生成AI時代のドキュメント基盤

ドキュメンテーション戦略

Documentation Team

目次

1. 生成AI時代のドキュメント基盤

このリポジトリは、以下の2つをサンプルとして提供する。

・**ドキュメントを記述する基盤**（Markdown + Mermaid + Draw.io + textlint + MkDocs）

・**記述した文書を公開する仕組み**（GitHub Pages / Azure Blob Storage静的サイト）

採用している技術の一覧は 技術スタック を参照する。

1.1 3つの実行環境

用途に応じて、次の3形態でドキュメント執筆環境を利用できる。

形態	想定用途	前提
ローカル（Linux / WSL）	普段の執筆・プレビュー	`mise` でツールを揃える
DevContainer	VS Code で隔離環境を使いたい	Docker + VS Code + Dev Containers拡張
code-server	ブラウザだけでハンズオンしたい	Docker（ローカル）または Azure（配布）

1.2 ローカル環境（Linux / WSL）

`mise.toml` でPython / uv / Node.js / pnpmのバージョンを固定している。以下の手順でツールを揃える。

1.2.1 1.mise のインストール

```
curl https://mise.run | sh
echo 'eval "$(~/.local/bin/mise activate bash)"' >> ~/.bashrc
exec "$SHELL"
```

1.2.2 2.依存関係のインストール

Debian / Ubuntu系ではまずWeasyPrintやPlaywrightのネイティブ依存（Chromium、Pango、Cairo、日本語フォントなど）をaptで導入する。`sudo` が必要で、初回のみ実行する。

```
mise run setup-system
```

続いてランタイムとPython / Nodeの依存関係をまとめて取得する。

```
mise run setup      # mise install / uv sync / Playwright / pnpm install をまとめて実行
```

macOSやWindows、その他のディストリビューションでは `setup-system` は対象外である。該当環境ではネイティブ依存を個別に導入する必要があるため、DevContainerまたはcode-serverの利用を推奨する。

1.2.3 3.ドキュメントのプレビュー

```
pnpm mkdocs         # http://127.0.0.1:8000 でライブプレビュー
```

1.3 DevContainer

VS CodeのDev Containers拡張を導入し、リポジトリを開いた際に表示される「Reopen in Container」を選ぶと、`.devcontainer/devcontainer.json` が `.devcontainer/Dockerfile` をビルドして起動する。

当DockerfileはGHCRに公開済みの下記ハンズオンイメージをベースにしており、ビルドはベースイメージのpullのみで完了するためホスト側に `mise` やPythonを入れずに同じツール構成で作業できる。

- ghcr.io/genai-docs/genai-docs-env:latest

ポート8000は自動フォワードされ、MkDocsライブプレビューにブラウザでアクセスできる。

1.4 code-server（ブラウザ版 VS Code）

ハンズオン配布や、ローカルに開発ツールを入れたくない場合に使う。実行環境（Dockerイメージ定義、起動・ Azureデプロイスクリプト、Bicep、CI）は別リポジトリ genai-docs/genai-docs-env に切り出している。該当リポジトリで `nr run:local` / `nr run:remote` / `nr azure:deploy` を実行する。

当リポジトリのDevContainerは同イメージ `ghcr.io/genai-docs/genai-docs-env:latest` をベースとして使うため、イメージ更新はenvリポジトリ側で完結する。

Azure上に共有ハンズオン環境を展開する手順は実行環境にまとめている。

1.5 詳しくは

- ユーザーガイド
- 実行環境の概要
- アーキテクチャー
- サンプル一覧

2. ユーザーガイド

本ドキュメントでは、ドキュメント基盤の日常的な利用方法を説明する。コマンドは `package.json` の `scripts` を `@antfu/ni` の `nr` でラップして実行する前提で記述する。

2.1 ni / nr とは

`@antfu/ni` は、リポジトリのロックファイル（`pnpm-lock.yaml` / `package-lock.json` / `yarn.lock`）から実パッケージマネージャーを自動判別して委譲するユーティリティである。本リポジトリではpnpmが検出されるため、`nr <script>` は `pnpm run <script>` と等価になる。

エイリアス	役割	本リポジトリでの実体
`ni`	依存関係のインストール	`pnpm install`
`nr`	`package.json` の script 実行	`pnpm run <script>`
`nlx`	ワンショット実行	`pnpm dlx <pkg>`

通常は `mise run setup` で依存関係の導入は完了するため、個別の `ni` は不要である。

2.2 文書記述

MkDocsを起動し、ライブリロードのプレビューで内容を確認しながらMarkdownを書く。

```
nr mkdocs           # http://127.0.0.1:8000 でライブプレビュー
```

2.3 Pull Request 作成前チェック

レビュー前に本番相当のビルドと品質チェックを通しておく。

```
nr mkdocs:build     # Mermaid の SVG/PNG 変換と PDF 出力を含む本番相当ビルド
nr lint:text        # textlint（表記ゆれ・用字用語などのチェック）
nr lint:text:fix    # textlint 自動修正（機械的に直せるものだけ）
```

`mkdocs:build` は `RENDER_SVG=1 RENDER_PNG=1 ENABLE_PDF=1` を有効にした上で `uv run mkdocs build` を実行する。PDFは `site/pdf/ドキュメンテーション戦略.pdf` に出力される。

2.4 スライド作成

Marpスライドのプレビューと PDF生成は次のとおり。

```
nr marp             # 変更監視つきプレビュー（docs/スライド）
nr marp:build       # docs/スライド/dist に PDF を出力
```

2.5 実行コマンドの整理

本リポジトリのコマンドは2系統に分かれており、役割で使い分ける。

系統	呼び出し方	用途
`package.json` scripts	`nr <script>`	ドキュメントのプレビュー／ビルド／品質チェック／スライド
`mise.toml` tasks	`mise run <task>`	ローカル開発環境のセットアップ（ランタイム・OS 依存の導入）

```
mise run setup-system   # Debian/Ubuntu のシステム依存導入（初回のみ、sudo）
mise run setup          # Python / Node 依存とランタイム導入
```

ハンズオン環境のDockerイメージビルドやAzureデプロイは別リポジトリ genai-docs/genai-docs-env に分離している。該当作業はenvリポジトリ側で実行する。

3. 技術スタック

このリポジトリで採用している主要な技術と、その用途をまとめる。

3.1 ドキュメント生成

技術	用途
Python + uv	MkDocs の実行環境と依存関係管理
MkDocs + Material for MkDocs	静的サイトジェネレーター
MkDocs プラグイン群	Mermaid の SVG/PNG 変換、PDF 出力、表読込
WeasyPrint	PDF 生成
Noto CJK フォント	PDF での日本語レンダリング

3.2 図表・スライド

技術	用途
Mermaid	Markdown 内での図表作成
Draw.io	SVG 図表作成
Marp	Markdown スライド作成
Chromium	Mermaid 変換と Marp のレンダリングに使用するブラウザエンジン

3.3 品質チェック

技術	用途
Node.js + pnpm	Marp と textlint の実行基盤
textlint	ドキュメント品質チェック

3.4 開発・ハンズオン環境

技術	用途
Docker	devcontainer とハンズオン環境のコンテナー基盤
code-server	ブラウザ版 VS Code（ハンズオン環境の IDE 本体）
VS Code 拡張機能	Markdown / Mermaid / Draw.io / Marp / textlint などの執筆支援

4. 実行環境

4.1 実行環境

このセクションでは、`MkDocs` サンプルを支える実行環境を説明する。

実行環境（Dockerイメージ、Azure Container Appsスクリプト、ハンズオン配布インフラ）は別リポジトリ genai-docs/genai-docs-env に切り出して管理している。本リポジトリはサンプル本体（ドキュメント内容とDevContainer設定）に集中する。

4.1.1 役割分担

- サンプル本体（当リポジトリ）： `docs/`, `mkdocs.yml`, `pyproject.toml`, `package.json`, `mise.toml`, `.devcontainer/`
- 実行環境（genai-docs-env）： Dockerイメージ定義、Azure Container Apps運用スクリプト、Bicep、GHCR公開CI

4.1.2 ドキュメント

- クラウド環境構築 - GitHub Pages / Azure Static Web Appsの公開基盤を構築する
- ハンズオン環境設計 - Azure Container Appsを使った配布用環境の設計方針
- ハンズオン環境情報 - 実装と運用導線の対応関係

4.2 クラウド環境構築手順

4.2.1 前提条件

- Azureサブスクリプションに対するリソース作成権限があること
- GitHubリポジトリの管理者権限があること
- `az` CLIと `gh` CLIがインストール済みであること
- `az login` と `gh auth login` が完了していること

4.2.2 手順

1. 命名規則とパラメーターを設定する

GitHubのOwnerとRepository名を設定する。

PowerShell Bash

```powershell
$Owner = "<org-or-user>"
$Repository = "<repo>"
$githubRepo = "$Owner/$Repository"

$location = "japaneast"
$swaLocation = "eastasia"

$resourceGroupName = "rg-$Repository-prod"
$swaName = "stapp-$Repository-prod"
$identityName = "id-$Repository-prod"
$federatedCredentialName = "fc-github-actions-main"
```

```bash
Owner="<org-or-user>"
Repository="<repo>"
githubRepo="${Owner}/${Repository}"

location="japaneast"
swaLocation="eastasia"

resourceGroupName="rg-${Repository}-prod"
swaName="stapp-${Repository}-prod"
identityName="id-${Repository}-prod"
federatedCredentialName="fc-github-actions-main"
```

2. リソースグループと Static Web App の作成

PowerShell Bash

```powershell
az group create --name $resourceGroupName --location $location

az staticwebapp create `
  --name $swaName `
  --resource-group $resourceGroupName `
  --location $swaLocation `
  --sku Standard

$defaultHostname = az staticwebapp show `
  --name $swaName `
  --resource-group $resourceGroupName `
  --query defaultHostname `
  -o tsv
```

```bash
az group create --name $resourceGroupName --location $location

az staticwebapp create \
  --name $swaName \
  --resource-group $resourceGroupName \
  --location $swaLocation \
  --sku Standard

defaultHostname=$(az staticwebapp show \
  --name $swaName \
  --resource-group $resourceGroupName \
  --query defaultHostname \
  -o tsv)
```

3. マネージド ID と OIDC フェデレーション資格情報の作成

既定ブランチが `main` 以外の場合は `--subject` の `refs/heads/main` を置き換える。

PowerShell Bash

```powershell
az identity create --name $identityName --resource-group $resourceGroupName --location $location

$identity = az identity show `
  --name $identityName `
  --resource-group $resourceGroupName `
  --query "{clientId:clientId, principalId:principalId}" `
  -o json | ConvertFrom-Json
$clientId = $identity.clientId
$principalId = $identity.principalId

az identity federated-credential create `
  --name $federatedCredentialName `
  --identity-name $identityName `
  --resource-group $resourceGroupName `
  --issuer "https://token.actions.githubusercontent.com" `
  --subject "repo:$githubRepo:ref:refs/heads/main" `
  --audiences "api://AzureADTokenExchange"
```

```bash
az identity create --name $identityName --resource-group $resourceGroupName --location $location

clientId=$(az identity show \
  --name $identityName \
  --resource-group $resourceGroupName \
  --query clientId \
  -o tsv)
principalId=$(az identity show \
  --name $identityName \
  --resource-group $resourceGroupName \
  --query principalId \
  -o tsv)

az identity federated-credential create \
  --name $federatedCredentialName \
  --identity-name $identityName \
  --resource-group $resourceGroupName \
  --issuer "https://token.actions.githubusercontent.com" \
  --subject "repo:$githubRepo:ref:refs/heads/main" \
  --audiences "api://AzureADTokenExchange"
```

4. Static Web App への RBAC 付与

伝播待ちで失敗する場合は数十秒待って再実行する。

PowerShell Bash

```
$swaId = az staticwebapp show `
  --name $swaName `
  --resource-group $resourceGroupName `
  --query id `
  -o tsv
az role assignment create `
  --assignee-object-id $principalId `
  --assignee-principal-type ServicePrincipal `
  --role Contributor `
  --scope $swaId

swaId=$(az staticwebapp show \
  --name $swaName \
  --resource-group $resourceGroupName \
  --query id \
  -o tsv)
az role assignment create \
  --assignee-object-id $principalId \
  --assignee-principal-type ServicePrincipal \
  --role Contributor \
  --scope $swaId
```

5. GitHub Pages の有効化

GitHub Pagesにデプロイするため、リポジトリ設定で有効化する必要がある。

1. GitHubリポジトリの `Settings -> Pages` に移動する。
2. `Build and deployment` の `Source` で `GitHub Actions` を選択する。

> **Environmentの自動作成**
>
> GitHub Actionsをソースに設定すると、`github-pages` という名前のenvironmentが自動的に作成される。ワークフローはこのenvironmentを使用してデプロイを行う。

6. GitHub Discussions の有効化とカテゴリ作成

```
gh repo edit $githubRepo --enable-discussions
```

1. GitHubの `Settings -> Discussions` に移動する。
2. `Set up discussions` をクリックする（初回のみ）。
3. `Categories -> New category` をクリックする。
4. `Name: Invitation`, `Description: SWA role sync invitations`, `Format: Announcement` を設定して作成する。

7. GitHub Actions の Variables と Secrets の登録

PowerShell Bash

```powershell
$tenantId = az account show --query tenantId -o tsv
$subscriptionId = az account show --query id -o tsv
$swaApiToken = az staticwebapp secrets list `
  --name $swaName `
  --resource-group $resourceGroupName `
  --query properties.apiKey `
  -o tsv

gh variable set AZURE_CLIENT_ID --body $clientId --repo $githubRepo
gh variable set AZURE_TENANT_ID --body $tenantId --repo $githubRepo
gh variable set AZURE_SUBSCRIPTION_ID --body $subscriptionId --repo $githubRepo
gh variable set AZURE_SWA_NAME --body $swaName --repo $githubRepo
gh variable set AZURE_SWA_RESOURCE_GROUP --body $resourceGroupName --repo $githubRepo

gh secret set AZURE_SWA_API_TOKEN --body $swaApiToken --repo $githubRepo
```

```bash
tenantId=$(az account show --query tenantId -o tsv)
subscriptionId=$(az account show --query id -o tsv)
swaApiToken=$(az staticwebapp secrets list \
  --name $swaName \
  --resource-group $resourceGroupName \
  --query properties.apiKey \
  -o tsv)

gh variable set AZURE_CLIENT_ID --body "${clientId}" --repo $githubRepo
gh variable set AZURE_TENANT_ID --body "${tenantId}" --repo $githubRepo
gh variable set AZURE_SUBSCRIPTION_ID --body "${subscriptionId}" --repo $githubRepo
gh variable set AZURE_SWA_NAME --body "${swaName}" --repo $githubRepo
gh variable set AZURE_SWA_RESOURCE_GROUP --body "${resourceGroupName}" --repo $githubRepo

gh secret set AZURE_SWA_API_TOKEN --body "${swaApiToken}" --repo $githubRepo
```

8. GitHub App の作成と連携情報の登録

1. `GitHub -> Settings -> Developer settings -> GitHub Apps -> New GitHub App` を開く。
2. `App name` は任意、`Homepage URL` はリポジトリURLを入力する。
3. `Webhook` の `Active` をオフにする。
4. `Permissions -> Repository -> Discussions: Read and write` を設定する。
5. `Create GitHub App` をクリックする。
6. `App ID` を控える。
7. `Private keys -> Generate a private key` でPEMをダウンロードする。
8. 連携情報を登録する：

PowerShell Bash

```powershell
gh variable set ROLE_SYNC_APP_ID --body "<appId>" --repo $githubRepo
gh secret set ROLE_SYNC_APP_PRIVATE_KEY --body (Get-Content -Raw -Path "<pemPath>") --repo $githubRepo

gh variable set ROLE_SYNC_APP_ID --body "<appId>" --repo $githubRepo
gh secret set ROLE_SYNC_APP_PRIVATE_KEY --body "$(cat '<pemPath>')" --repo $githubRepo
```

4.2.3 完了確認

- `https://$defaultHostname` にアクセスできることを確認する。

4.3 ハンズオン実行環境 設計概要書

4.3.1 1. 背景

ハンズオン形式の研修・勉強会を実施する際、参加者のローカル環境差異が大きな障壁となる。とくにDevContainerを基盤とした開発環境は、Docker Desktop、VS Code、WSL2等の事前セットアップが必要であり、Windows・Mac・Linuxの各OSごとに手順が異なる。参加者に事前準備を求めることは、当日のトラブルシューティングに時間を取られるリスクを伴い、ハンズオン本来の学習目的を損なう。

この課題を解消するため、開催側がクラウド上に統一された実行環境を用意し、参加者はブラウザのみでアクセスできる構成を採用する。

4.3.2 2. 目的

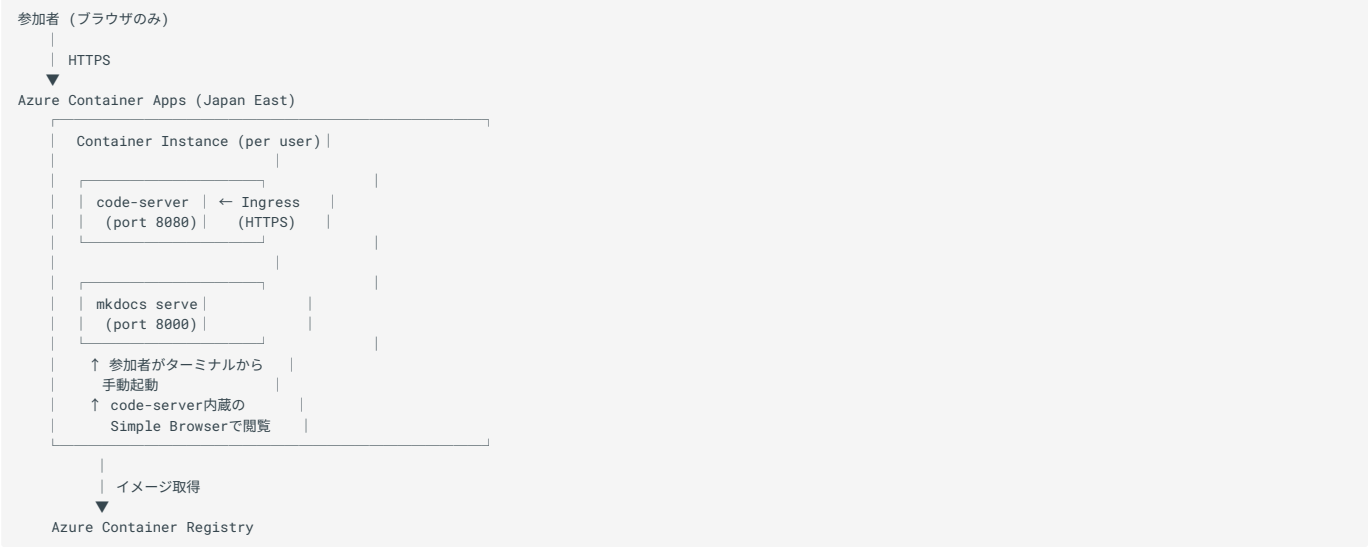
本環境は以下を実現することを目的とする。

- 参加者のローカル環境に依存しない、統一されたハンズオン実行環境の提供
- 参加者の事前準備をゼロにし、ブラウザアクセスのみで作業を開始できる状態の実現
- MkDocs等のツールをターミナルから起動し、ブラウザ上でプレビューを確認できるワークフローの提供
- Visual Studio Enterpriseサブスクリプションに付帯するAzureクレジット（$150/月）の範囲内での運用

なお、DevContainer自体の操作習得はハンズオンの目的に含まない。DevContainerはあくまで環境構築の手段として開催側が利用し、参加者はその成果物であるコンテナー環境上で作業する。

4.3.3 3. アーキテクチャ

3.1 構成概要



3.2 構成要素と役割

構成要素	役割
Azure Container Registry	ハンズオン用コンテナイメージの格納
Azure Container Apps Environment	Container Appの実行基盤（参加者全員で共有）
Azure Container Apps (個別)	参加者ごとに1インスタンスをデプロイ。code-serverを稼働させる
code-server	ブラウザ上でVS Code相当のエディター・ターミナルを提供
コンテナイメージ	DevContainerの定義をもとに構築した、ハンズオンに必要なツール・資材を含むイメージ

3.3 ネットワークとアクセス制御

- 各Container Appは外部Ingress（HTTPS）を有効にし、参加者ごとに一意のFQDNを割り当てる
- 認証はcode-serverのパスワード認証を利用し、参加者ごとに個別のパスワードを環境変数で設定する
- mkdocsのプレビュー（port 8000）はコンテナー内部でのみリッスンし、code-server内蔵のSimple Browserを通じて参加者が閲覧する。外部への追加ポート公開は行わない

3.4 コンテナイメージの構成方針

コンテナイメージには以下を含める。

- code-server（ブラウザ版VS Code）
- ハンズオン対象のツール群（MkDocs、関連プラグイン等）
- 必要なVS Code拡張機能（プリインストール）
- ハンズオン資材（ドキュメントソース、設定ファイル等）

イメージのベースには `mcr.microsoft.com/devcontainers` 系のイメージを利用し、既存のDevContainer定義との整合性を保つ。

3.5 参加者の操作フロー

1. 開催側から配布されたURL・パスワードを受け取る
2. ブラウザでURLにアクセスし、パスワードを入力する
3. code-serverのエディター・ターミナルが表示される
4. ターミナルから `mkdocs serve` 等のコマンドを実行する
5. code-server内のSimple Browserで `localhost:8000` を開き、プレビューを確認する

4.3.4 4. コスト見通し

Visual Studio Enterpriseサブスクリプションに付帯するAzureクレジット（$150/月）の範囲内での運用を前提とする。

リソース	概算
Container Apps (1vCPU / 2GBメモリ × 参加者数)	約 $0.06/時間/台
Container Registry (Basic SKU)	$5/月

20名規模・4時間のハンズオンの場合、合計 $10〜15程度を見込む。クレジット上限に対して十分な余裕がある。

4.3.5 5. 運用上の考慮事項

5.1 事前準備

- コンテナイメージのビルドとレジストリへのプッシュは、ハンズオン開催前に完了させておく
- 参加者分のContainer Appデプロイはスクリプト化し、一括で実行できるようにする
- 参加者への配布物（URL・パスワードの一覧）を事前に作成する

5.2 当日の運用

- Container Appsのmin-replicasを1に設定し、スケールインによるコンテナー停止を防止する
- 参加者がターミナルで誤ってcode-serverプロセスを停止した場合に備え、復旧手順を用意する

5.3 事後の片付け

- ハンズオン終了後、リソースグループごと削除することで全リソースを一括削除し、課金を停止する

4.3.6 6. 実装に関する注意事項

本書に含まれるDockerfile・CLIコマンド等の具体的な実装内容は参考情報である。実際のハンズオン内容・参加者数・ネットワーク要件に応じて、実装の過程で詳細を決定・評価すること。とくに以下の項目は実装段階での検証が必要である。

- コンテナーのCPU・メモリサイズの適正値（ハンズオン内容の負荷に依存）
- code-serverのバージョンおよびVS Code拡張機能の互換性
- Simple Browserによるmkdocsプレビューの表示品質と制約
- 参加者が同時アクセスした際のContainer Apps Environmentの性能
- パスワード認証の強度が研修用途として十分かの判断

4.3.7 7. 実装・設計メモ

- 開発環境情報
  - `settings.local.json`

4.4 ハンズオン実行環境

この文書は、ハンズオン実行環境の全体像と運用導線をまとめたものである。Azureリソース定義や各パラメーターの詳細は文書に重複記載せず、実装そのものを正としてBicep / スクリプトを参照する。

4.4.1 リポジトリ分離

ハンズオン実行環境（Dockerイメージ、Azure Container Apps運用スクリプト、Bicep、GHCR公開CI）は別リポジトリ genai-docs/genai-docs-env で管理している。当リポジトリはサンプル本体に集中し、実行環境の更新はenvリポジトリ側で完結する。

4.4.2 構成概要

参加者がブラウザのみで作業できるよう、Azure Container Apps上に参加者ごとの `code-server` 環境をデプロイする。コンテナイメージは `ghcr.io` に格納し、Container Appsから取得する。

```
参加者（ブラウザ）
    |
    | HTTPS + パスワード認証
    ▼
Container App（参加者ごと）
    |
    ├── code-server : 8080
    └── mkdocs serve : 8000
        ※ 参加者が必要に応じて起動
        ※ 外部公開はしない

        ↑
        | 配置先
        |
Container Apps Environment
        |
        └── ログ送信先: Log Analytics Workspace

        ↑
        | イメージ取得
        |
GitHub Container Registry (ghcr.io)
```

4.4.3 実装との対応

環境構築に関わる主要ファイルはすべて genai-docs-env に存在する。

- インフラのエントリーポイント： `scripts/Deploy-GenaiDocsEnv.ps1`
- イメージビルド： `scripts/Build-Image.ps1`
- 環境削除： `scripts/Remove-GenaiDocsEnv.ps1`
- イメージ更新（RG温存）： `scripts/Update-GenaiDocsImage.ps1`
- 共有インフラ定義： `deploy/azure/main.bicep`
- 参加者用アプリ定義： `deploy/azure/container-app.bicep`
- 実行コンテナー定義： `Dockerfile`
- デプロイ設定： `settings.local.json`

