cloudflare Workers の Hello World プロジェクトを作って。

【条件】
- 作成先はリポジトリ直下の hello-worker/
- npm create cloudflare@latest を使う
- テンプレートは「Hello World example」→「Worker only」→ JavaScript
- 対話の質問には次で答えて: git は Yes、デプロイは No（あとで自分でやる）
- 作成が終わったら、生成された wrangler.jsonc と src/index.js の中身を表示して、
  それぞれが何をしているか 3 行以内で説明して
- 最後に、私が次に打つコマンドを 1 つだけ教えて
```

▌Step 2-3 | 表示を変える

```
src/index.js のレスポンスを「こんにちは、仙台！」に変えて。
日本語が文字化けしないように、必要ならヘッダも直して。
wrangler dev は動かしたままでよいか、再起動が必要かも教えて。
```

▌Step 3-1 | デプロイで詰まったとき

```
wrangler deploy が失敗する。実行したコマンドと出力は次の通り。
（ここに貼る）
原因と、次に打つコマンドを 1 つだけ教えて。
```

付録｜プロンプト早見表（全文・コピペ用）②

▌Step 4-1｜JSON を返す

hello-worker の Worker を、JSON を返すように変えて。

【要件】
1. / にアクセスすると次の JSON を返す
   { "message": "こんにちは、仙台！", "time": "（現在時刻の ISO8601）" }
2. Content-Type は application/json; charset=utf-8
3. ブラウザの JavaScript から呼べるように CORS ヘッダを付ける
   （Access-Control-Allow-Origin: *）
4. /health にアクセスすると { "status": "ok" } を返す
5. それ以外のパスは 404 と { "error": "not found" } を返す

変更後、wrangler dev で確認する手順と、デプロイのコマンドを教えて。

▌Step 4-2｜オープンデータを中継する（任意）

/opendata というパスを追加して、仙台市のオープンデータ（CKAN）の
データセット検索結果を JSON で返すようにして。
検索キーワードはクエリパラメータ ?q= で受け取る。
CORS ヘッダは付けたままにして。
まず wrangler dev で動かして、curl で確認する方法を教えて。

▌全体が詰まったとき

いま hello-worker で（やろうとしていること）をしたいが、うまくいかない。
実行したコマンドと出力は次の通り。
（ここに貼る）
現在の状態を確認するコマンドと、次に打つコマンドを 1 つだけ教えて。

▌発展｜Day2 の可視化を Worker に載せる（宿題）

Day2 で作った handson-viz を、Cloudflare Workers で配信できるようにして。
静的ファイルの配信方法（assets の設定）を wrangler.jsonc に追加し、
wrangler dev で確認してから wrangler deploy するまでの手順を教えて。

ターミナルで使う主なコマンド

```
npx wrangler login --device      # ログイン（Codespaces ではこれが確実）
npx wrangler whoami              # ログイン状態の確認
npm create cloudflare@latest     # プロジェクト作成（OpenCode に任せてもよい）
npx wrangler dev                 # ローカル起動（ポート 8787）
npx wrangler deploy              # 公開（workers.dev の URL が発行される）
npx wrangler delete              # 公開した Worker を削除
npx wrangler logout              # ログアウト（共用 PC を使ったときは必ず）
```

▌手打ちルートのコード掲載ページ

Step	内容	ページ
2-1	プロジェクト作成の質問と答え	p.12
2-3	表示を変える src/index.js	p.13
4-1	JSON を返す src/index.js（全文）	p.18
4-2	/opendata の追加分	p.19

本の章マップ

今日やったこと／発展	本の対応箇所
Step 1　座学	1.1 Cloudflare Workers とは
Step 2　Hello, World	1.2 Cloudflare Workers をはじめよう
Step 3　デプロイ	1.2 Cloudflare Workers をはじめよう
Step 4　JSON・パスの分岐	2.1 Hono ではじめる Cloudflare Workers
発展　画面を配る	1.3 作って学ぶ Cloudflare Pages
発展　アプリらしくする	2.2 Hono で作る短文投稿 Web アプリ
発展　定期実行	4.2 Cron Triggers で定期実行
発展　データを持つ	5.1 Hyperdrive ／ 5.2 Durable Objects
発展　生成 AI を組み込む	6.1 Cloudflare における生成 AI 活用 ／ 6.2 Agents SDK

今日の内容は第 1 章にあたります。次に読むなら第 2 章（Hono）。パスの分岐が 3 つを超えたあたりで素の `fetch` ハンドラは書きづらくなるので、そのタイミングで読むと Hono が何を解決しているのか腹落ちします。

参考リンク

- 配布書籍『Web開発者のための［入門］Cloudflare Workers』（技術評論社、2026年2月刊）
- Cloudflare Workers 公式ドキュメント：`https://developers.cloudflare.com/workers/`
- 公式チュートリアル動画：`https://www.youtube.com/watch?v=H7Qe96fqg1M`
- Day2 手順書（OpenCode ハンズオンマニュアル）
- リポジトリ README（`kzkski/cloudflare-dev-container`）

後日メニュー（この資料では扱わない）

- Hono でのルーティングとテンプレート（本 2.1〜2.2）
- KV / D1 でデータを保存する
- Cron Triggers で定期実行（本 4.2）
- 独自ドメインの割り当て
- Workers AI / Agents SDK（本 6 章）

3 日間でやったこと

Day	内容	持ち帰るもの
Day1	Tableau で人流データを可視化	データを「見る」目線
Day2	OpenCode で可視化を作り、一時 URL で共有	AI に作らせる進め方
Day3	Cloudflare Workers で恒常 URL に公開	自分で公開できる手段

> **この 3 日間で変わること**
> 「見る」「作る」「公開する」が、すべて自分の手の中で完結するようになりました。あとは題材を、業務の中の困りごとに置き換えるだけです。