詳細なAzureリソース定義、Container Appの構成、パラメーター、命名規則、出力値はBicepを参照すること。

4.4.4 運用フロー

構築

環境構築はenvリポジトリで `nr azure:deploy -- -UserCount <人数>` を実行する。スクリプトは `settings.local.json` を読み込み、Azure上へ共有インフラと参加者用Container App群をデプロイする。コンテナーイメージは事前にCIで `ghcr.io/genai-docs/genai-docs-env:latest` として公開されている前提である。

参加者情報の出力やログは、envリポジトリ直下の `genai-docs-out/` に保存される。

当日運用

参加者は配布されたURLとパスワードで `code-server` にログインする。空のworkspaceで起動するため、本リポジトリのサンプルを利用する場合は初回に `git clone` で展開する。`mkdocs serve` の起動は参加者の操作で行い、プレビューはコンテナー内部の `localhost:8000` を利用する前提である。

片付け

環境削除はenvリポジトリで `nr azure:remove -- -All` を利用する。削除対象の判定方法や一括削除の挙動はスクリプト実装を参照すること。

4.4.5 補足

- ハンズオン配布用の実行コンテナーはenvリポジトリの `Dockerfile` で定義し、CIで `ghcr.io/genai-docs/genai-docs-env:latest` として公開している。
- ローカル開発用Dev Containerは当リポジトリの `.devcontainer/Dockerfile` で同イメージの `:latest` をベースとして使う構成である。
- 実装詳細を文書へ転記しすぎると乖離しやすいため、構成や挙動の正確な確認はコードを優先する。

5. ハンズオン準備

5.1 ハンズオン準備

配布されたcode-server（ブラウザ版VS Code）上で、ハンズオンを始める前に済ませておく一連のセットアップ手順をまとめる。

5.1.1 手順の全体像

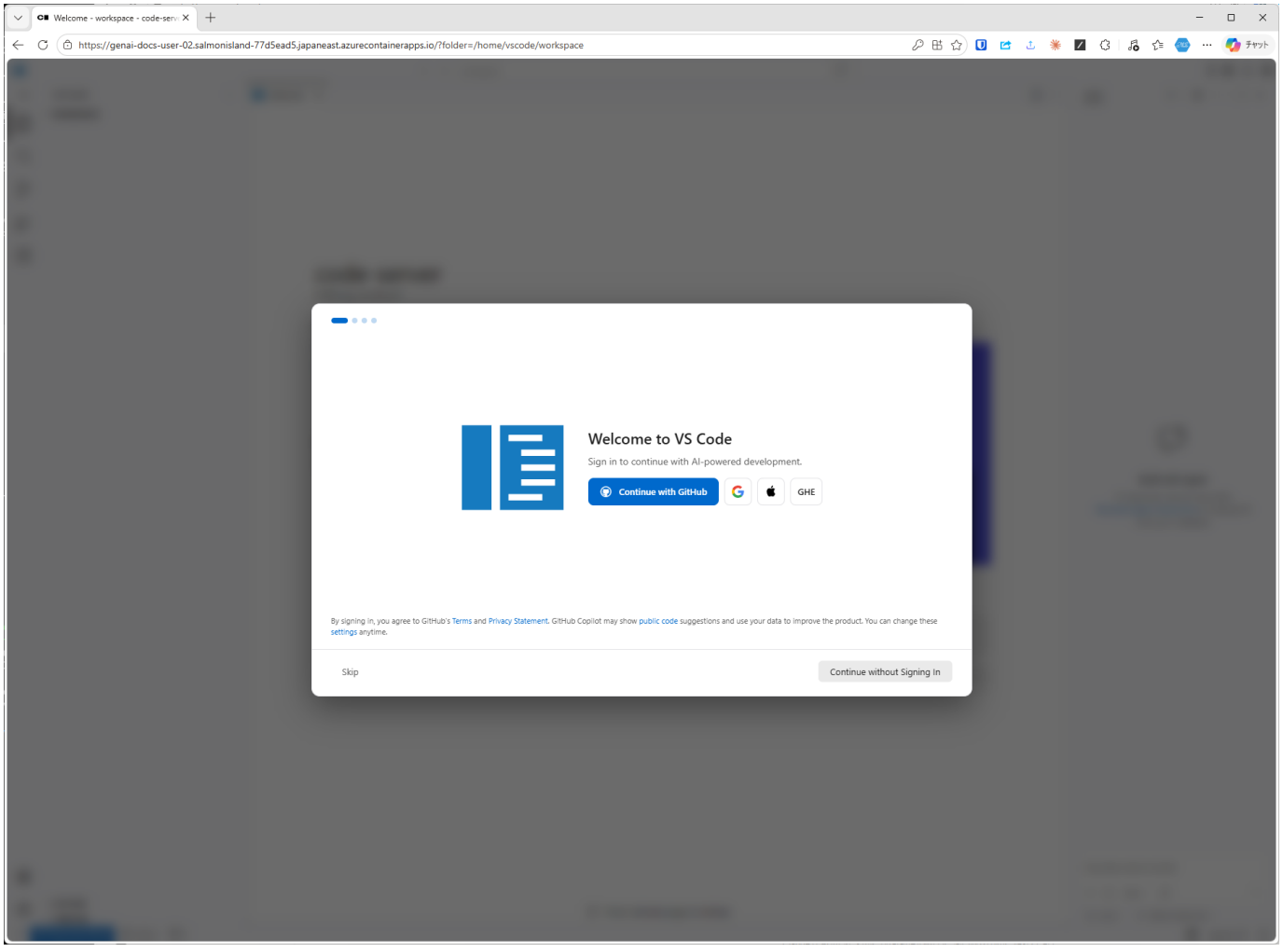
1. VS Code (code-server) の初期化 ― Welcome画面のウィザード
2. GitHub リポジトリの準備 ― Forkし、VS Code上でクローン
3. AIエージェントのサインイン ― 利用するエージェントのみ実施すればよい
    - Claude Code
    - Cursor CLI
    - Codex CLI
    - GitHub Copilot CLI

5.2 VS Code (code-server) の初期化

ブラウザでcode-serverを開くと、初回にWelcomeウィザードが走る。以下の順にクリックしていく。

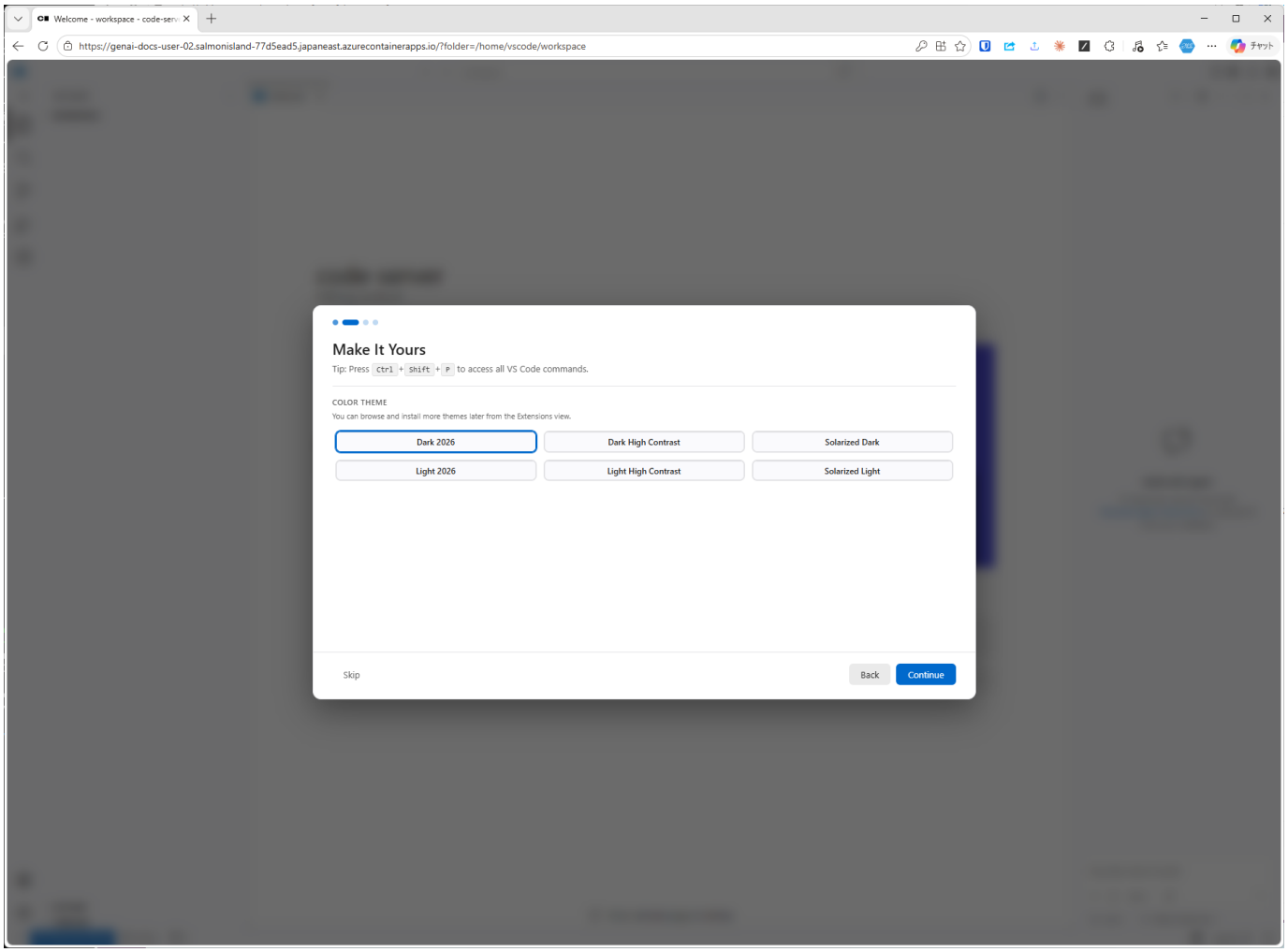
5.2.1 1. サインイン

`Continue without Signing In` で十分（AI機能は後から個別に設定する）。



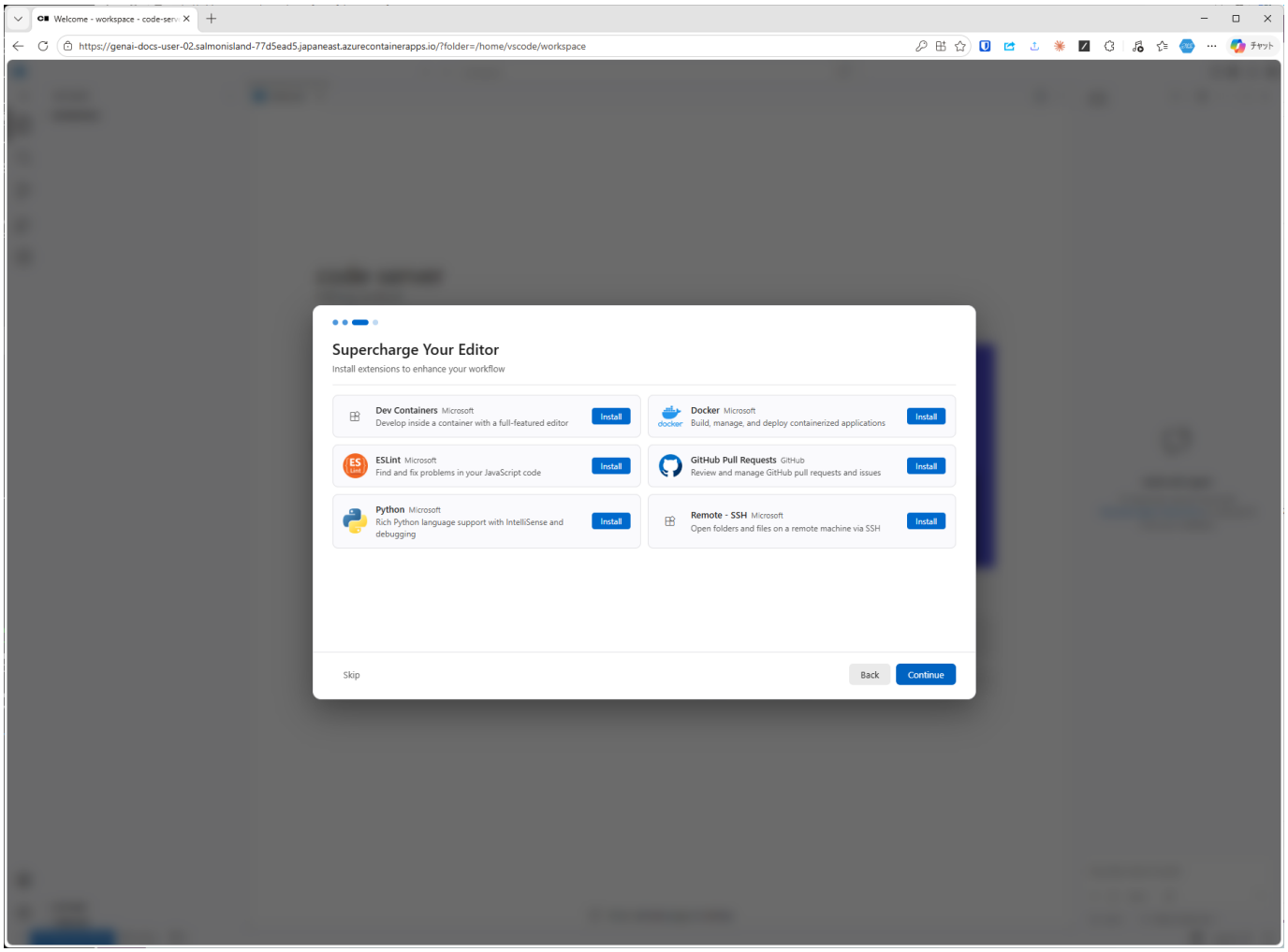
5.2.2 2. カラーテーマ

好みのテーマ（既定の `Dark 2026` など）を選び `Continue`。



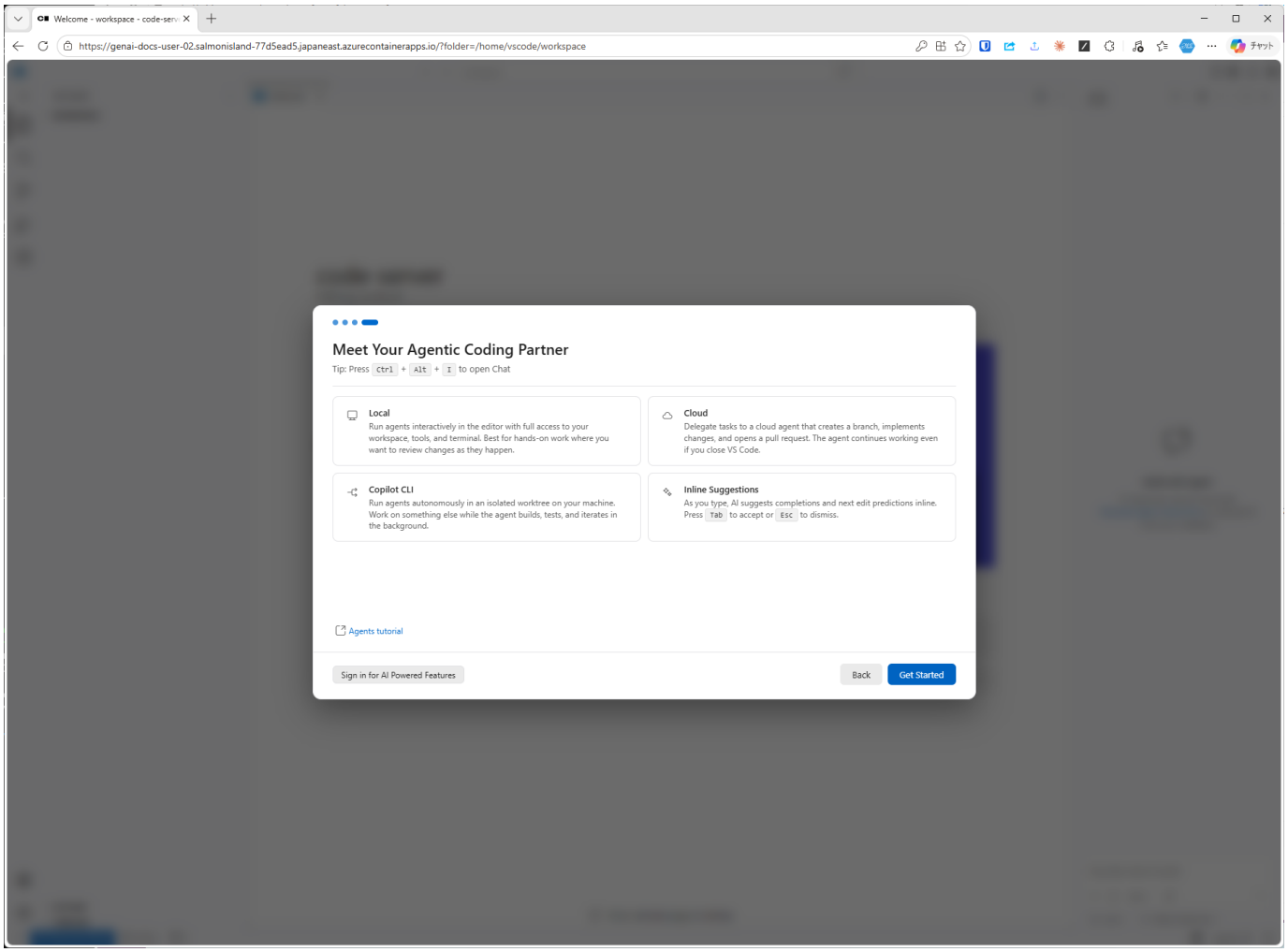
5.2.3 3. 拡張機能

必要なものだけ `Install`。未選択でも進めて構わない。



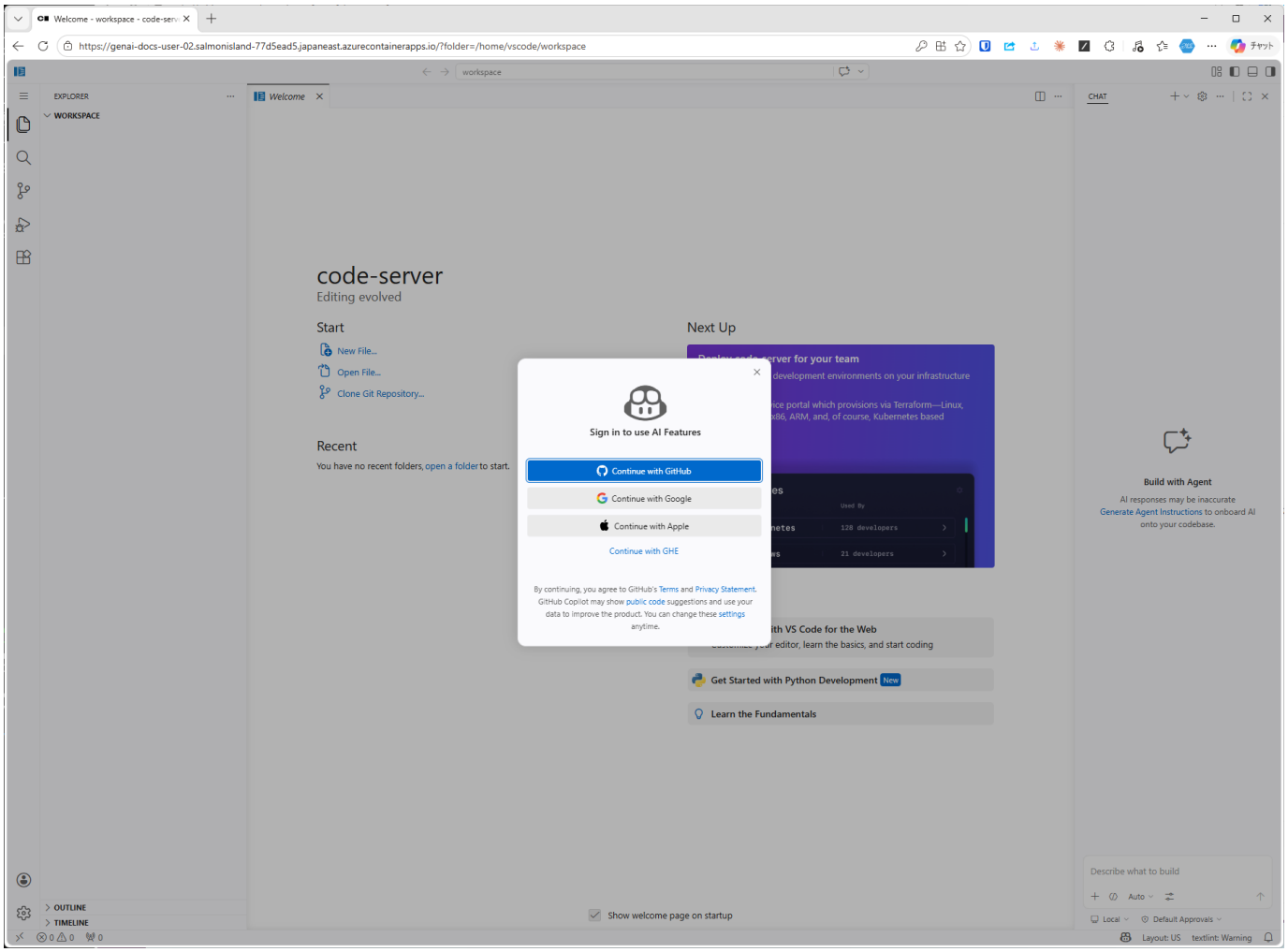
5.2.4 4. Agentic Coding の紹介

内容を確認して `Get Started`。



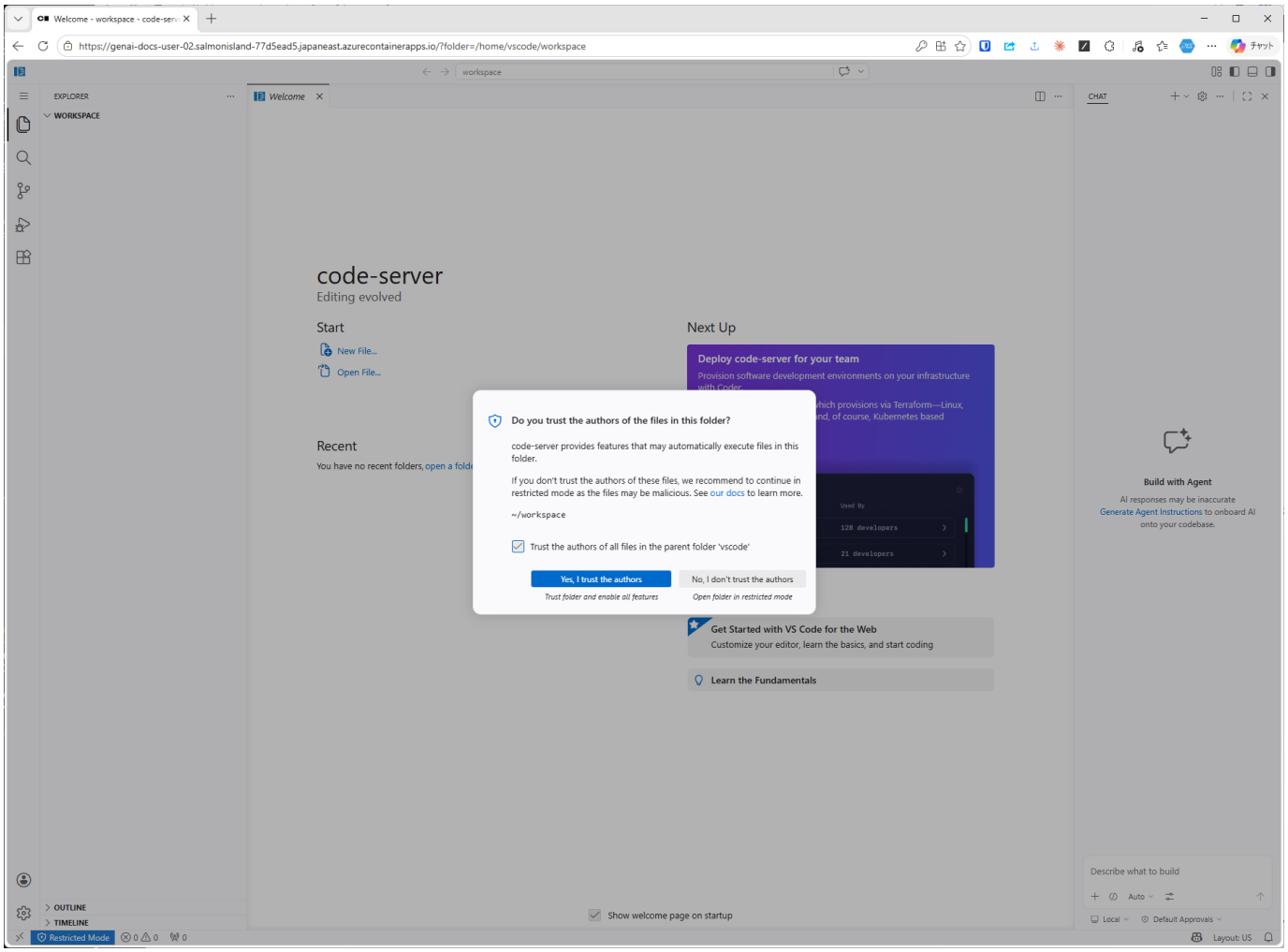
5.2.5 5. AI Features サインイン（スキップ可）

VS Code組込みのAIサインインは後続の各CLIエージェントで個別に設定するため、ここでは閉じて構わない。



5.2.6 6. フォルダーを信頼する

ワークスペースを開いた直後に信頼確認ダイアログが出る。`Yes, I trust the authors` を選ぶ。



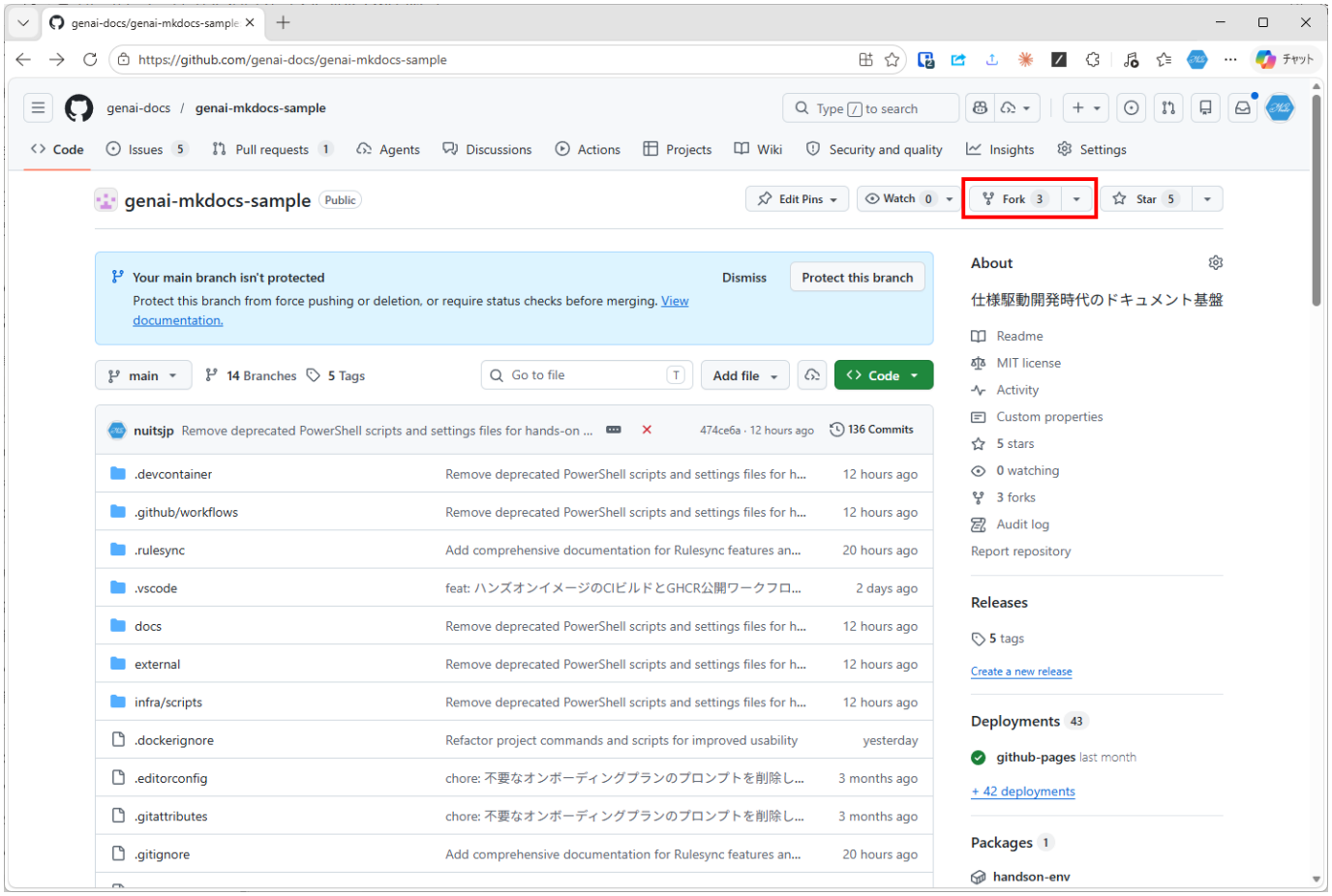
次は GitHub リポジトリの準備 に進み、ForkしたリポジトリをVS Code上でクローンする。

5.3 GitHub リポジトリの準備

`genai-docs/genai-mkdocs-sample` を自分のアカウントにForkし、そのクローンURLをVS Codeのターミナルから `git clone` する。

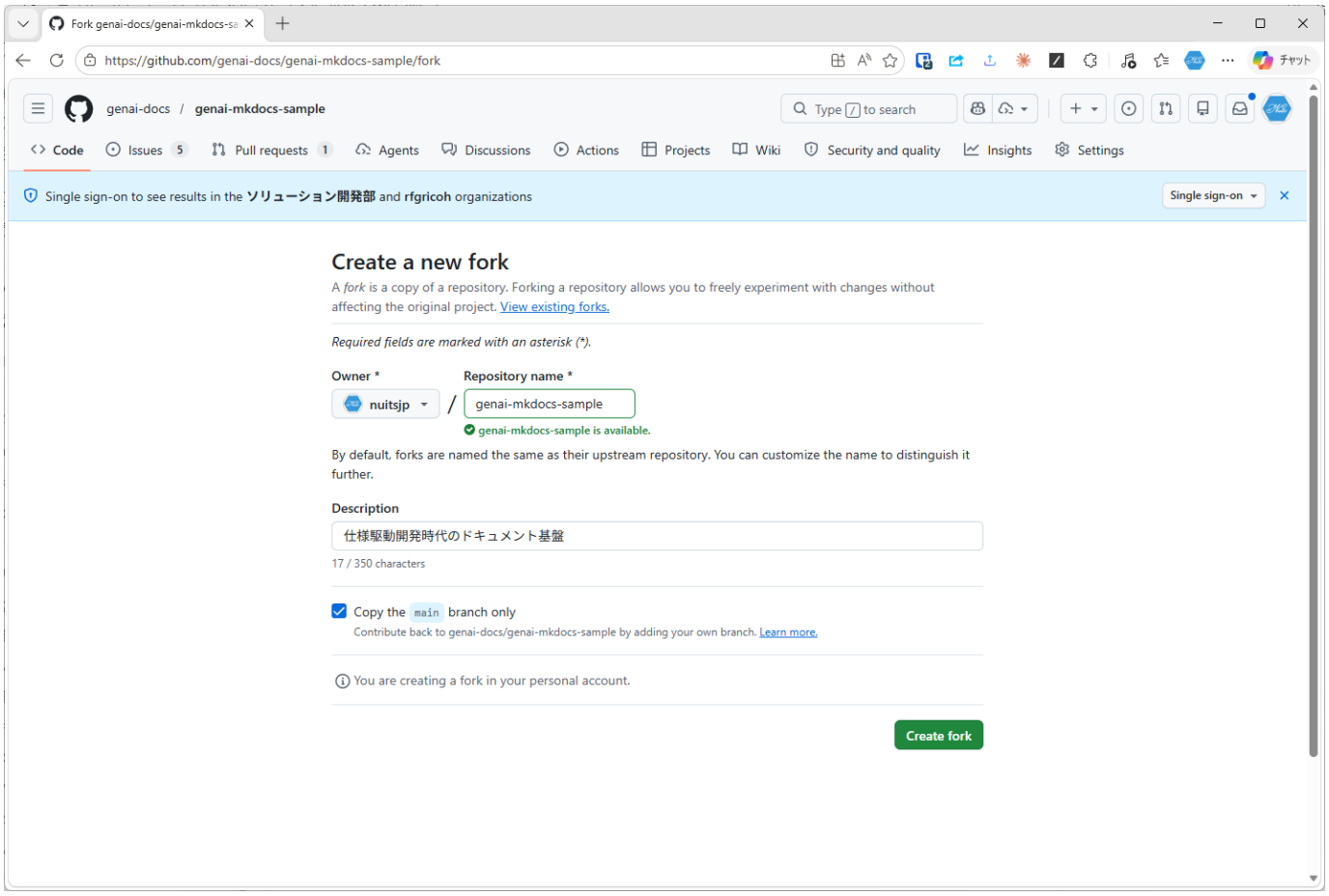
5.3.1 1. リポジトリを開く

対象リポジトリのトップページを開く。



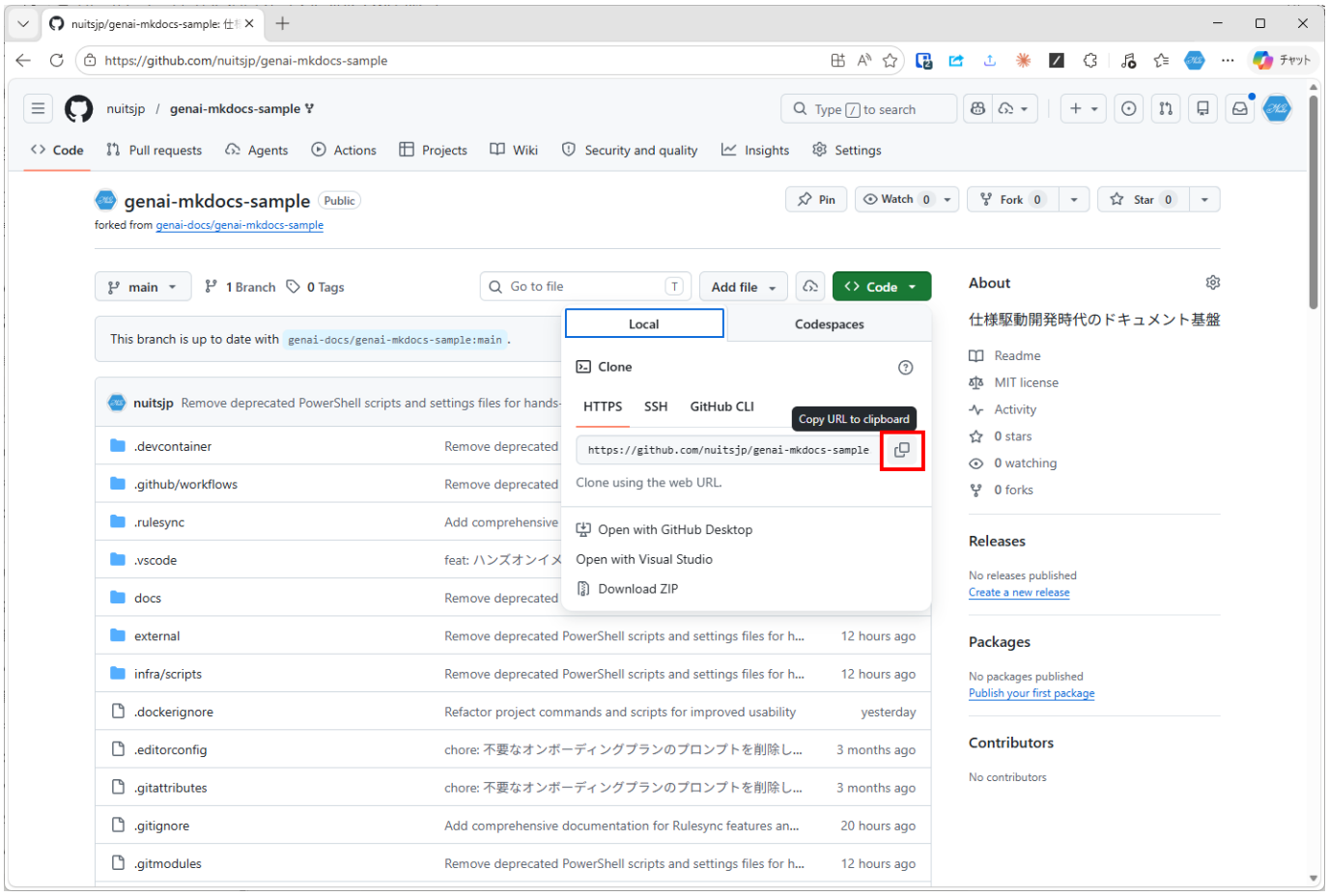
5.3.2 2. Fork を作成する

`Fork` ボタンから **自分のアカウント** をOwnerに指定して `Create fork` をクリックする。



5.3.3 3. クローン URL をコピーする

Forkした **自分のリポジトリ** を開き、`Code` → `HTTPS` タブからURLをコピーする。以降の `git clone` にはこのURLを使う。



5.3.4 4. VS Code でターミナルを開いてクローンする

VS Code上で `Ctrl + @` を押してターミナルを開き、コピーした **自分の Fork リポジトリの URL** を指定して `git clone` を実行する。

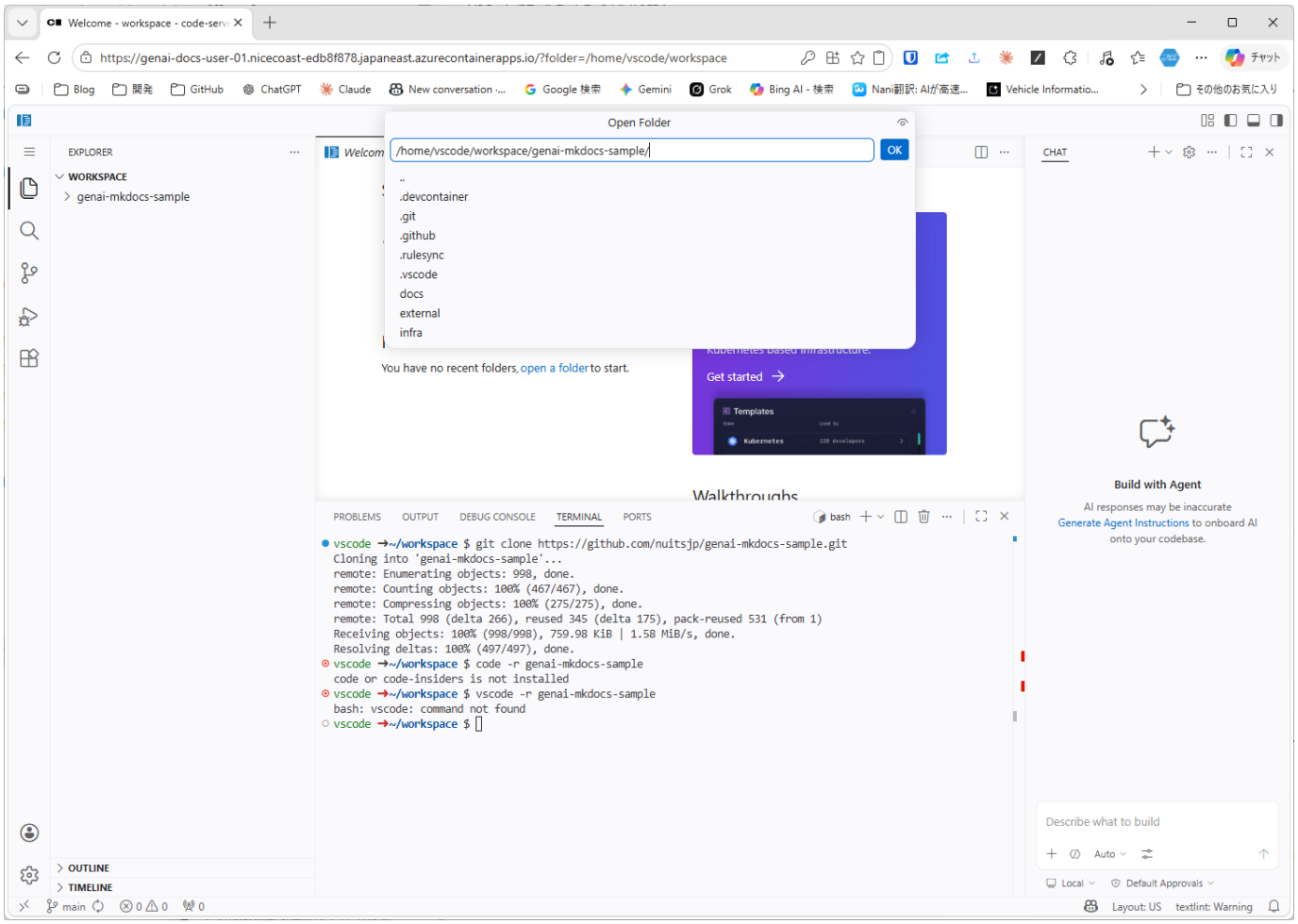
```
git clone https://github.com/<your-account>/genai-mkdocs-sample.git
```

`<your-account>` は自分のGitHubアカウント名に読み替える。

5.3.5 5. クローンしたフォルダーを開き直す

`File` → `Open Folder...` から下記のパスを入力し、クローンしたリポジトリを開き直す。

```
/home/vscode/workspace/genai-mkdocs-sample/
```

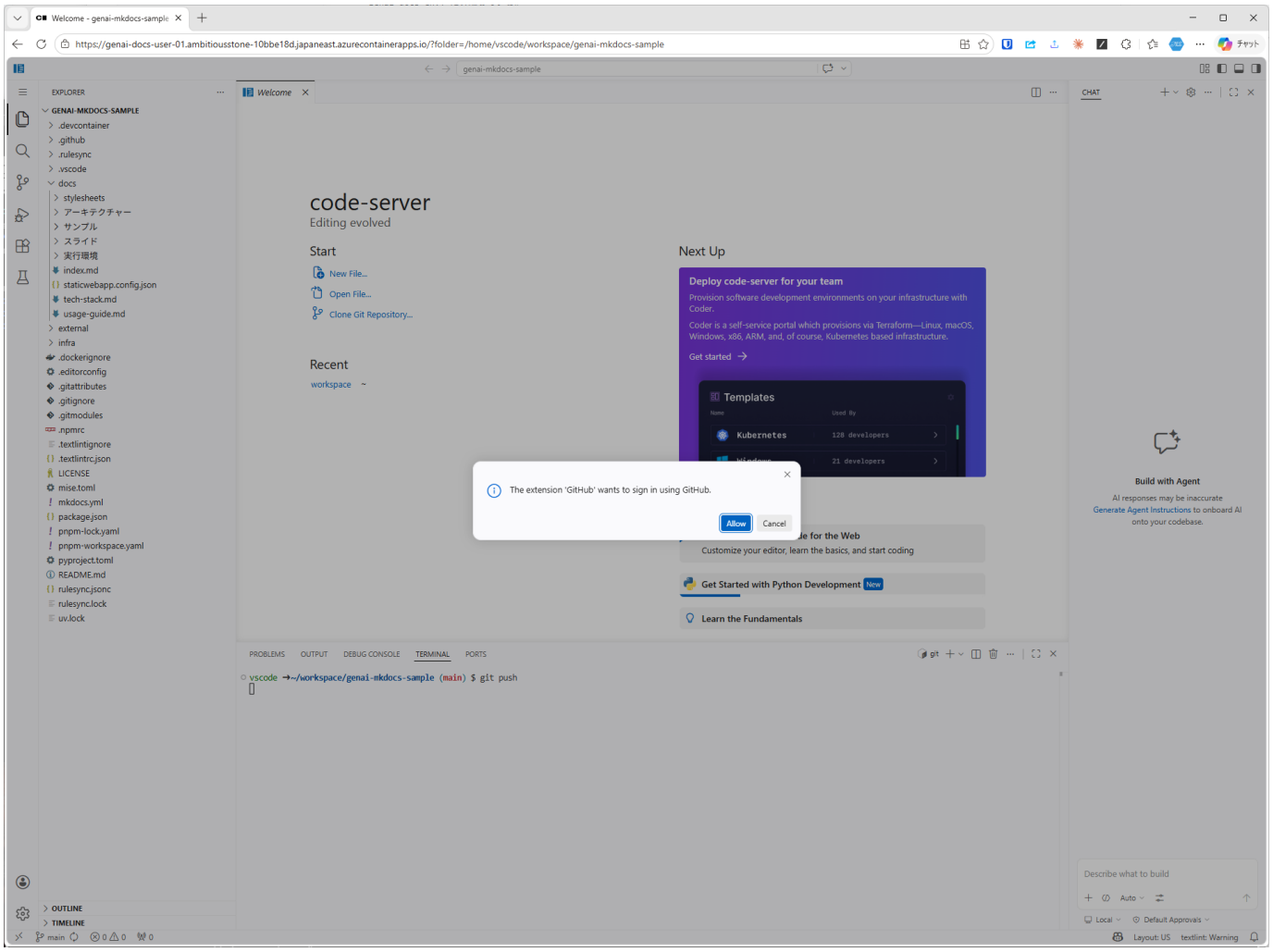


5.3.6 6. GitHub アカウントでサインインする

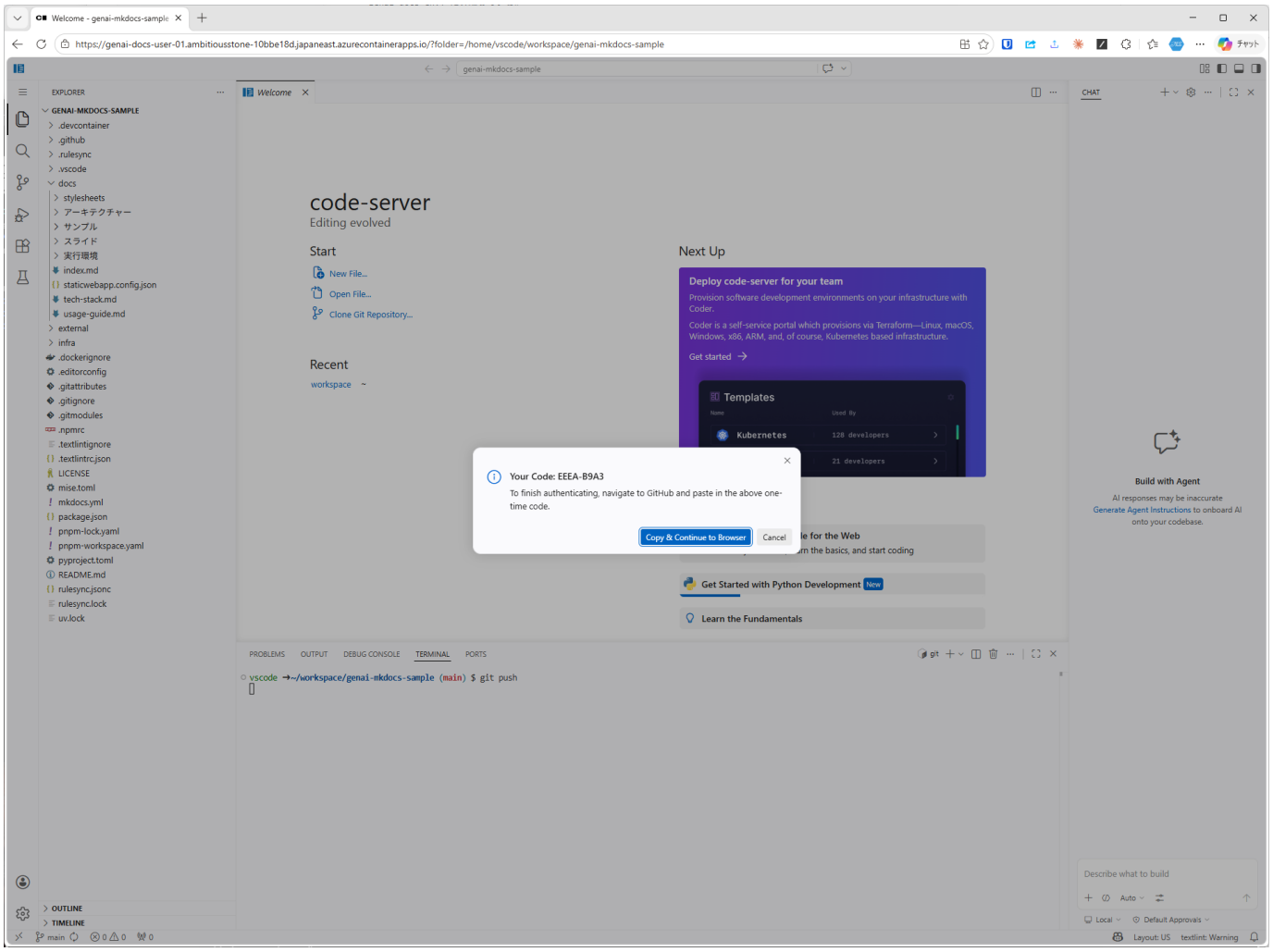
VS Code上で `Ctrl + @` を押してターミナルを開き、git push を実行する。初回はGitHubアカウントでのサインインが求められるので、画面の指示に従ってサインインする。

```
git push
```

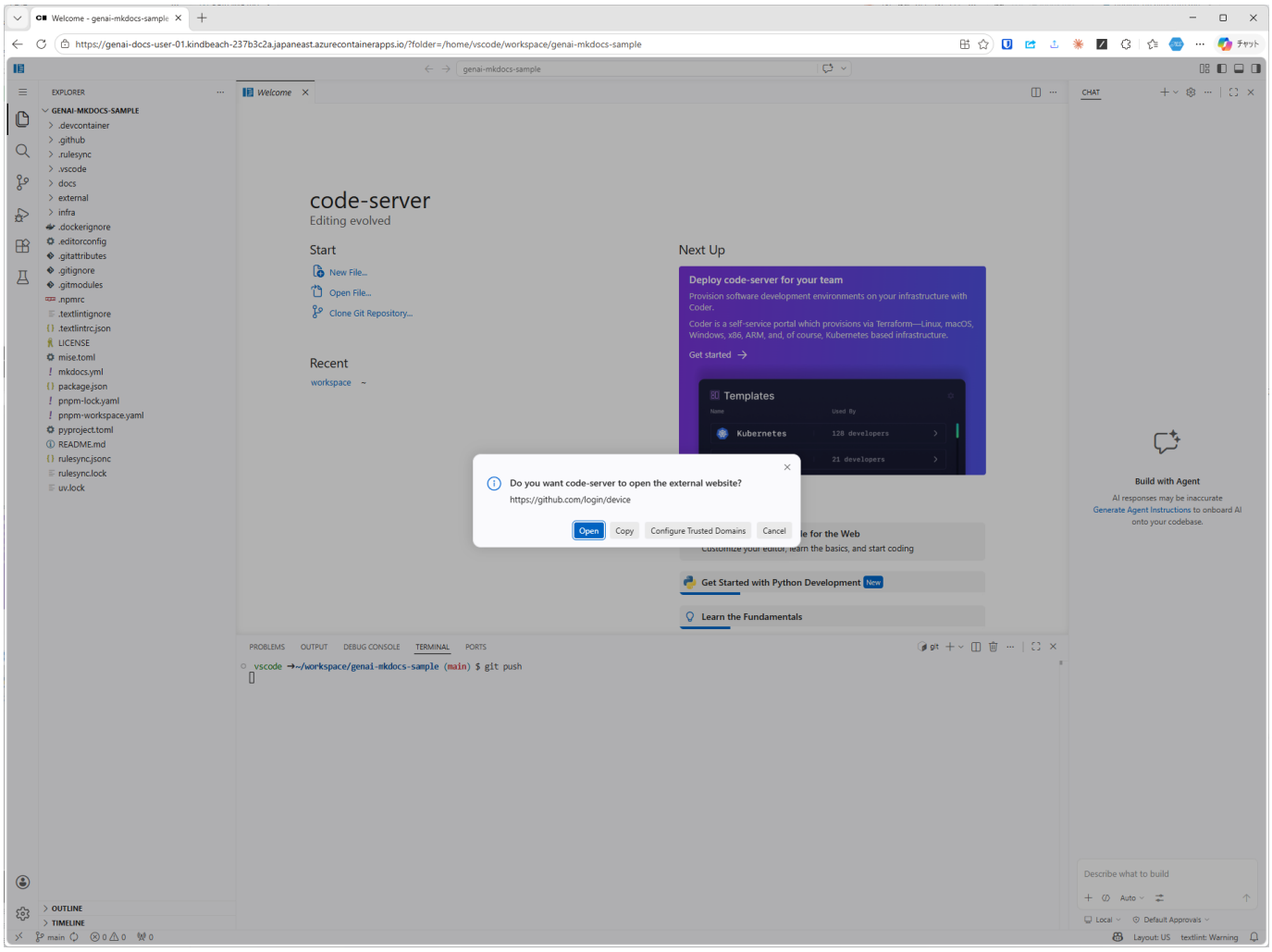
「Allow」を選択してサインインを許可する。



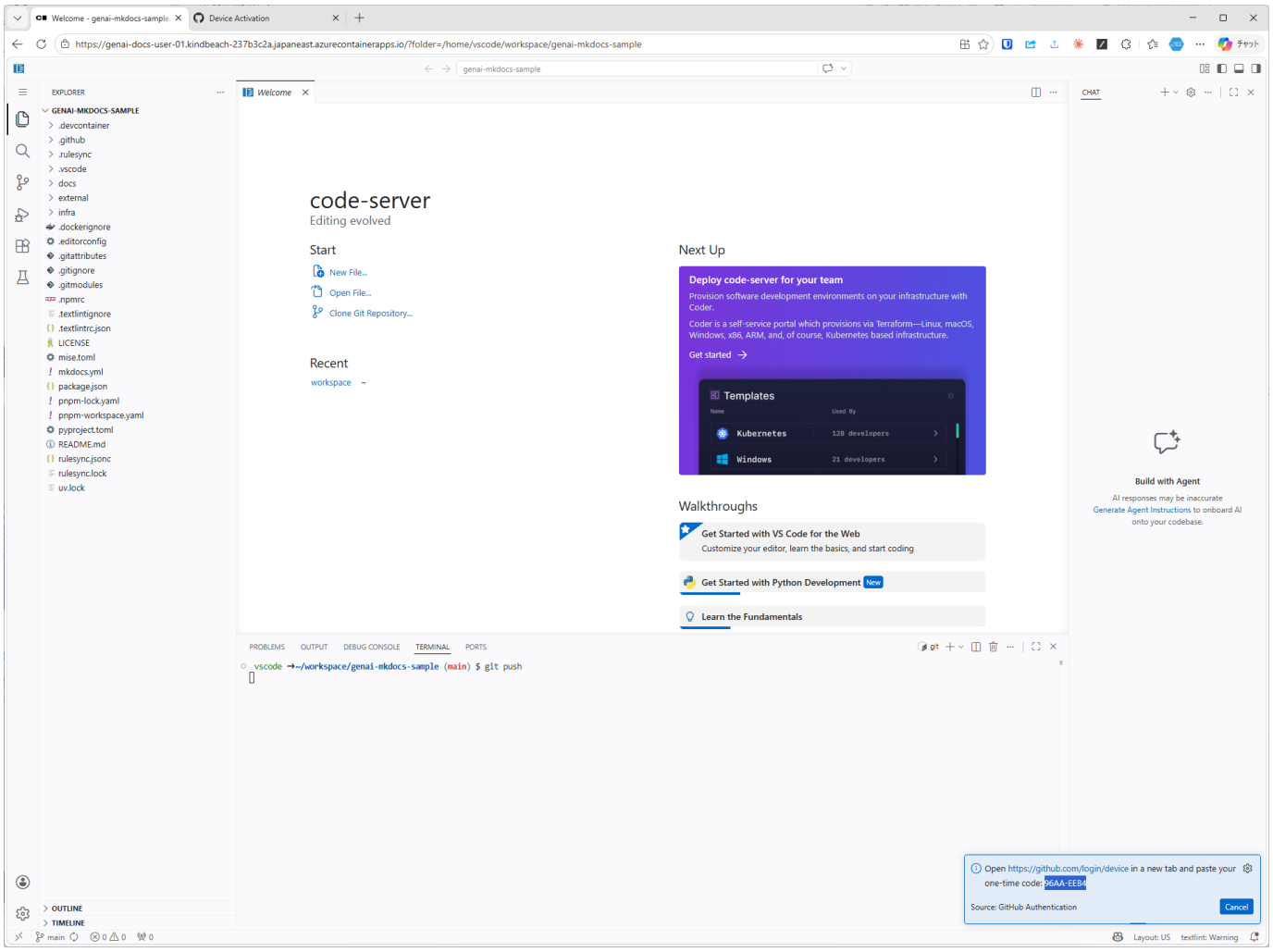
「Copy and Continue to Browser」を選択。



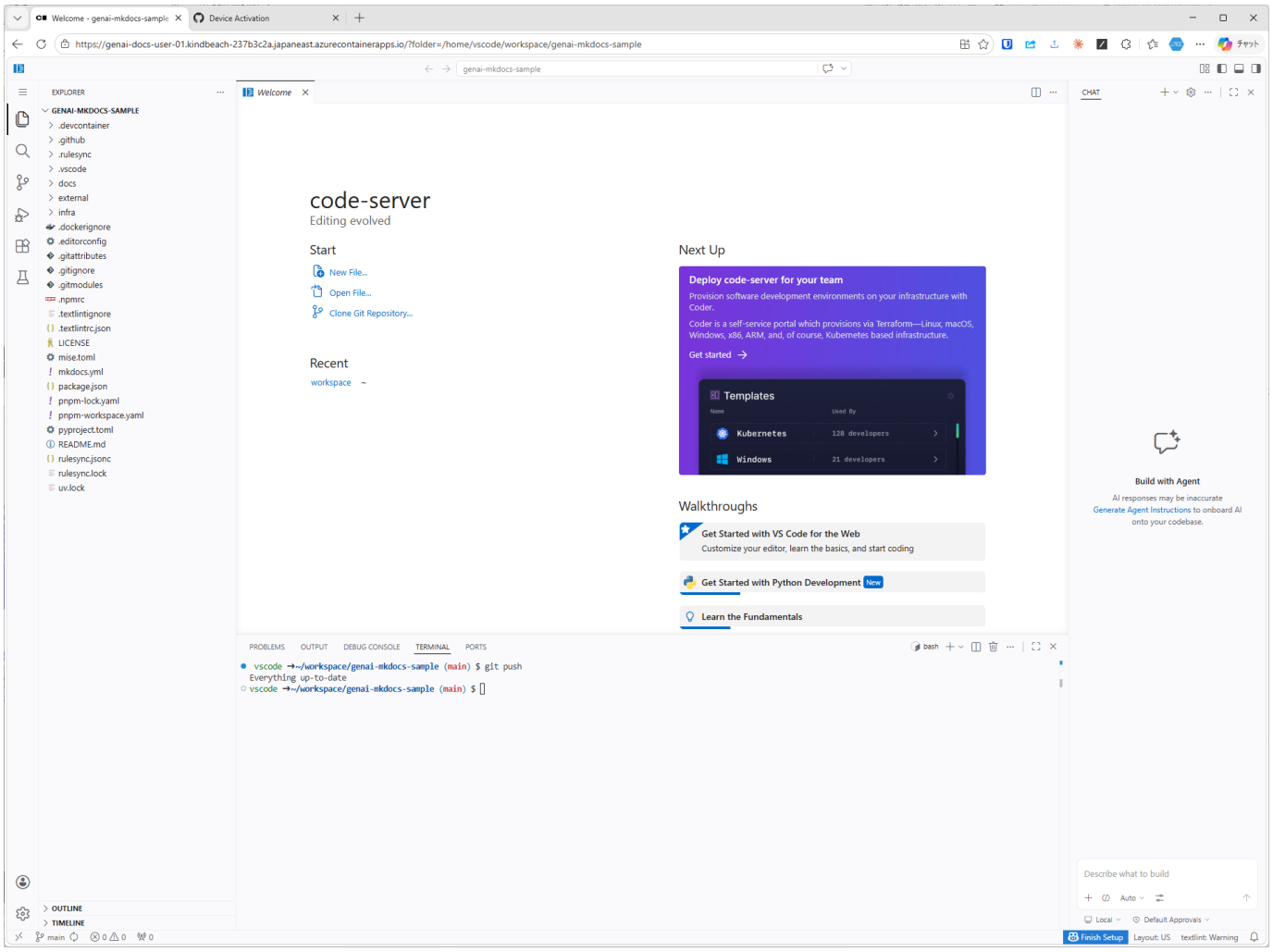
「Open」を選択してブラウザでGitHubの認証ページを開く。



Device Codeのコピーが求められますが、コピーが上手くいかない場合、VS Codeの画面にDevice Codeが表示されているので、そちらをブラウザの入力欄に直接入力しても認証する。



gitの空pushが成功していればOK。



5.3.7 6. Gitのユーザー名とメールアドレスを設定する

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

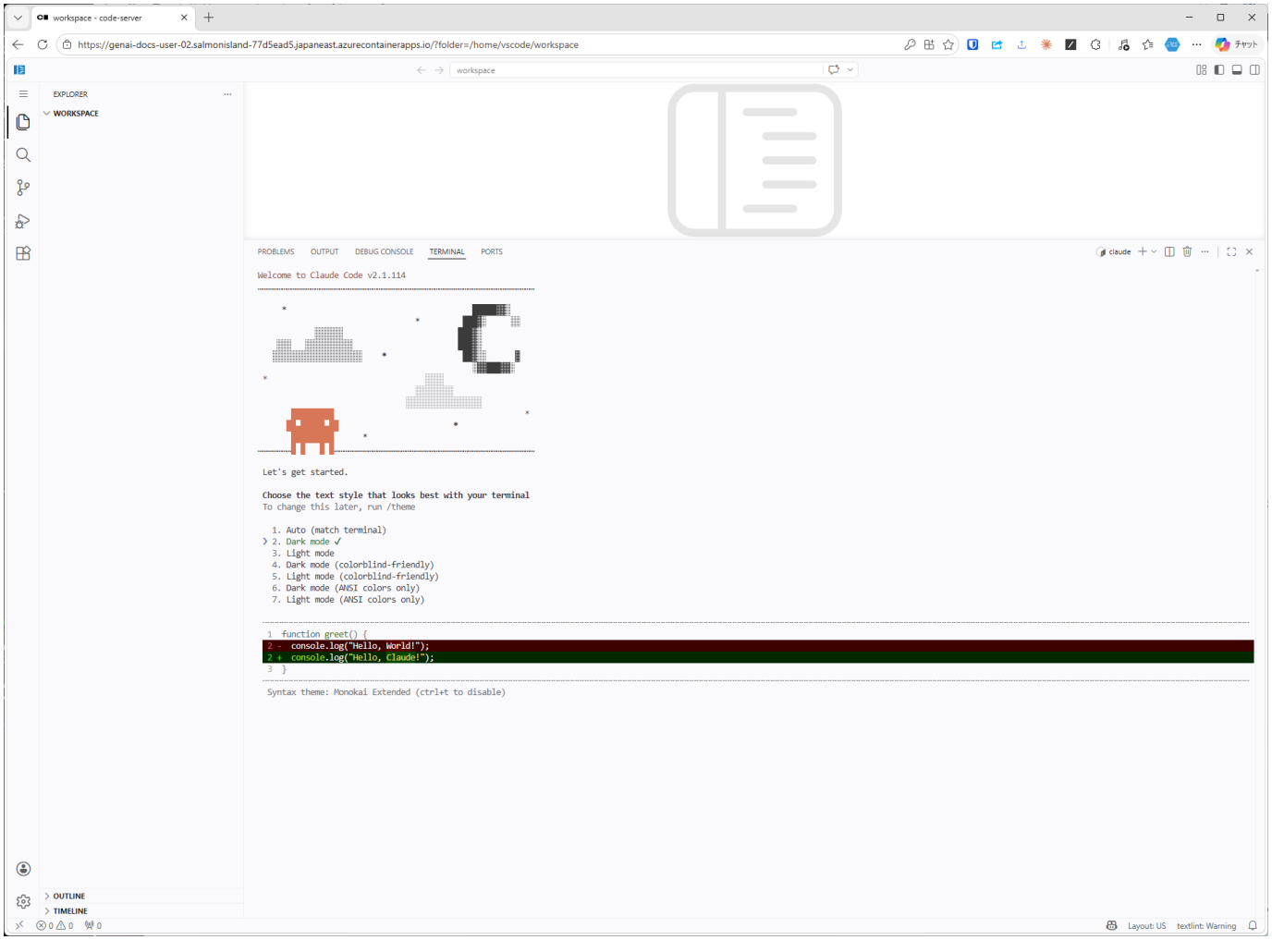
```
git config --global user.email "your_email@example.com"
```

5.4 Claude Code のサインイン

ターミナルで `claude` を起動し、ブラウザ経由で認証コードを取り込む。

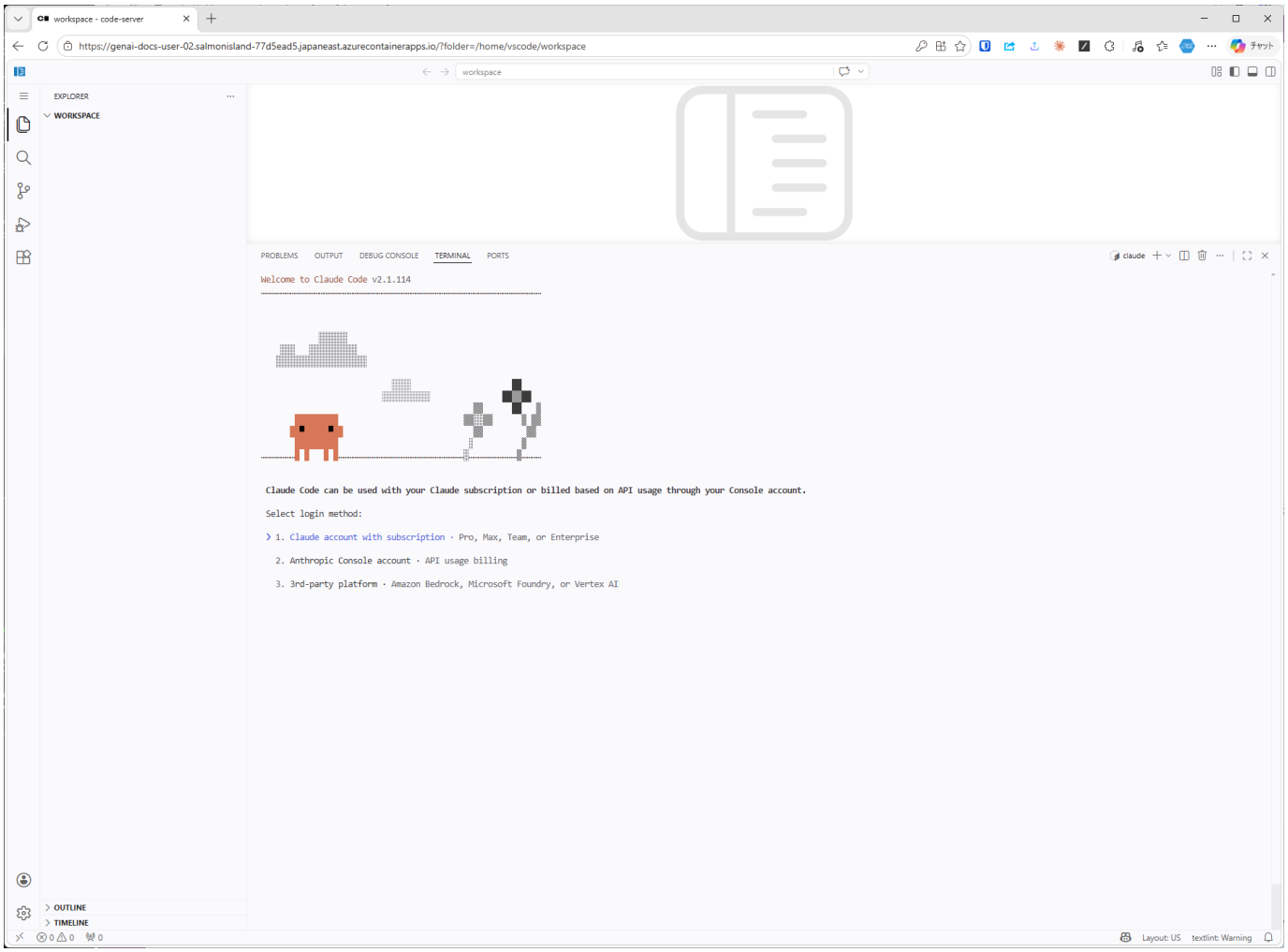
5.4.1 1. テーマを選択する

任意のテーマを選択する。



5.4.2 2. 認証方式を選択する

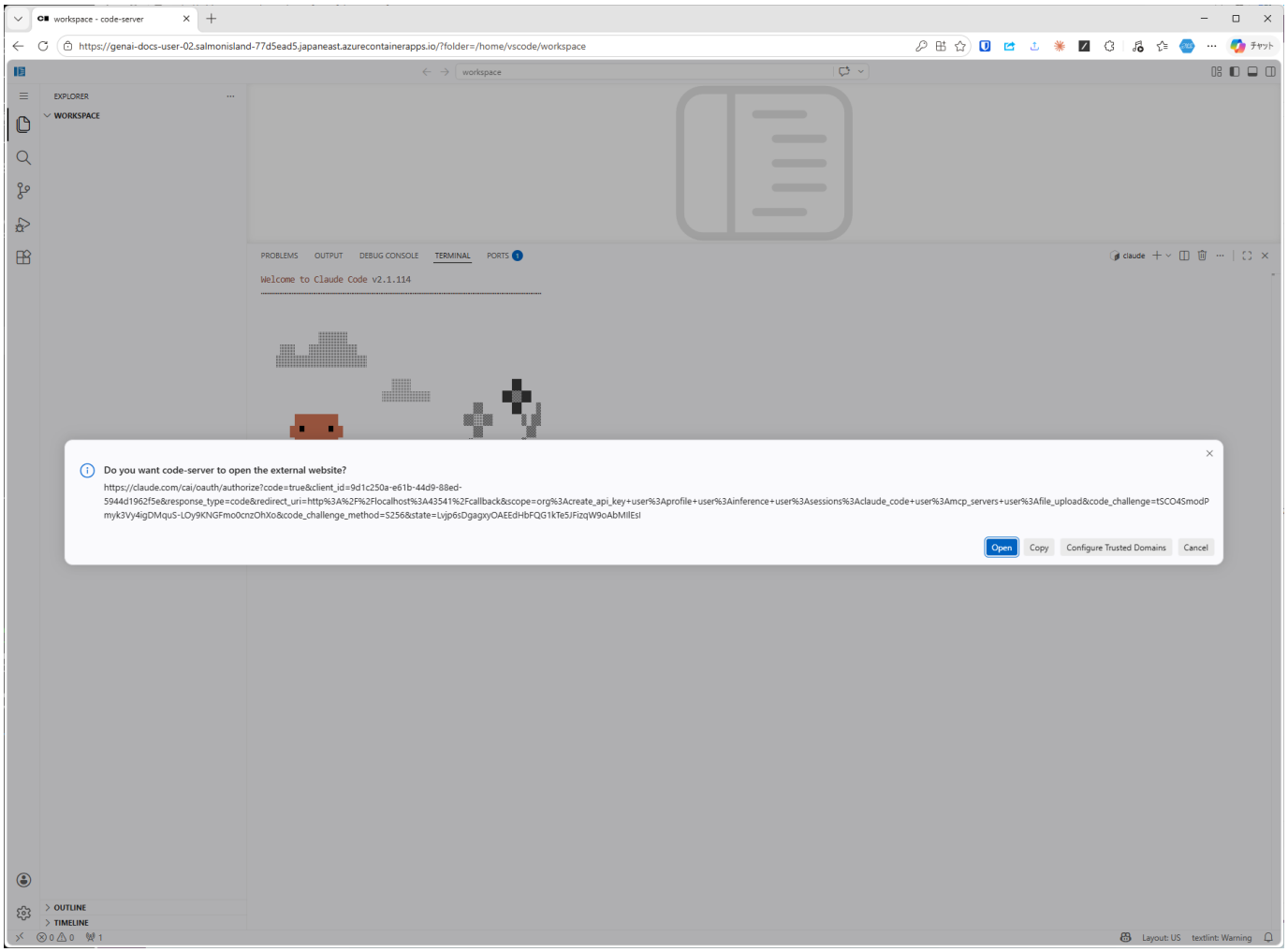
任意の認証方式を選択する。ここではサブスクリプション認証を選択する。



5.4.3 3. 「外部サイトを開く」ダイアログは閉じる

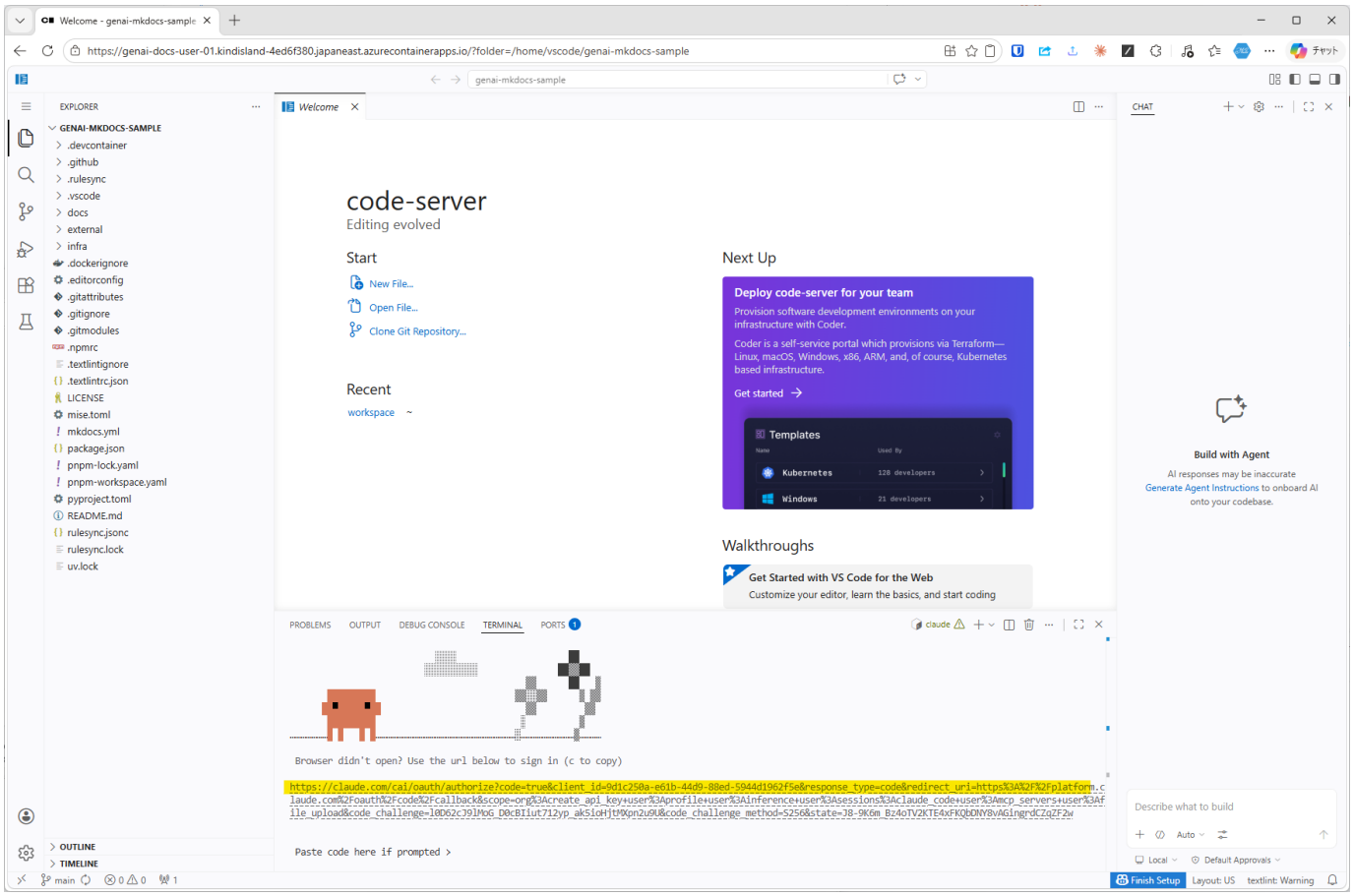
`claude.com` への遷移を促すダイアログが表示されるが、ここから `Open` を押すとリダイレクト先（ローカルホスト）の都合で正常に認証が完了しない。

このダイアログは閉じる。



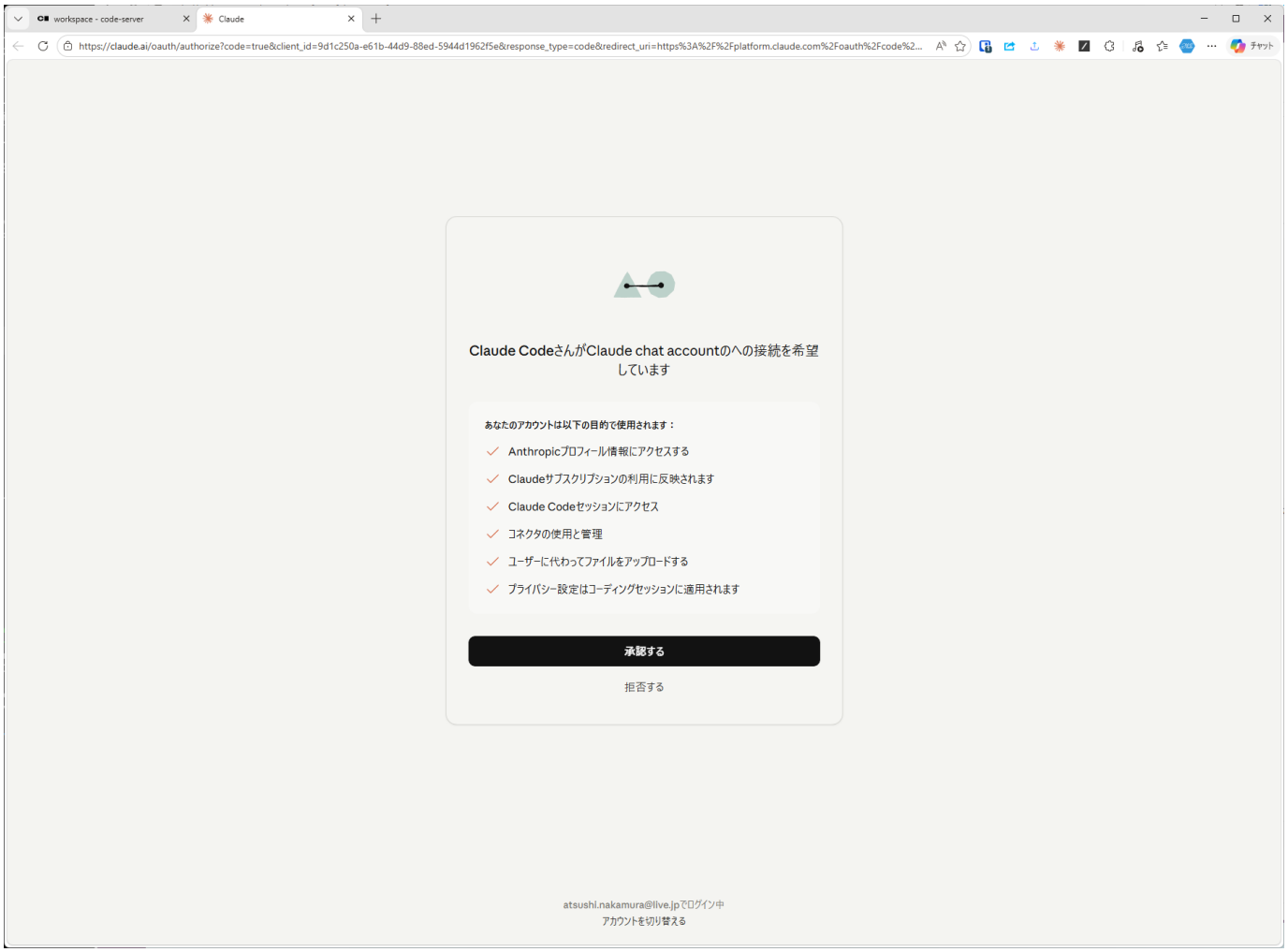
5.4.4 4. コンソール上の URL を `Ctrl + Click` で開く

コンソールにURLが表示される。そのURLを `Ctrl + Click` で開き、ブラウザでClaudeにログインする。



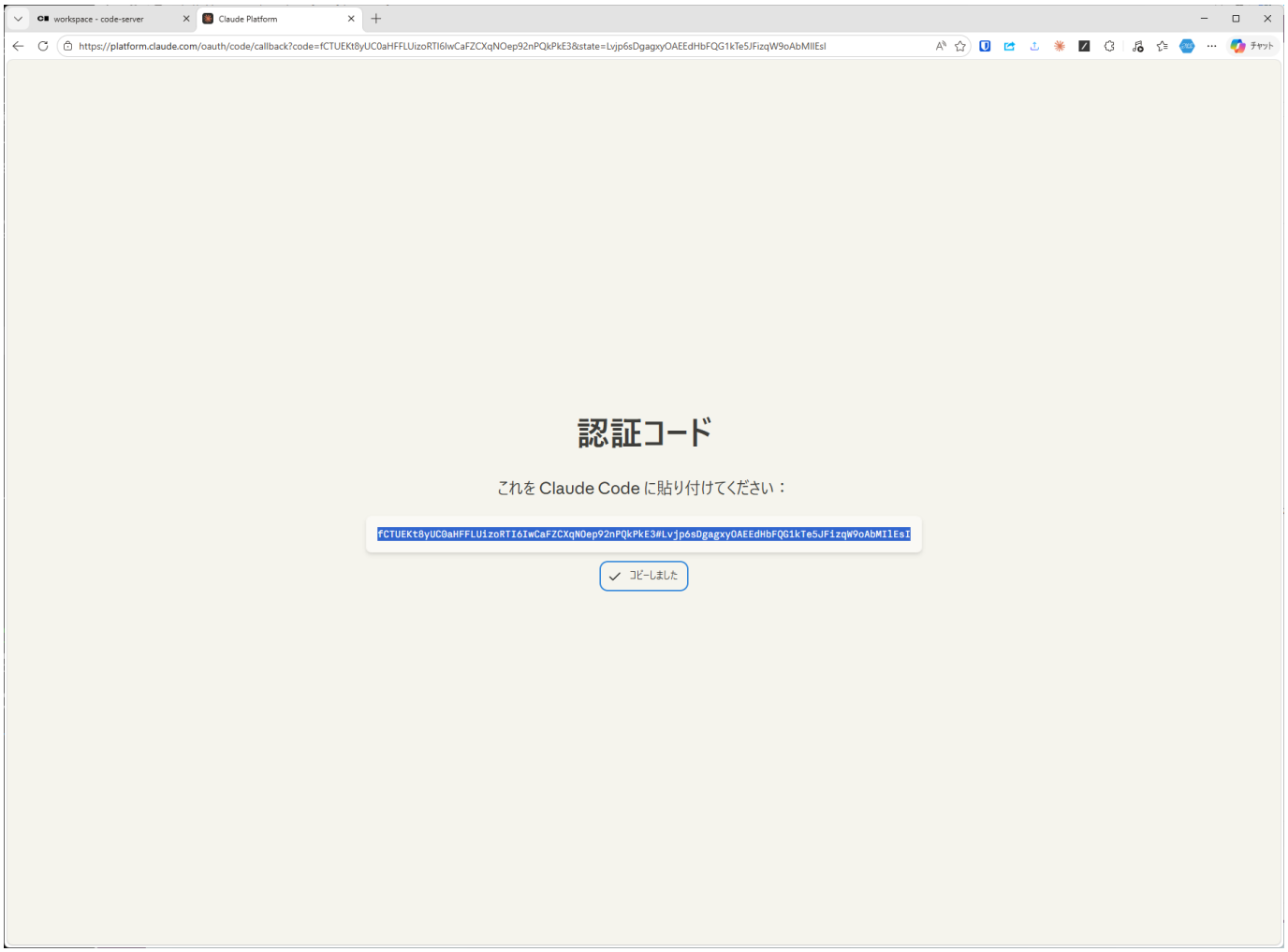
5.4.5 5. OAuth を承認する

ブラウザで `承認する` をクリック。



5.4.6 6. 認証コードをコピーする

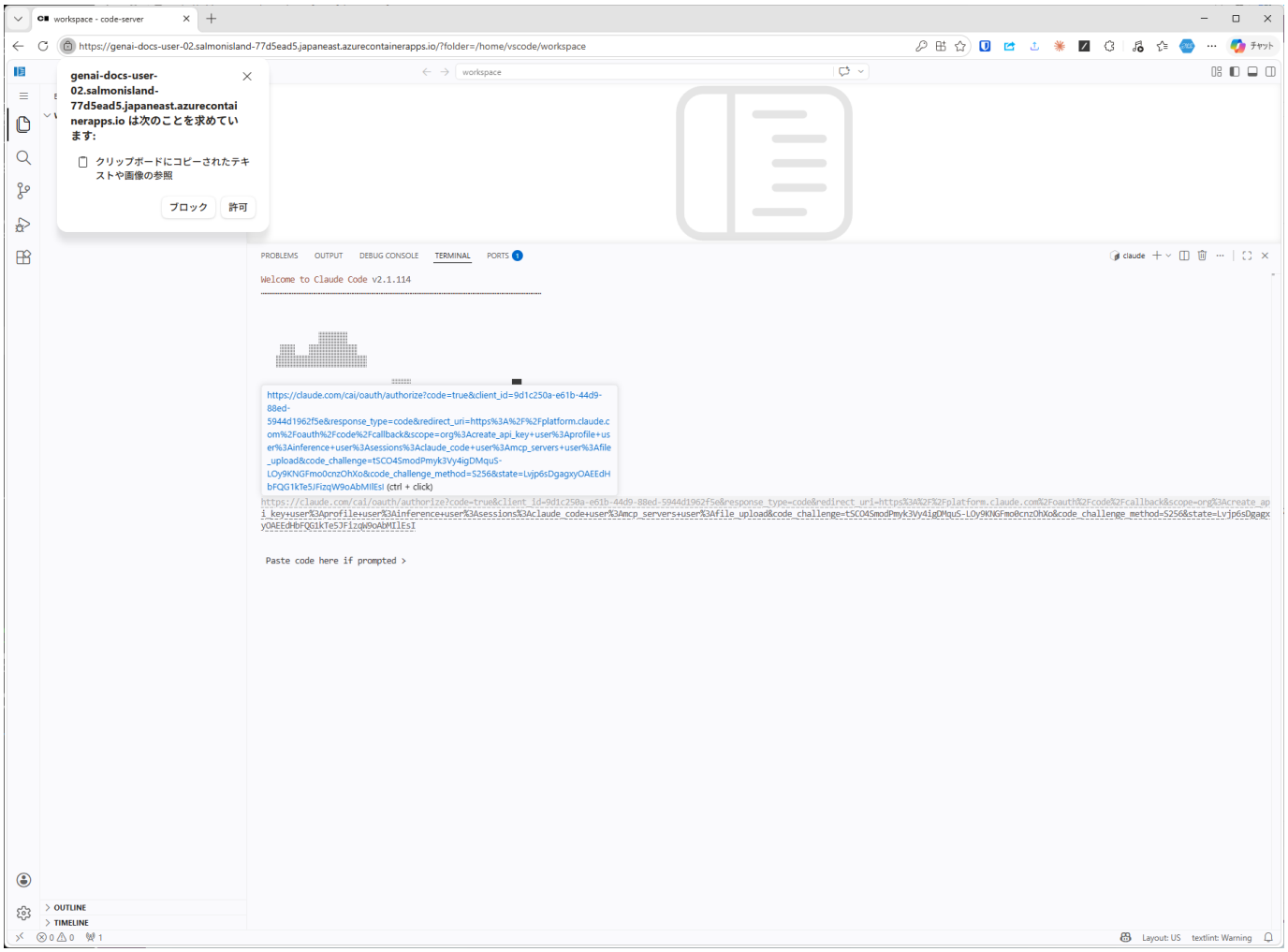
承認後に表示される認証コードを `コードをコピー` ボタンでクリックする。



5.4.7 7. コンソールにコードをペーストし、クリップボード参照を許可する

コンソールの `Paste code here if prompted >` に貼り付けると、ブラウザがクリップボード参照の許可を求めてくる。`許可` をクリックするとコードが貼り付けられる。

Enterを押して、サインインを完了する。

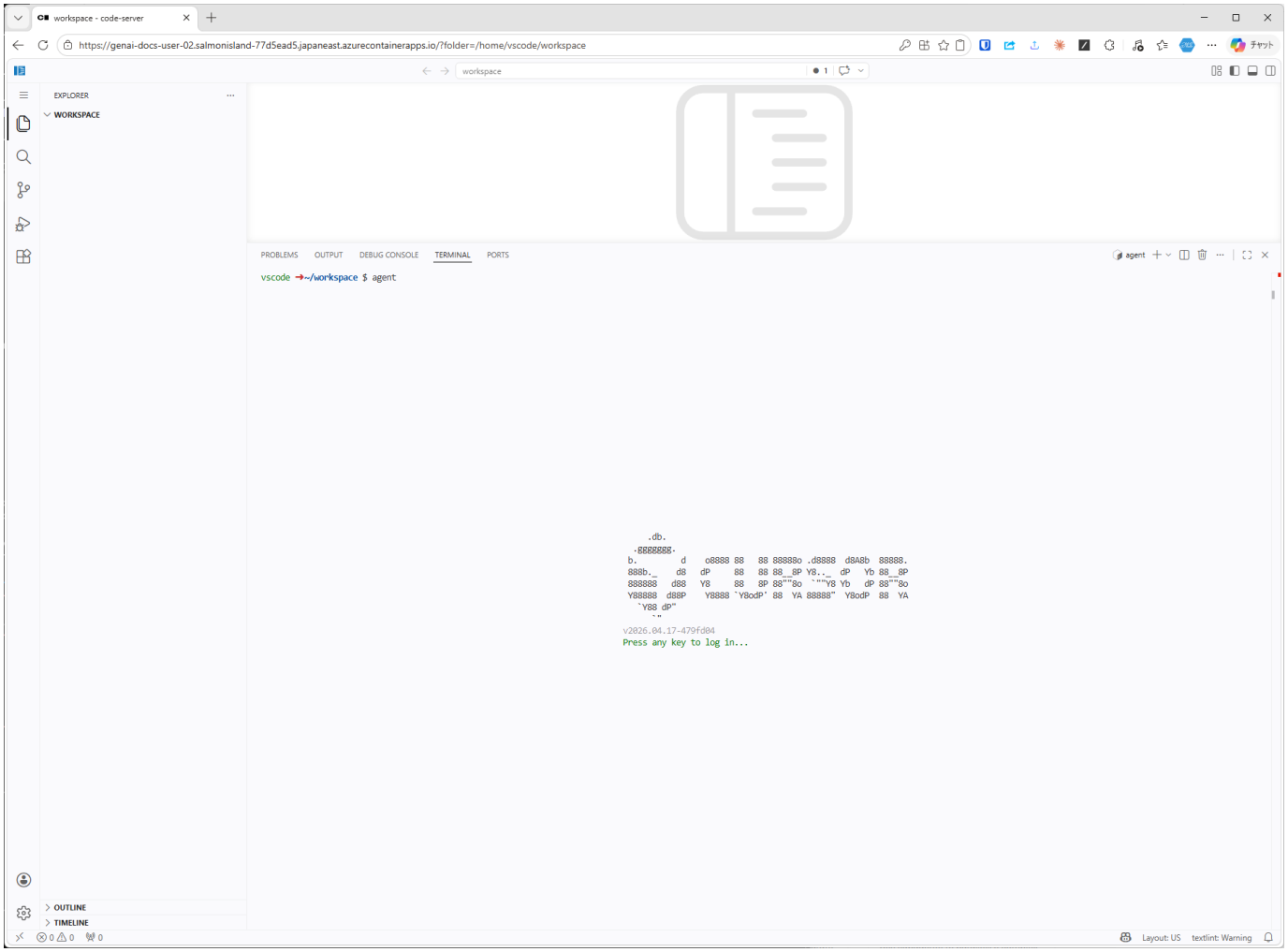


5.5 Cursor CLI のサインイン

VS Codeのターミナルで `agent` コマンドを実行し、ブラウザ経由でサインインする。

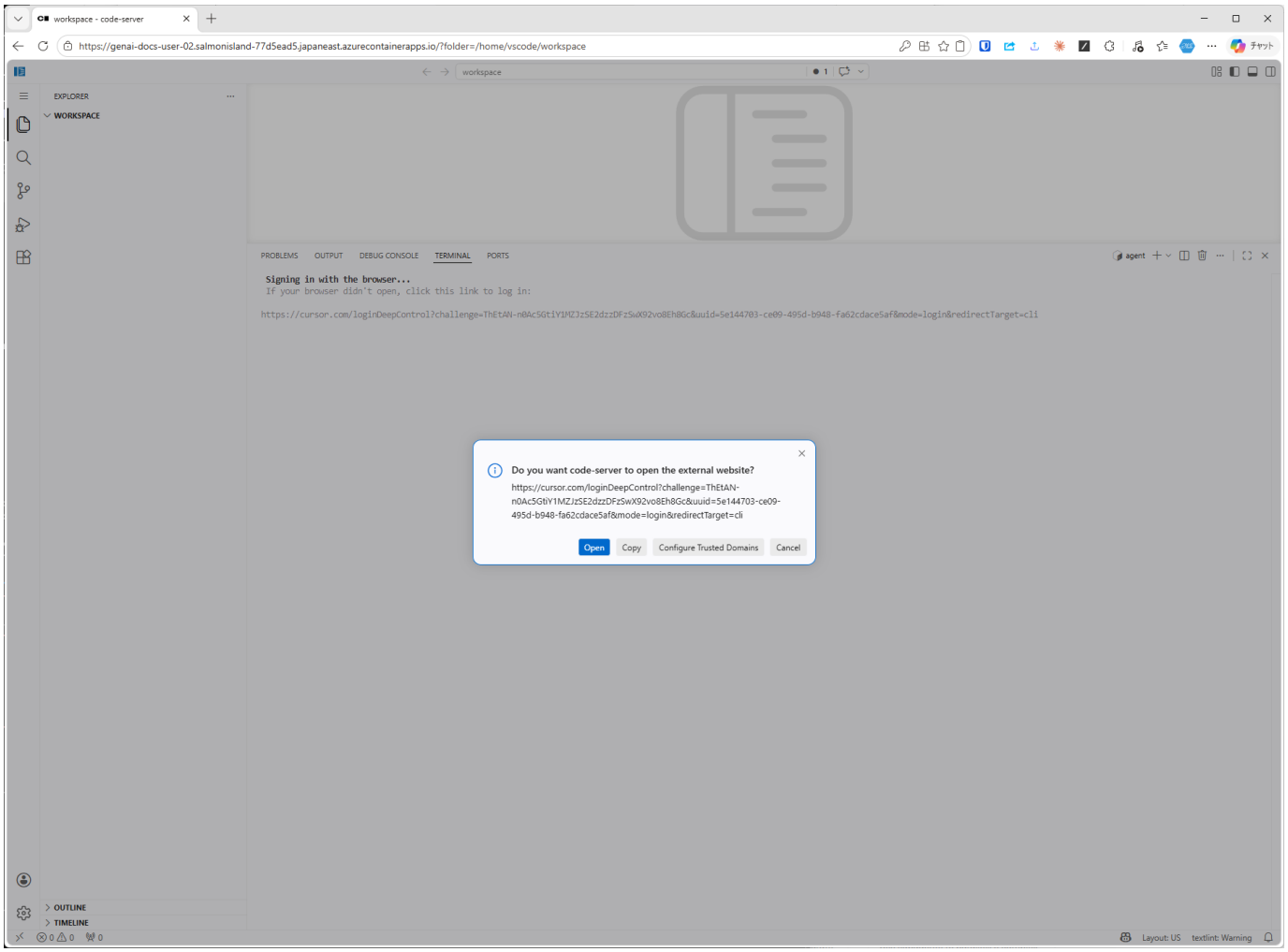
5.5.1 1. ターミナルを開いて `agent` を実行する

`Ctrl + @` でターミナルを開き、`agent` コマンドを実行する。起動ロゴと `Press any key to log in...` が表示される。



5.5.2 2. 任意のキーを押してブラウザ認証を承認する

任意のキーを押すと、`cursor.com` への遷移を促すダイアログが表示される。`Open` をクリックしてブラウザを開き、表示されたCursorのサインインを承認すると完了する。

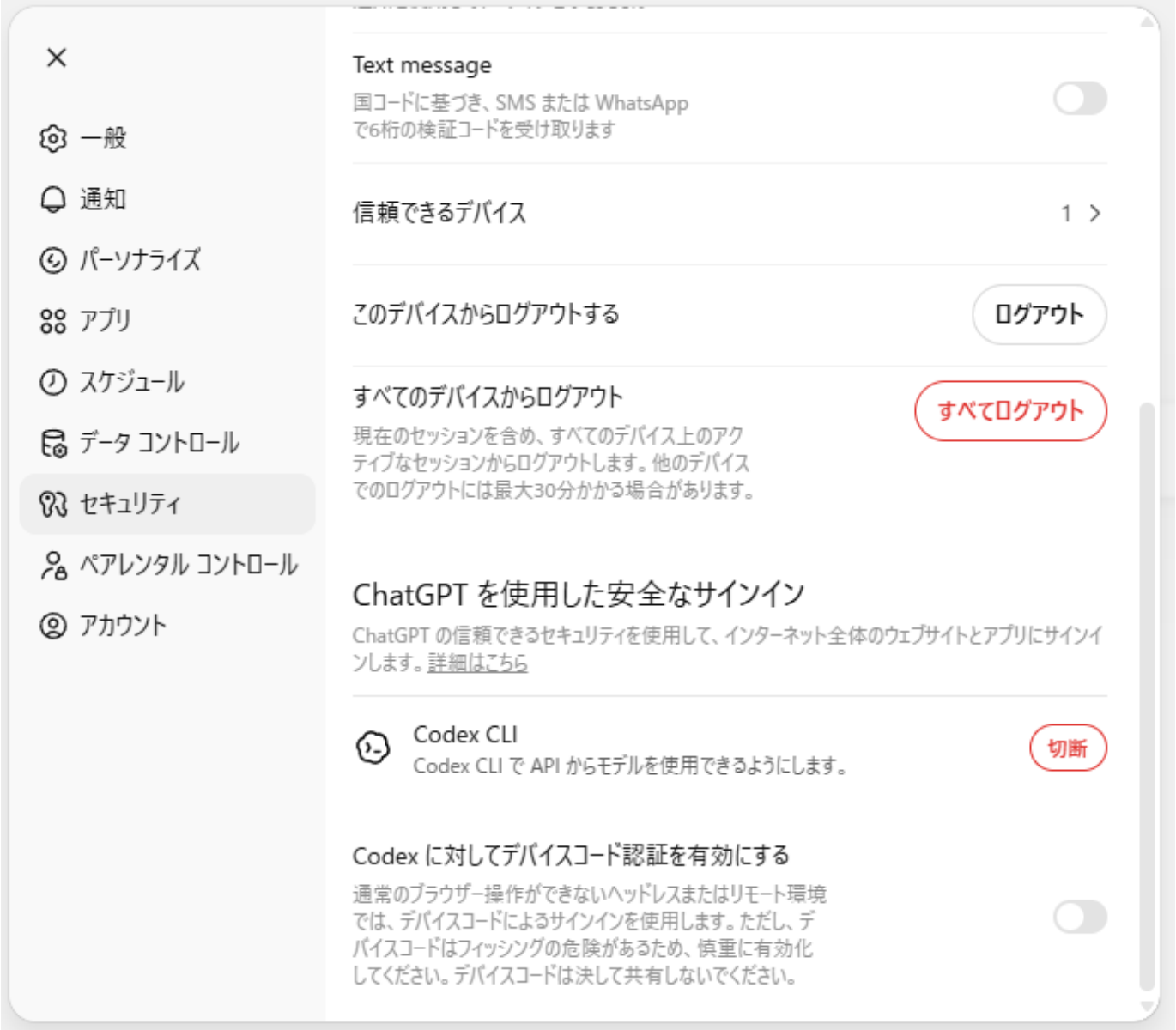


5.6 Codex CLI のサインイン

> ⚠ **デバイスコード認証を使う理由**
>
> Codex CLIは本環境ではURL認証が正常に起動しないため、**デバイスコード認証** を使う。 ただしOpenAIではフィッシング対策のため、デバイスコード認証は既定で **OFF** にされている。 本手順のために **一時的に ON** にし、認証が完了したら **必ず OFF に戻す**。

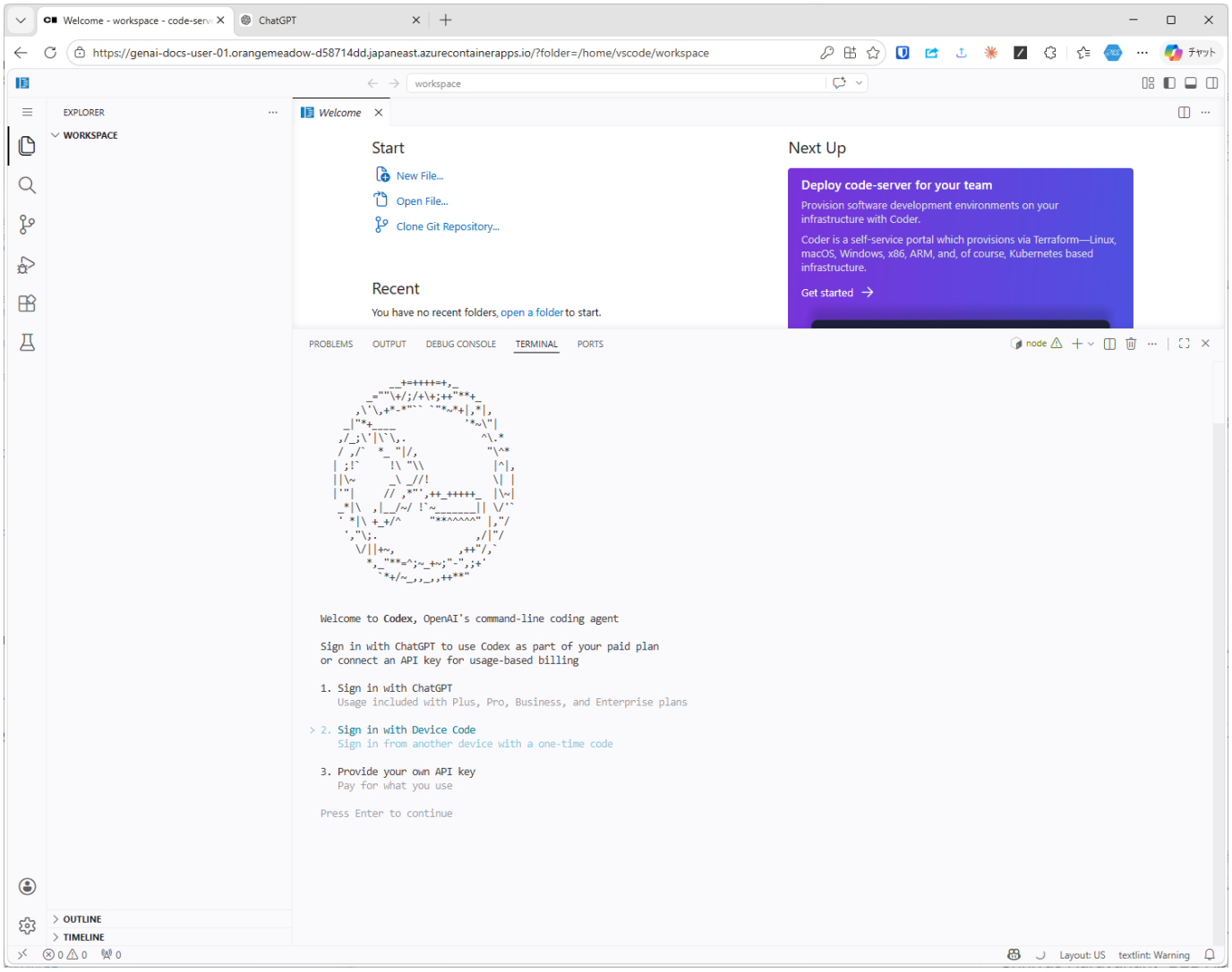
5.6.1 1. ChatGPT 側でデバイスコード認証を有効化する

ChatGPTの `設定` → `セキュリティ` → `ChatGPT を使用した安全なサインイン` で、`Codex に対してデバイスコード認証を有効にする` を **ON** にする。



5.6.2 2. Codex CLI を起動してサインイン方式を選ぶ

ターミナルで `codex` を起動し、`2. Sign in with Device Code` を選ぶ。



表示されるデバイスコードをChatGPT側の案内ページに入力してサインインを完了する。

5.6.3 3. デバイスコード認証を OFF に戻す

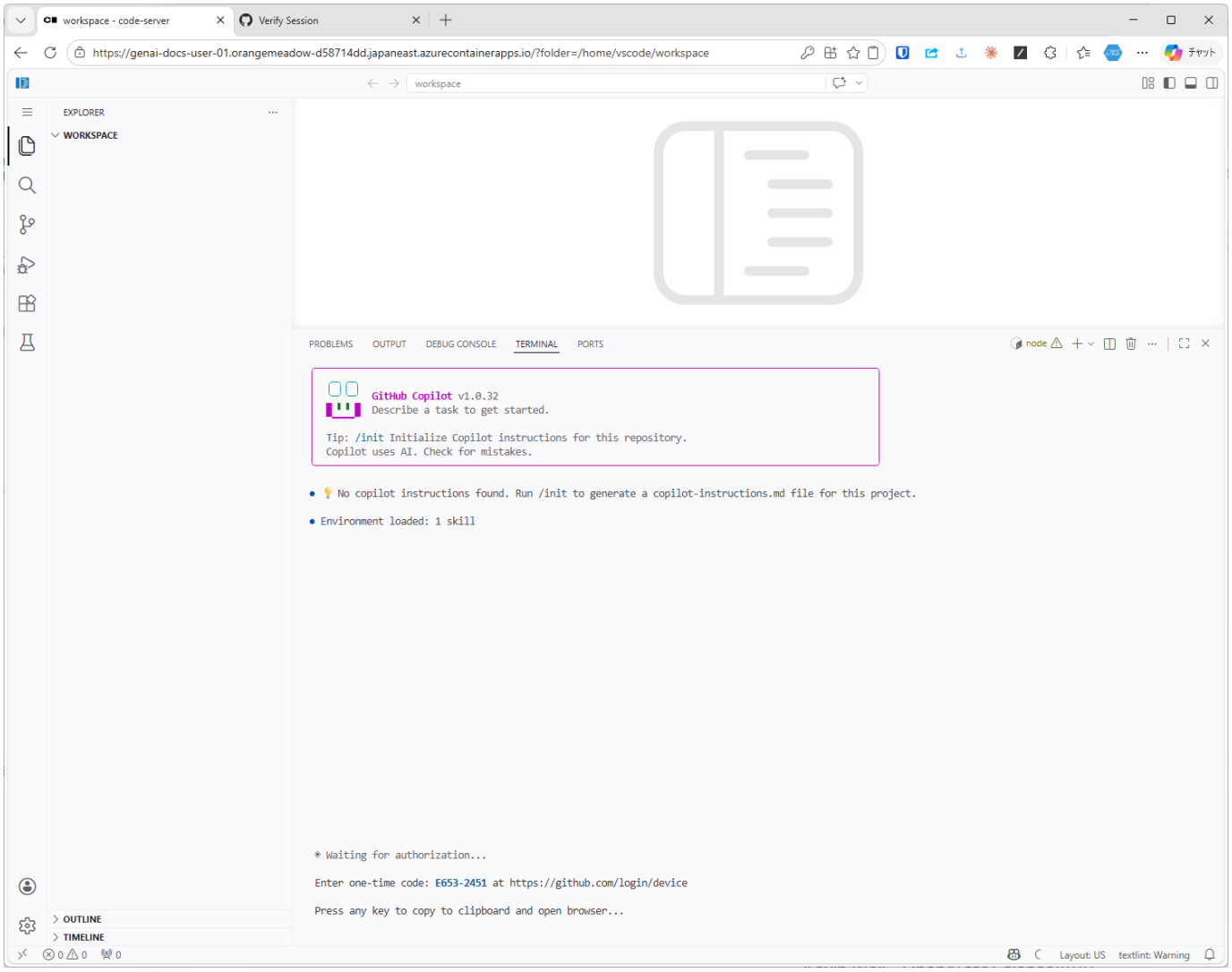
サインインが完了したら、手順1と同じ画面で `Codex に対してデバイスコード認証を有効にする` を **必ず OFF** に戻す。ONのまま放置するとフィッシング被害のリスクが高まるため、この作業を忘れないこと。

5.7 GitHub Copilot CLI のサインイン

ターミナルで `copilot` を起動し、ワンタイムコード方式でGitHubアカウントに接続する。

5.7.1 1. Copilot CLI を起動

起動すると `Waiting for authorization...` とともに **one-time code**（例： `E653-2451`）と `https://github.com/login/device` のURLが表示される。



5.7.2 2. ブラウザで one-time code を入力

`https://github.com/login/device` を開き、ターミナルに表示されたone-time codeを入力して承認する。ターミナルがサインイン完了状態に切り替わる。

6. アーキテクチャー

6.1 アーキテクチャ

このセクションでは、ドキュメント基盤の技術的な仕組みと設定について説明する。

6.1.1 ドキュメント

- ワークフロー アーキテクチャ - GitHub Actionsワークフローの実行条件と内部構造
- テキスト校正 - textlintによる日本語文書の品質管理

6.1.2 デプロイ構成

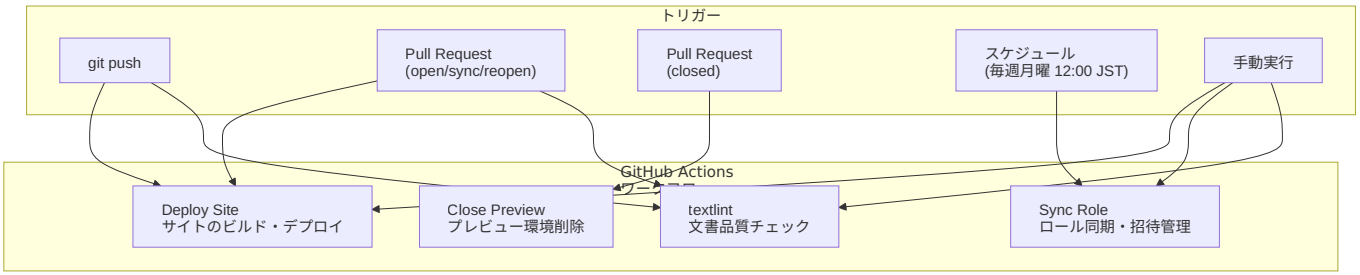
サイトのデプロイには `nuitsjp/azure-blob-storage-site-deploy` を利用している。プレフィックス方式によるマルチ環境配置、OIDC認証、PRステージングの自動作成・削除などの設計意図は、該当ActionのREADME（日本語版）を参照する。

6.2 ワークフロー アーキテクチャ

本リポジトリで使用するGitHub Actionsワークフローの実行条件と内部構造について説明する。

6.2.1 ワークフロー概要

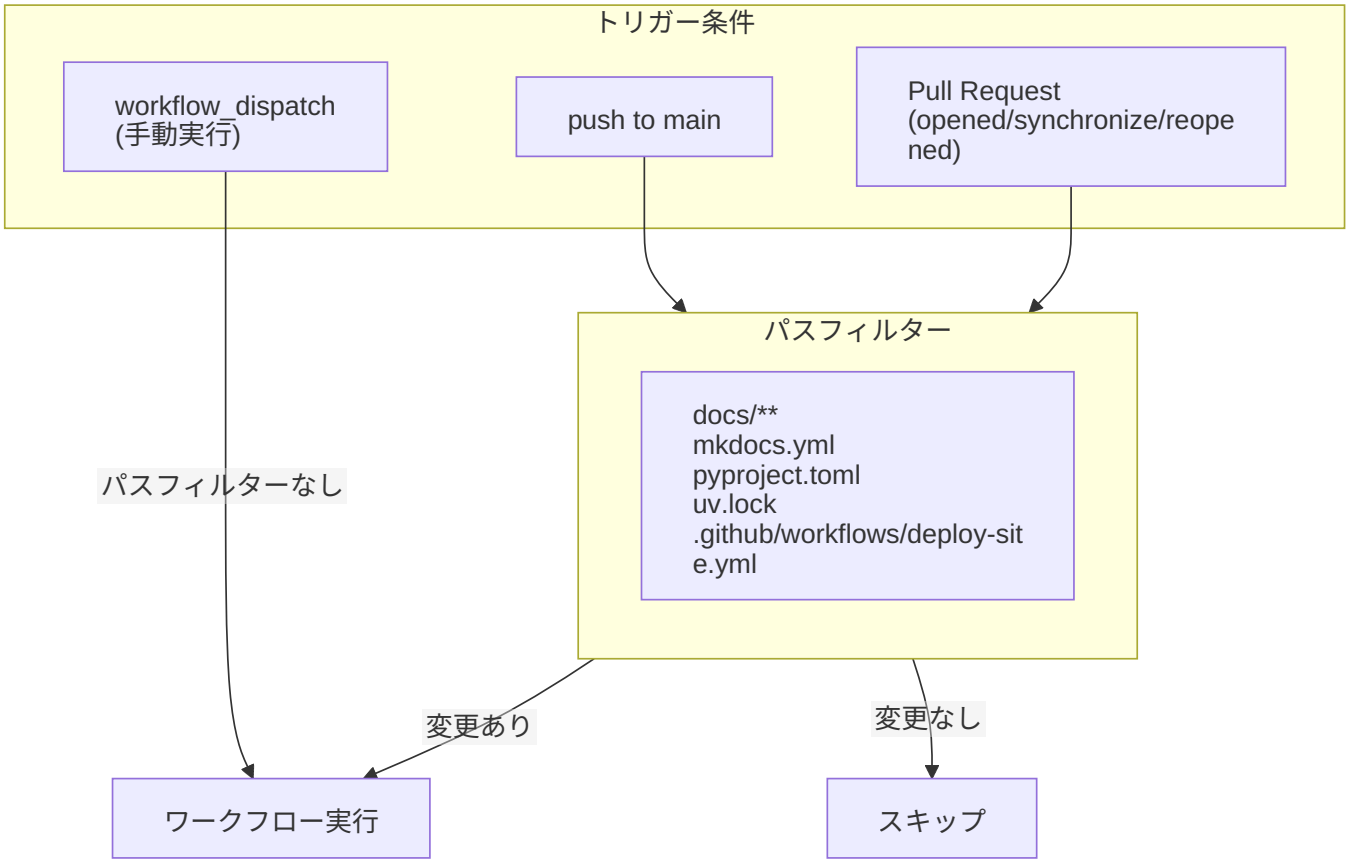
本システムでは以下の4つのワークフローを使用する。



ワークフロー	ファイル	主な役割
Deploy Site	`.github/workflows/deploy-site.yml`	MkDocs サイトのビルドと GitHub Pages / Azure SWA へのデプロイ
Close Preview	`.github/workflows/close-preview.yml`	PR クローズ時に Azure SWA のプレビュー環境を削除
Sync Role	`.github/workflows/role-sync-released.yml`	GitHub 権限に基づく Azure SWA ロールの同期と招待管理
textlint	`.github/workflows/textlint.yml`	Markdown 文書の日本語品質チェック

6.2.2 Deploy Site ワークフロー

実行条件



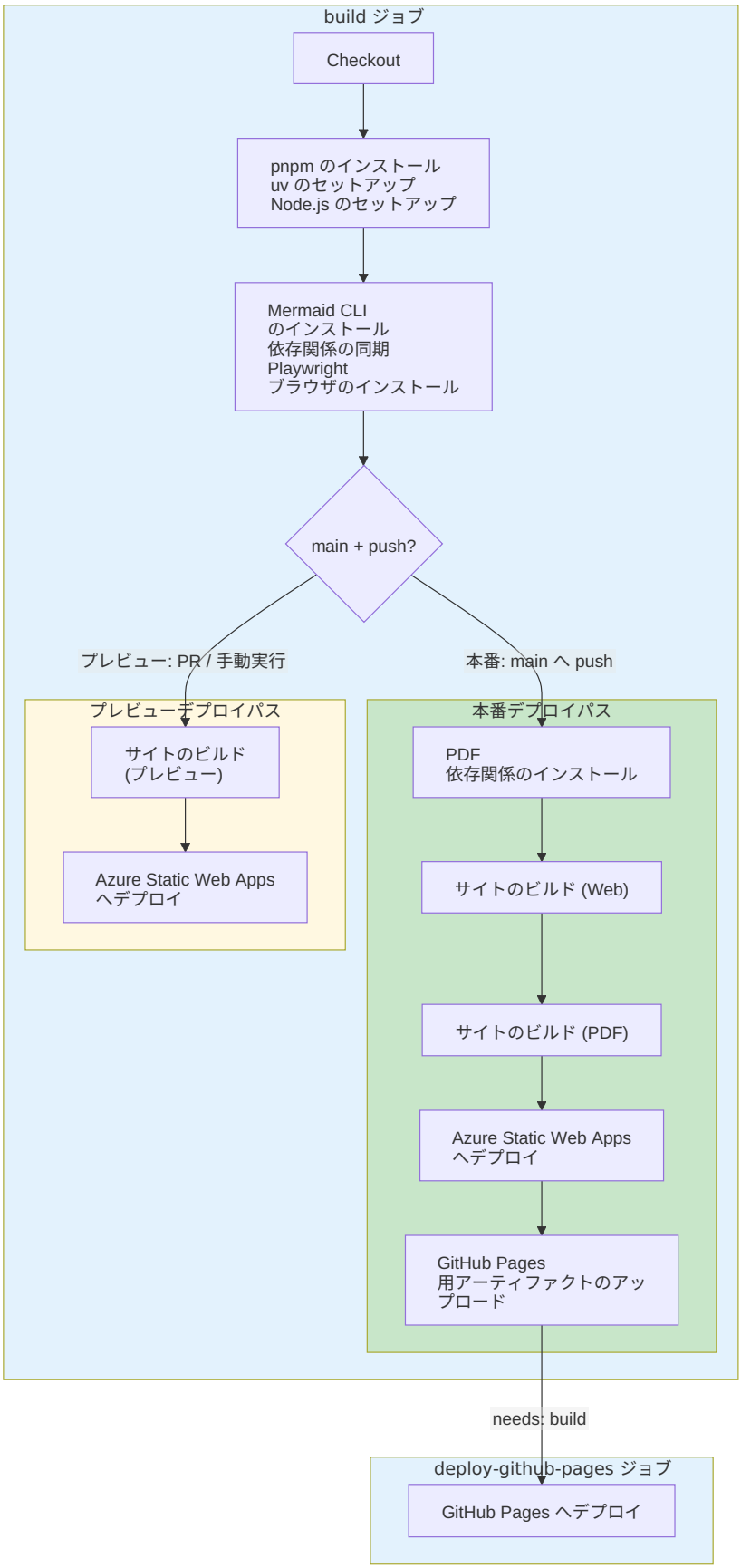
トリガー	条件	パスフィルター
`push`	`main` ブランチへのプッシュ	`docs/**`, `mkdocs.yml`, `pyproject.toml`, `uv.lock`, `.github/workflows/deploy-site.yml`
`pull_request`	opened, synchronize, reopened	同上
`workflow_dispatch`	手動実行	なし（常に実行）

権限

```yaml
permissions:
  contents: read       # リポジトリ読み取り
  pages: write         # GitHub Pages への書き込み
  id-token: write      # OIDC トークン取得
  pull-requests: write # PR コメント
```

ジョブ構成

ワークフローは `build` ジョブと `deploy-github-pages` ジョブの2つで構成される。



ジョブ詳細

BUILD ジョブ

ステップ	説明	条件
Checkout	サブモジュール含むチェックアウト	-
pnpm のインストール	pnpm パッケージマネージャー	-
uv のセットアップ	Python (uv) 環境構築	-
Node.js のセットアップ	Node.js 20 環境構築	-
Mermaid CLI のインストール	`@mermaid-js/mermaid-cli` グローバルインストール	-
依存関係の同期	`uv sync` による Python 依存関係	-
Playwright ブラウザのインストール	Chromium ブラウザ	-
PDF 依存関係のインストール	PDF 生成用システムライブラリ	main + push 時のみ
サイトのビルド (Web)	SVG 変換でビルド	main + push 時のみ
サイトのビルド (PDF)	PNG 変換 + PDF 生成	main + push 時のみ
サイトのビルド (プレビュー)	変換なしでビルド	main + push 以外
Azure Static Web Apps へデプロイ	サイトをデプロイ	-
GitHub Pages 用アーティファクトのアップロード	Pages 用アーティファクト	main + push 時のみ

DEPLOY-GITHUB-PAGES ジョブ

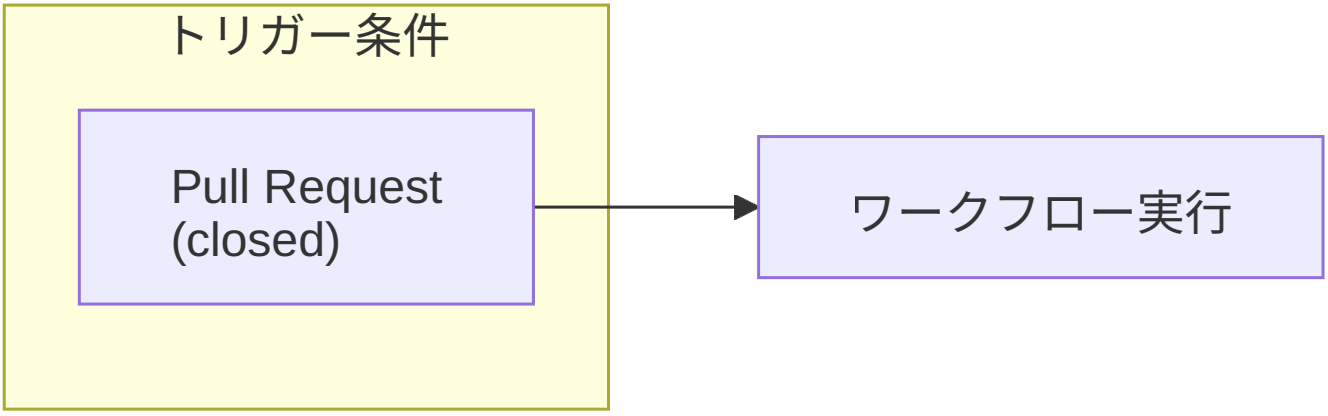
依存: `build` ジョブ完了後
実行条件: mainブランチへのpush時のみ

> **別ジョブとして分離している理由**
>
> GitHub Pagesへのデプロイは `build` ジョブに統合せず、別ジョブとして実装している。理由は以下の通り。
>
> 1. **Environments UI での追跡**: `environment` 設定により、GitHub UI の「Environments」タブでデプロイ履歴とステータスを確認可能
> 2. **リトライ容易性**: デプロイ失敗時に `build` を再実行せず、デプロイのみをリトライ可能
> 3. **公式推奨パターン**: GitHub 公式ドキュメントで推奨されている `upload-pages-artifact` → 別ジョブで `deploy-pages` というパターンに準拠

6.2.3 Close Preview ワークフロー

PRがクローズされた際に、Azure SWAのプレビュー環境を削除する専用ワークフロー。

実行条件

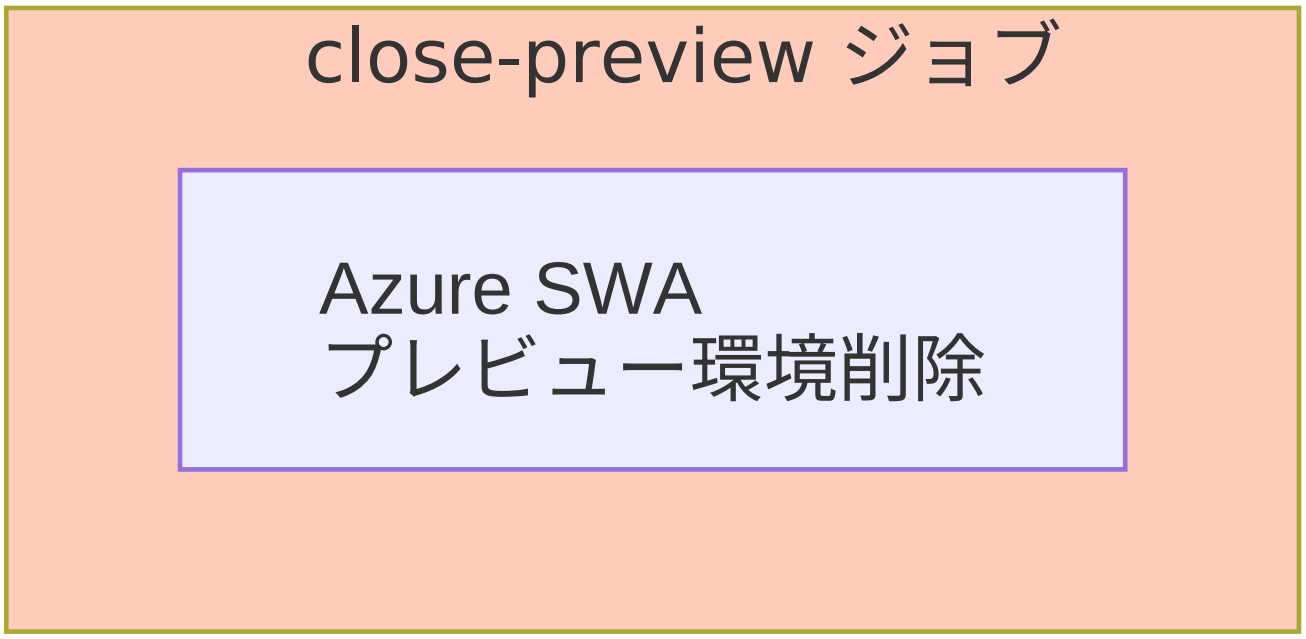


トリガー	条件
`pull_request`	closed

権限

```yaml
permissions:
  contents: read  # リポジトリ読み取りのみ
```

ジョブ構成



ジョブ詳細

CLOSE-PREVIEW ジョブ

Azure Static Web Appsのプレビュー環境を削除する。

ステップ	説明	使用アクション
プレビュー環境のクローズ	プレビュー環境の削除	`Azure/static-web-apps-deploy@v1`

6.2.4 Sync Role ワークフロー

実行条件

トリガー	条件
`schedule`	毎週月曜日 03:00 UTC（日本時間 12:00）
`workflow_dispatch`	手動実行

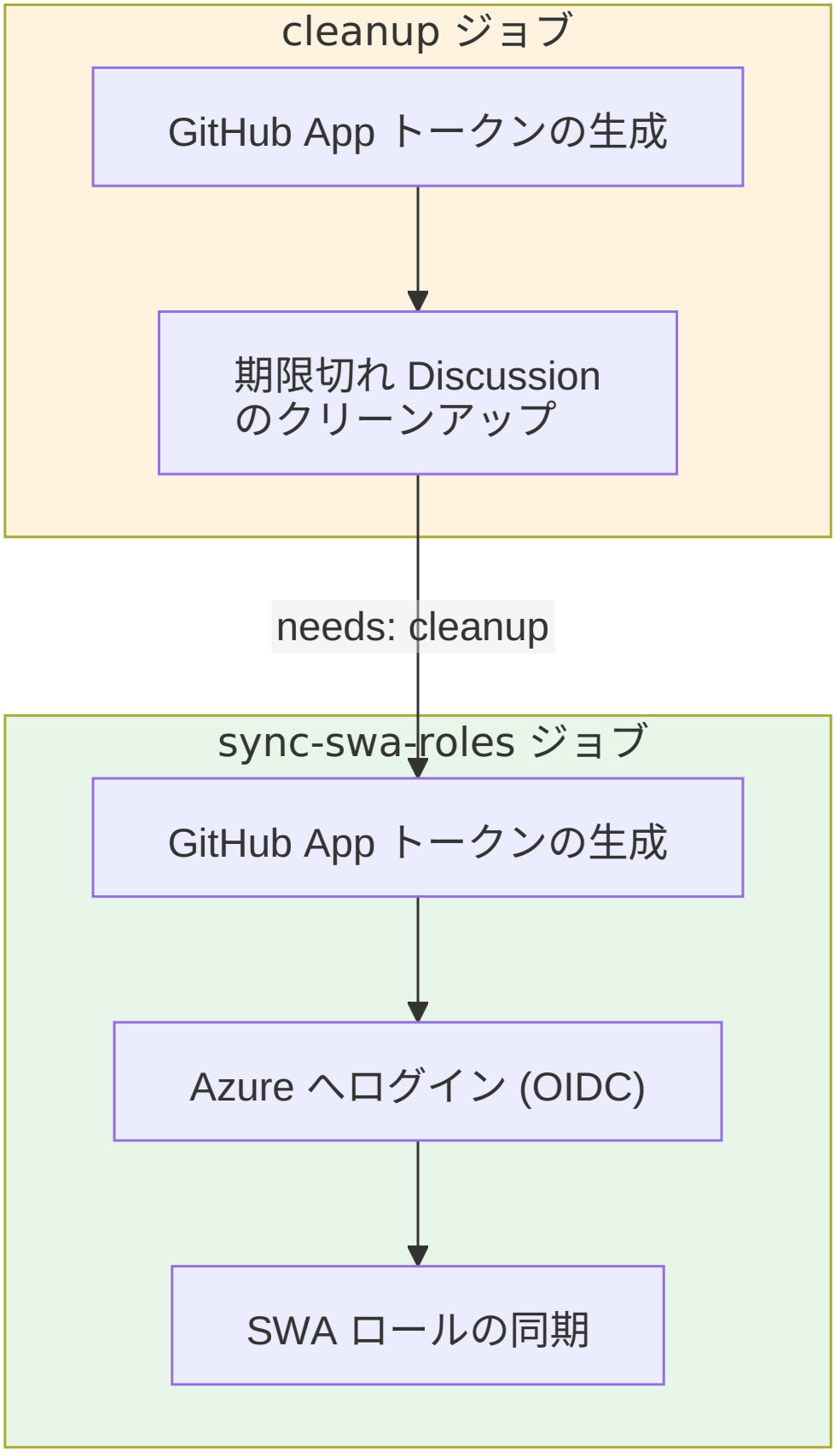
権限

```yaml
permissions:
  id-token: write    # OIDC トークン取得（Azure 認証用）
  contents: read     # リポジトリ読み取り
  discussions: write # Discussions への書き込み
```

環境変数

変数	説明	ソース
`SWA_NAME`	Azure Static Web App 名	`vars.AZURE_SWA_NAME`
`SWA_RG`	リソースグループ名	`vars.AZURE_SWA_RESOURCE_GROUP`
`DISCUSSION_CATEGORY`	招待用カテゴリ名	`Invitation` (固定値)

ジョブ構成



ジョブ詳細

CLEANUP ジョブ

期限切れの招待Discussionを削除する。

ステップ	説明	使用アクション
GitHub App トークンの生成	GitHub App からインストールトークンを生成	`actions/create-github-app-token@v1`
期限切れ Discussion のクリーンアップ	期限切れ招待を削除	`nuitsjp/swa-github-discussion-cleanup@v1`

SYNC-SWA-ROLES ジョブ

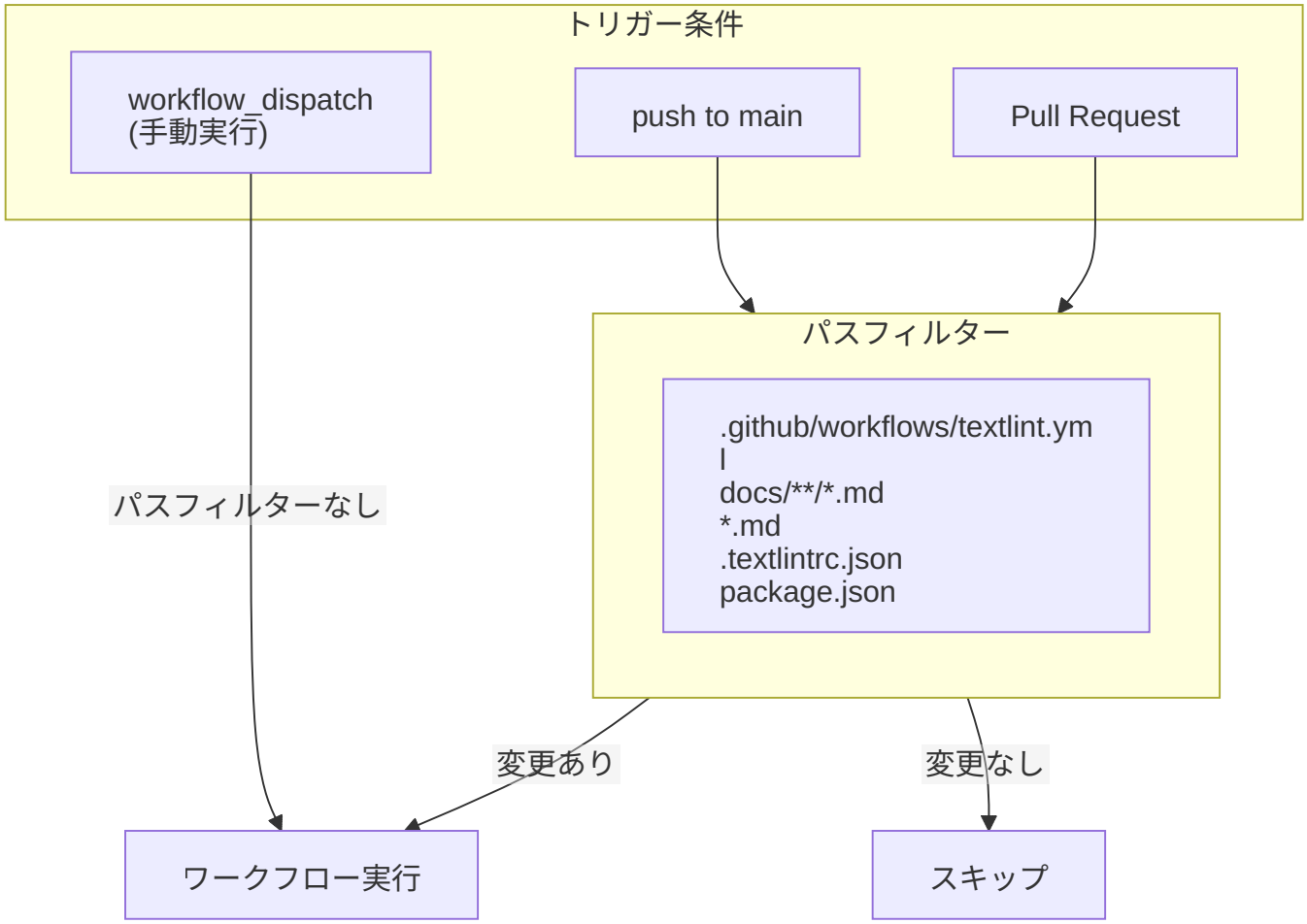
依存: `cleanup` ジョブ完了後

GitHubリポジトリ権限に基づきAzure SWAのロールを同期し、招待URLをDiscussionsに登録する。

ステップ	説明	使用アクション
GitHub App トークンの生成	GitHub App からインストールトークンを生成	`actions/create-github-app-token@v1`
Azure へログイン (OIDC)	OIDC でパスワードレスログイン	`azure/login@v2`
SWA ロールの同期	ロール同期と招待 URL 登録	`genai-docs/swa-github-role-sync@v1`

6.2.5 textlint ワークフロー

実行条件

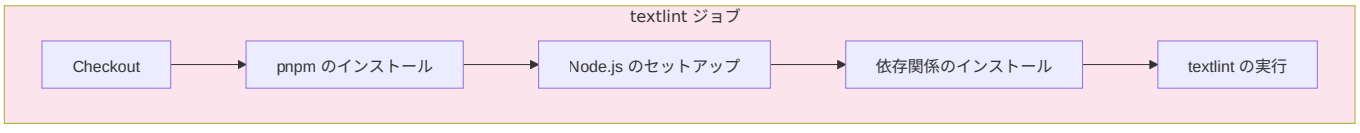


トリガー	条件	パスフィルター
`push`	`main` ブランチへのプッシュ	`.github/workflows/textlint.yml`, `docs/**/*.md`, `*.md`, `.textlintrc.json`, `package.json`
`pull_request`	すべてのタイプ	同上
`workflow_dispatch`	手動実行	なし（常に実行）

権限

```yaml
permissions:
  contents: read  # リポジトリ読み取りのみ
```

ジョブ構成



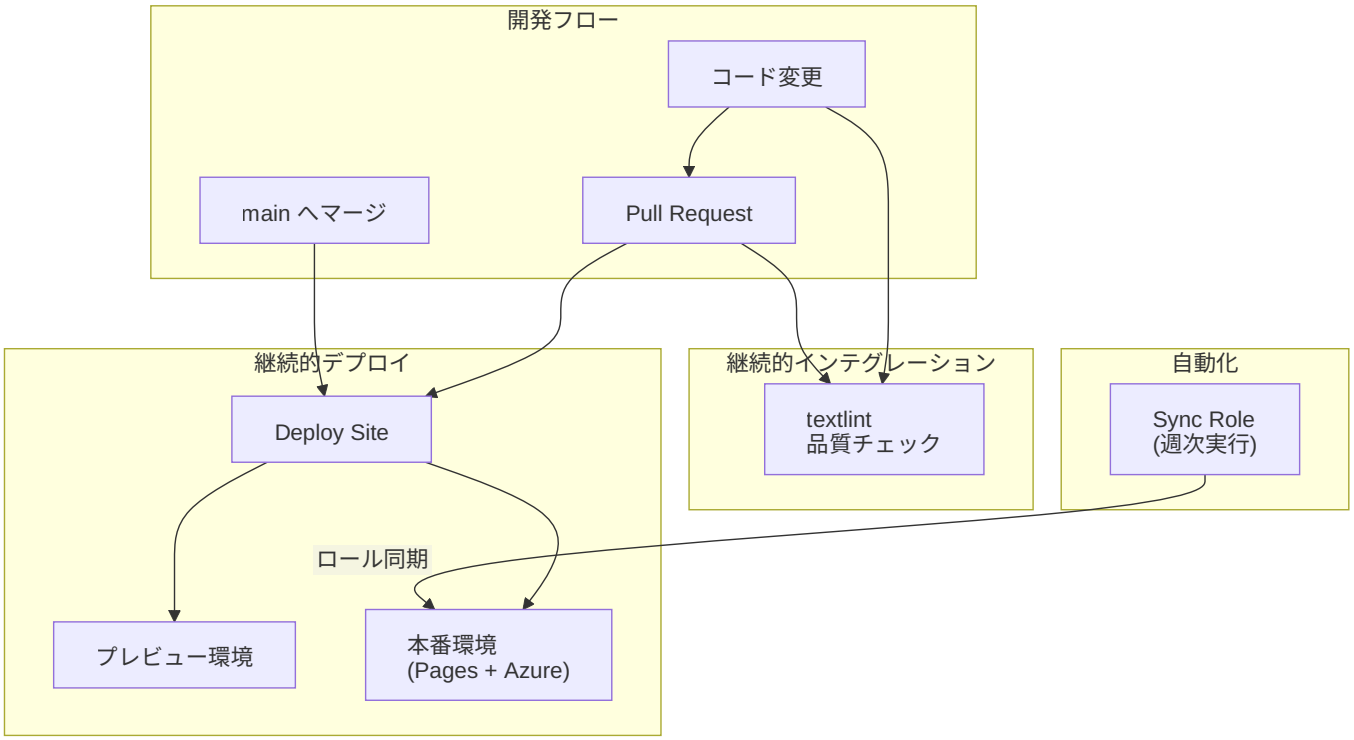
ジョブ詳細

TEXTLINT ジョブ

Markdown文書の日本語品質をチェックする。

ステップ	説明	備考
Checkout	リポジトリのチェックアウト	-
pnpm のインストール	pnpm パッケージマネージャー	-
Node.js のセットアップ	Node.js 20 環境構築	pnpm キャッシュ有効
依存関係のインストール	pnpm install	`--frozen-lockfile` で厳密インストール
textlint の実行	pnpm run lint:text	エラー時はワークフロー失敗

6.2.6 ワークフロー間の関係



6.2.7 関連ドキュメント

- デプロイ構成 - デプロイ先のActionへの参照リンク
- テキスト校正 - textlintのルール設定
- クラウド環境構築 - Azure / GitHubリソースの構築手順

6.3 テキスト校正

このリポジトリーでは、textlintの設定と依存関係を単一の基点に集約し、VS Code・CLI・CIの3経路で同一ルールの校正を実現している。

6.3.1 アーキテクチャーの要点

- ルールの単一ソース化（`.textlintrc.json`）
- 除外対象の一元管理（`.textlintignore`）
- 依存関係の固定（`package.json` と `pnpm-lock.yaml`）
- 実行経路の分離とルール共通化

6.3.2 共通ルールの中核

textlint本体と各種ルールセットは `package.json` に集約し、実行時は `.textlintrc.json` を必ず読み込む設計である。これにより、エディター内とCLI/CIでの差分を作らない。

```
ルール定義: .textlintrc.json
除外設定: .textlintignore
依存固定: package.json + pnpm-lock.yaml
```

6.3.3 利用シーン別の実行経路

1. VS Codeのリアルタイム校正

VS Code拡張は `.vscode/settings.json` で `.textlintrc.json` を明示参照している。入力時（`onType`）にtextlintが走り、ルールはローカルの `node_modules` と設定ファイルから解決される。

- 設定参照の明示（`textlint.configPath: ".textlintrc.json"`）
- 実行タイミングの統一（`textlint.run: "onType"`）
- 自動修正の無効化（`textlint.autoFixOnSave: false`）

2. CLIによるローカル全文書の校正

`pnpm run lint:text` がtextlintを起動し、`"**/*.md"` を対象に校正する。CLIも `.textlintrc.json` と `.textlintignore` を同じく参照するため、VS Codeと同一ルールで検出される。自動修正を適用する場合は `pnpm run lint:text:fix` を使う。

```
pnpm run lint:text      # 校正のみ
pnpm run lint:text:fix  # 自動修正を適用
```

3. CI（GitHub Actions）による校正

`.github/workflows/textlint.yml` がPR/Push時にtextlintを実行する。`pnpm install --frozen-lockfile` で依存を固定し、`pnpm run lint:text` でローカルと同一コマンドを実行するため、CIでも同じルールが保証される。

```
pnpm install --frozen-lockfile
pnpm run lint:text
```

6.3.4 同一ルールを維持する設計

3つの経路はいずれも `.textlintrc.json` を参照し、依存は `pnpm-lock.yaml` で固定される。変更点を一か所に集約することで、エディター・CLI・CIの結果が一致する構成になっている。

7. サンプル

7.1 サンプル

このセクションでは、本プロジェクトで使用する各種ツールの利用方法とサンプルを紹介する。

7.1.1 ツールの使い分け

Mermaid（推奨）

テキストベースで図を記述するツールである。以下の理由から、基本的にはMermaidの使用を推奨する。

- AIが解釈・生成しやすい
- バージョン管理が容易（差分が見やすい）
- コードレビューがしやすい

Draw.io（限定的に使用）

VS CodeのDraw.io Integration拡張を使用する。

使用する場面

- Mermaidでは表現が難しい複雑な図
- 接続線の位置を細かく制御したい場合
- 自由なレイアウトが必要な場合

注意

AIの解釈が困難になるため、どうしても必要な理由があるときのみ使用すること。

Marp

プレゼンテーション資料を作成するツールである。MkDocsと統合してWebで表示し、PDFはMkDocsのto-pdfプラグインで統合的に出力される。

7.1.2 サンプルページ

- Mermaid - フローチャート、シーケンス図、状態遷移図などのサンプル
- Draw.io - VS Code拡張を使ったSVG図の作成方法
- Marp - プレゼンテーション資料の作成方法

7.2 Mermaid

Mermaidはテキストベースで図を記述できるツールである。本プロジェクトでは推奨ツールとして位置づけている。

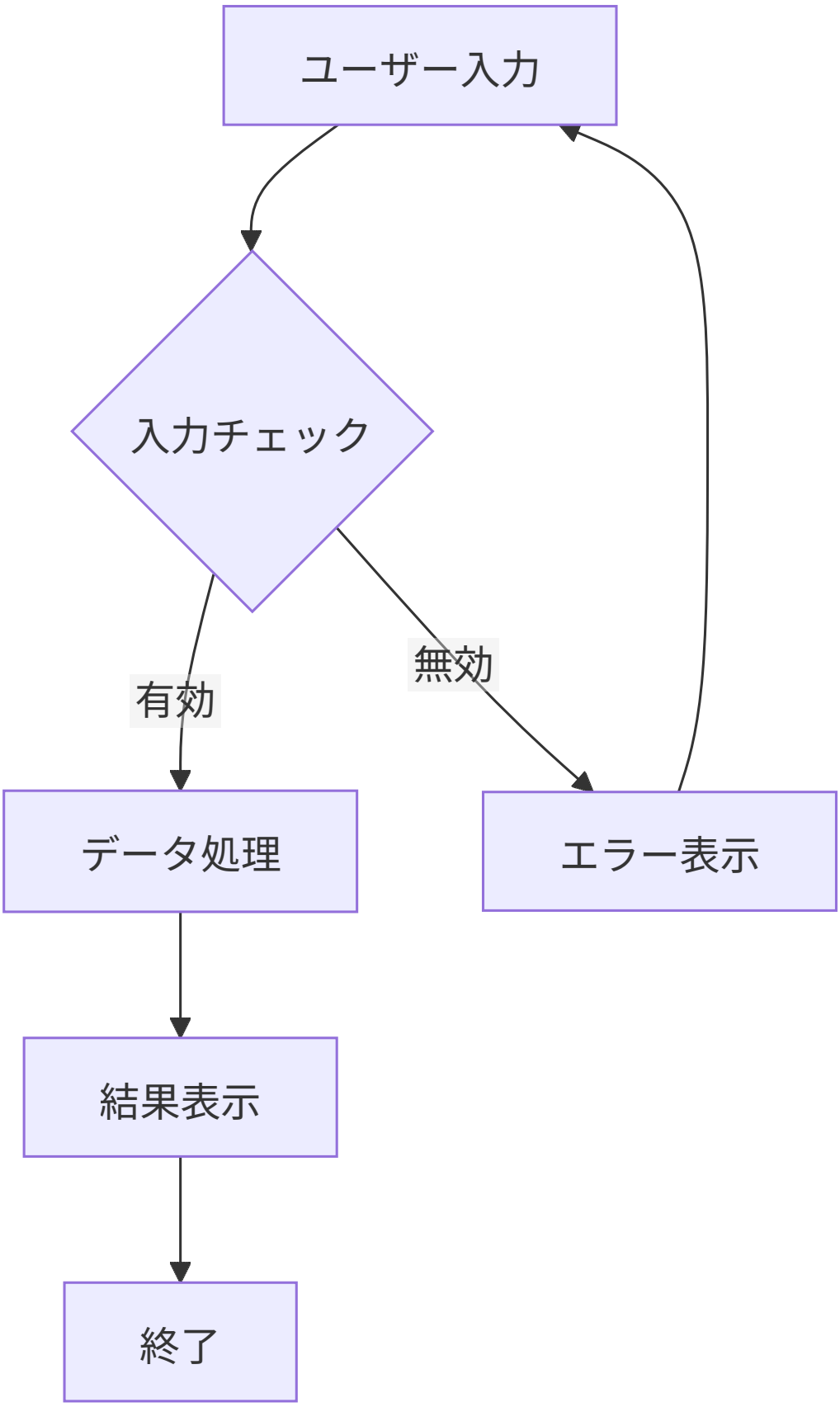
7.2.1 基本的な使い方

Markdownファイル内で、以下のようにコードブロックを記述する。

````
```mermaid
graph TD
    A[開始] --> B[処理]
    B --> C[終了]
```
````

7.2.2 フローチャート

処理の流れを表現する基本的な図である。

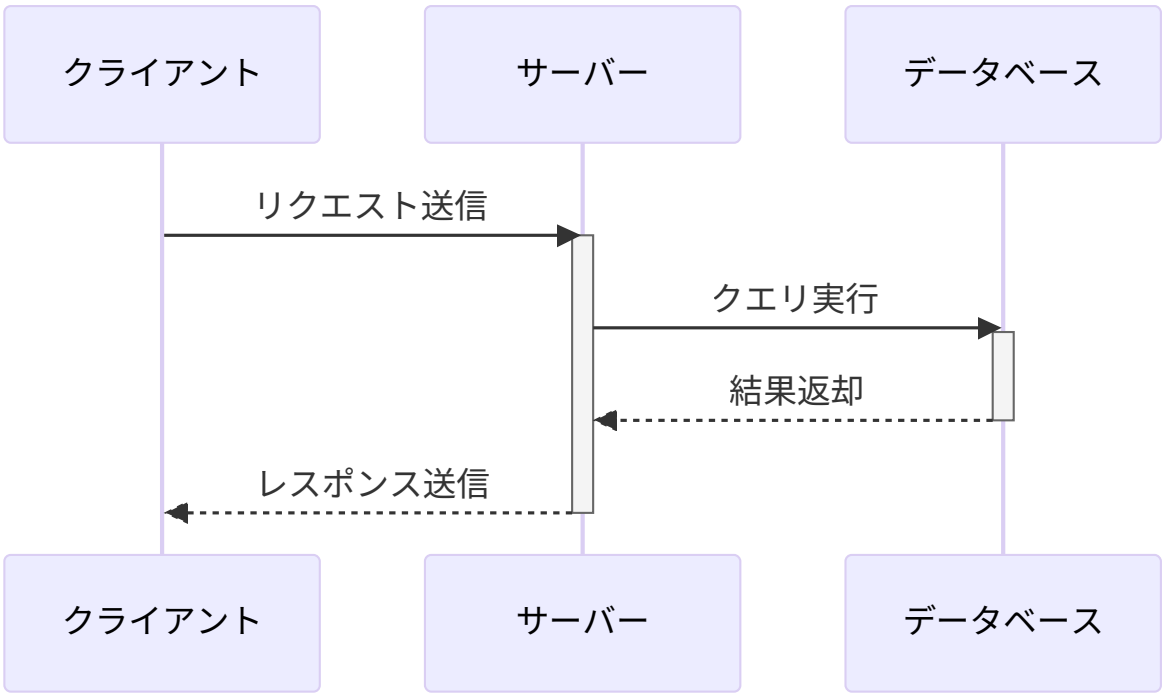


記法のポイント

- `graph TD`: 上から下へ流れる図（Top to Down）
- `graph LR`: 左から右へ流れる図（Left to Right）
- `[テキスト]`: 四角形のノード
- `{テキスト}`: ひし形のノード（条件分岐）
- `-->`: 矢印
- `-->|ラベル|`: ラベル付き矢印

7.2.3 シーケンス図

オブジェクト間のやり取りを時系列で表現する。

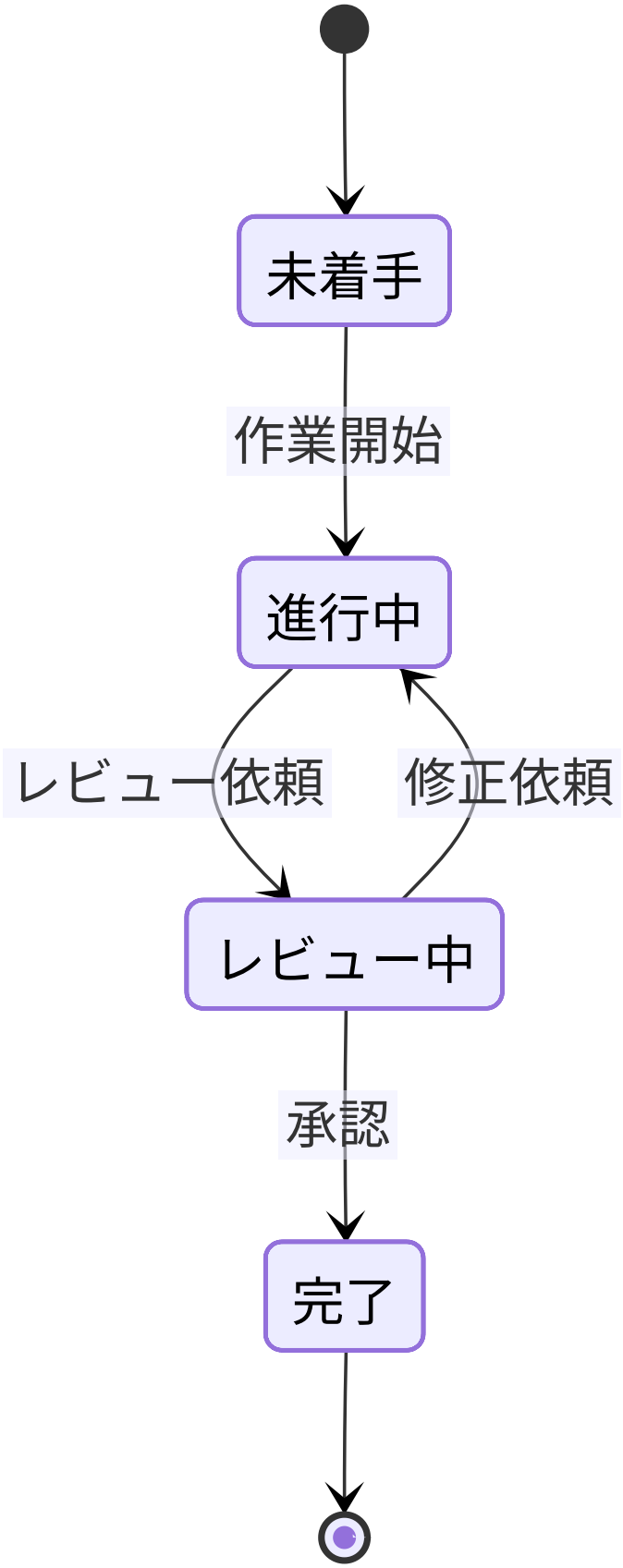


記法のポイント

- `participant`: 参加者を定義
- `->>`: 同期メッセージ
- `-->>`: 応答メッセージ
- `activate/deactivate`: 活性化バーの表示

7.2.4 状態遷移図

システムやオブジェクトの状態の変化を表現する。

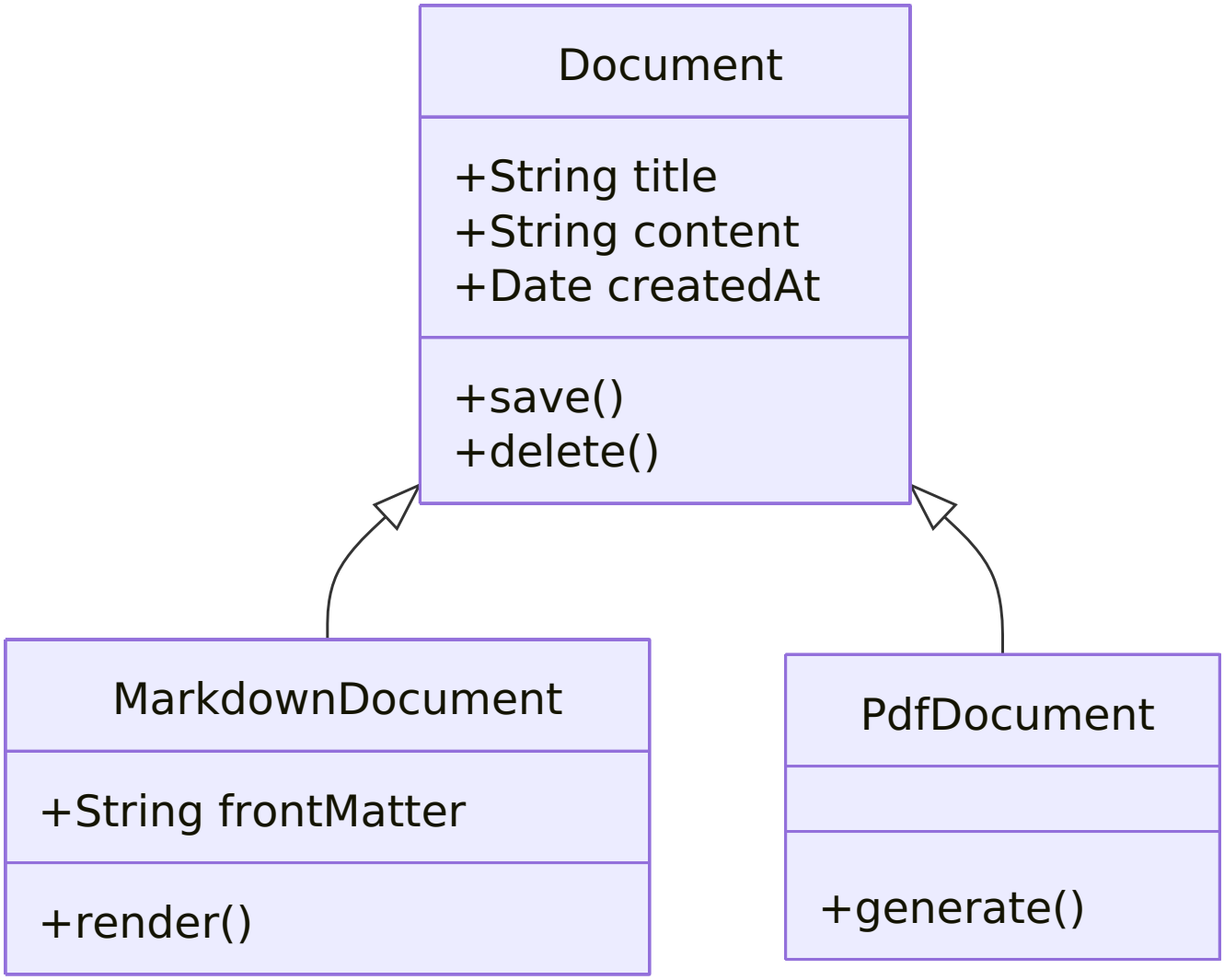


記法のポイント

- `[*]`: 開始・終了状態
- `状態名`: 状態を定義
- `-->`: 遷移

7.2.5 クラス図

クラスの構造と関係を表現する。

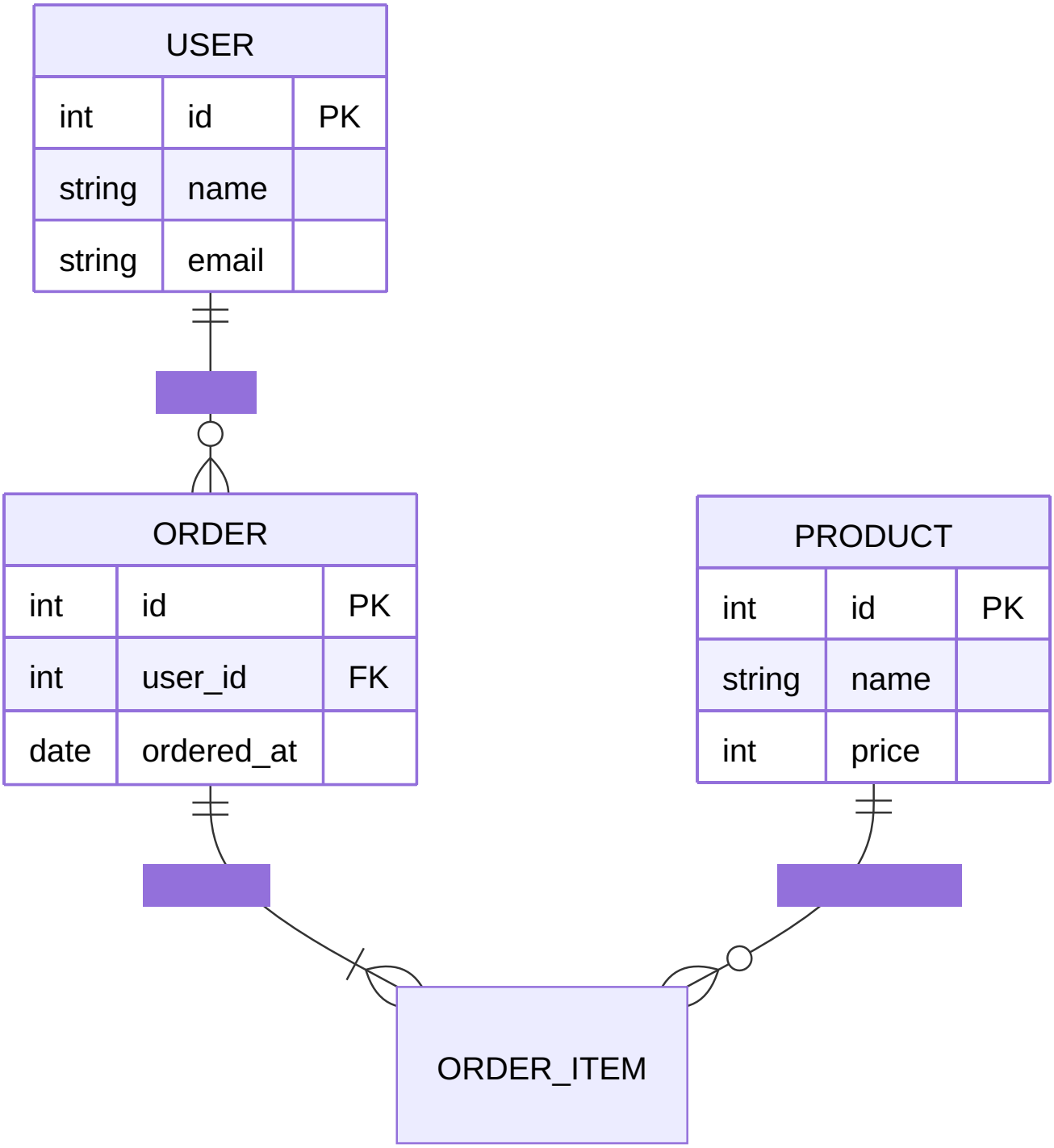


記法のポイント

- `class クラス名`: クラスを定義
- `+`: public
- `-`: private
- `<|--`: 継承関係

7.2.6 ER図

データベースのエンティティと関係を表現する。



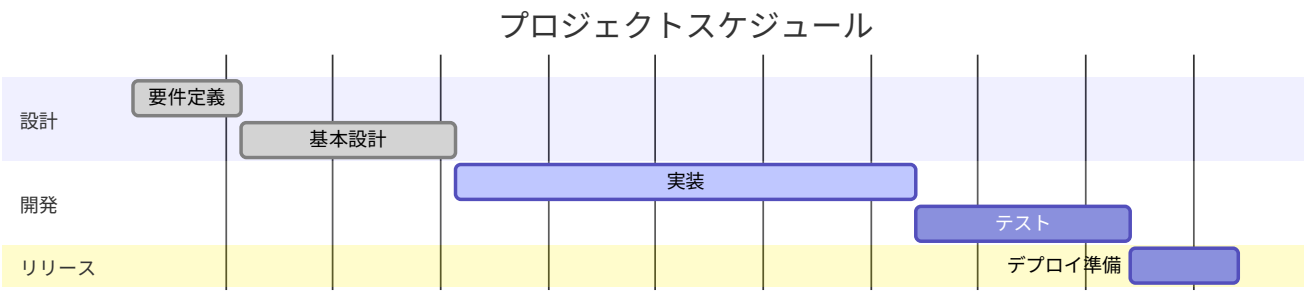
記法のポイント

- `||--o{` : 1対多の関係
- `||--|{` : 1対1以上の関係
- `PK` : 主キー

- FK : 外部キー

7.2.7 ガントチャート

プロジェクトのスケジュールを表現する。



7.2.8 参考リンク

- Mermaid公式ドキュメント
- Mermaid Live Editor

7.3 Draw.io

Draw.ioはGUIベースで図を作成できるツールである。VS CodeのDraw.io Integration拡張を使用して、エディター内で直接編集できる。

7.3.1 使用する場面

Draw.ioは以下のような場面で使用する。

- Mermaidでは表現が難しい複雑な図
- 接続線の位置を細かく制御したい場合
- 自由なレイアウトが必要な場合
- アイコンやイラストを含む図

7.3.2 注意事項

Draw.ioはどうしても必要な理由があるときのみ使用すること。

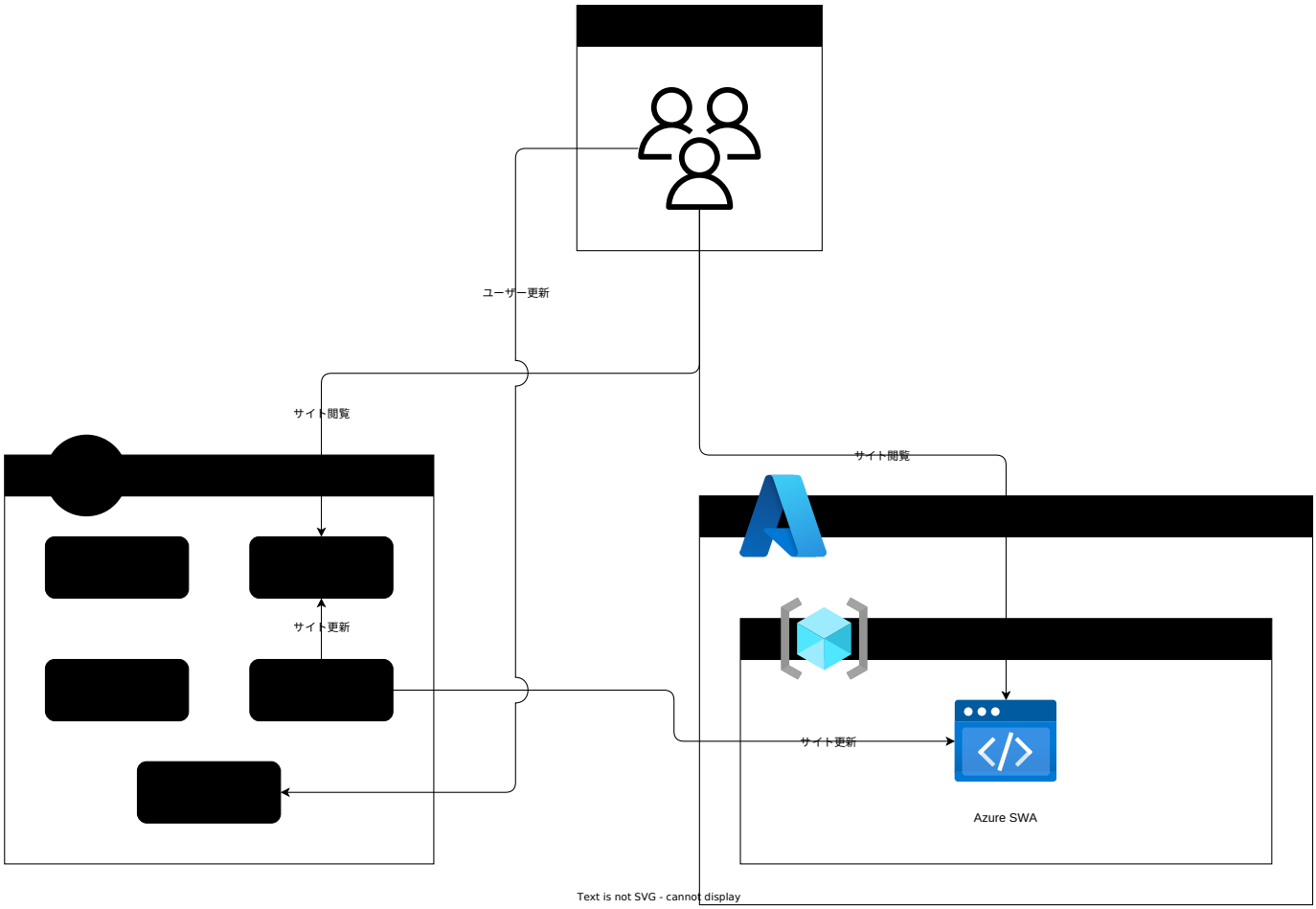
理由：

- バイナリ形式に近いため、AIによる解釈・生成が困難
- 差分の確認が難しい
- テキストベースのMermaidと比較してバージョン管理が複雑

基本的な図（フローチャート、シーケンス図、状態遷移図など）はMermaidを使用すること。

7.3.3 サンプル

以下はDraw.ioで作成した図の例である。



7.3.4 セットアップ

VS Code拡張のインストール

1. VS Codeを開く
2. 拡張機能マーケットプレイスで「Draw.io Integration」を検索
3. 「Draw.io Integration」（hediet.vscode-drawio）をインストール

7.3.5 ファイル形式

Draw.ioで作成するファイルは `.drawio.svg` 形式を使用する。

SVG形式を使用する理由：

・ベクター形式のため、Webで閲覧しやすく拡大縮小が可能

・PDF化する際に、MkDocsのsvg-to-pngプラグインでPNGに変換できる

7.3.6 使い方

新規作成

1. 対象の文書と同じディレクトリに `.drawio.svg` ファイルを作成
2. 例： `overview.md` の図なら `overview-architecture.drawio.svg`

3. VS Codeでファイルを開くと、Draw.ioエディターが起動
4. 図を作成・編集
5. 保存（Ctrl+S）

Markdownでの参照

同じディレクトリにあるため、相対パスで簡潔に参照できる。

```
![アーキテクチャ図](./overview-architecture.drawio.svg)
```

7.3.7 ファイル配置のルール

Draw.ioファイルは、関連する文書と同じディレクトリに配置する。

```
docs/
├── section-a/
│   ├── overview.md
│   ├── overview-architecture.drawio.svg
│   └── detail/
│       ├── api-design.md
│       └── api-design-flow.drawio.svg
```

命名規則

`{文書名}-{図の内容}.drawio.svg`

- 文書名： 関連するMarkdownファイルの名前（拡張子なし）
- 図の内容： 図が表す内容を簡潔に表現
- 例： `api-design-flow.drawio.svg`, `system-architecture.drawio.svg`

この配置方式の理由

1. **近接性**: 文書と関連図が同じディレクトリにあり、関係が明確
2. **移動・削除の容易さ**: 文書を移動・削除する際、関連ファイルも一緒に扱える
3. **命名による関連付け**: ファイル名で所属する文書が明確
4. **スケーラビリティ**: 階層が深くなっても管理が破綻しない
5. **git追跡可能**: `docs/assets/images/` はMkDocs生成画像用でgitignore対象のため、文書と同じ場所に配置

7.3.8 参考リンク

- Draw.io公式サイト
- VS Code Draw.io Integration

7.4 Marp

Marpはマークダウンからプレゼンテーション資料を作成するツールである。本プロジェクトではMkDocsと統合してWebで表示する。

7.4.1 概要

MarpはMarkdown形式でスライドを記述し、HTML/PDF/PPTXなどに変換できるツールである。

本プロジェクトでの特徴：

・MkDocsのページとしてWeb上で閲覧可能

・PDF出力はMkDocsのto-pdfプラグインで統合的に行う

・スライド単体のPDF出力も可能

7.4.2 基本構文

スライドの区切り

スライドは`---`（水平線）で区切る。

```
---
marp: true
---

# スライド1

最初のスライドの内容

---

# スライド2

2枚目のスライドの内容

---

# スライド3

3枚目のスライドの内容
```

フロントマター

スライドの先頭にフロントマターを記述して設定を行う。

```
---
marp: true
theme: default
paginate: true
header: 'ヘッダーテキスト'
footer: 'フッターテキスト'
---
```

主な設定項目：

設定	説明
`marp: true`	Marpスライドとして認識させる
`theme`	テーマ（default, gaia, uncover）
`paginate`	ページ番号を表示
`header`	ヘッダーテキスト
`footer`	フッターテキスト
`size`	スライドサイズ（16:9, 4:3）

7.4.3 スタイリング

ディレクティブ

スライドごとにスタイルを変更できる。

```
---

<!-- _class: lead -->
# タイトルスライド

中央寄せのリードスライド

---

<!-- _backgroundColor: #123 -->
<!-- _color: white -->
# 背景色を変更

このスライドは背景が暗い

---
```

画像の配置

```
---

# 画像の配置

![width:300px](../assets/images/my-diagram.drawio.svg)

<!-- 背景画像として使用 -->
![bg right:40%](../assets/images/background.png)

---
```

画像の指定方法：

- `width:300px` - 幅を指定
- `height:200px` - 高さを指定
- `bg` - 背景画像として使用
- `bg right:40%` - 右側40%に背景画像

7.4.4 サンプルスライド

marp-sample.htmlは、以下のテキストをMarkdownファイルとして保存し、Marpで作成したサンプルスライドをHTML形式に変換したものである。

```
---
marp: true
theme: default
paginate: true
```

```
---

# プロジェクト概要

生成AI時代のドキュメント基盤

---

## 技術スタック

- MkDocs + Material for MkDocs
- Mermaid（ダイアグラム）
- Draw.io（複雑な図）
- Marp（プレゼンテーション）

---

## まとめ

- テキストベースでドキュメント管理
- バージョン管理が容易
- 自動ビルド・デプロイ
```

7.4.5 VS Code拡張

Marp for VS Code拡張を使用すると、エディター内でプレビューを確認できる。

1. VS Codeを開く
2. 拡張機能で「Marp for VS Code」を検索
3. インストール

7.4.6 PDF出力

MkDocs統合でのPDF

MkDocsサイト全体をPDF化する場合は、to-pdfプラグインを使用する。

```
MKDOCS_PDF=1 uv run mkdocs build
```

スライド単体のPDF

Marp CLIを使用してスライド単体をPDF化できる。

```
# Marp CLIのインストール
npm install -g @marp-team/marp-cli

# PDFに変換
marp slides.md --pdf
```

HTMLに変換

```
marp slides.md --html
```

7.4.7 ファイル配置

Marpスライドはdocsディレクトリ内の任意の場所に配置できる。

```
docs/
├── presentations/
│   ├── project-overview.md
│   └── technical-design.md
└── samples/
    └── marp.md
```

7.4.8 参考リンク

- Marp公式サイト
- Marp for VS Code
- Marpit Markdown

7.5 プロジェクト概要

生成AI時代のドキュメント基盤

7.5.1 技術スタック

- MkDocs + Material for MkDocs
- Mermaid（ダイアグラム）
- Draw.io（複雑な図）
- Marp（プレゼンテーション）

7.5.2 まとめ

- テキストベースでドキュメント管理
- バージョン管理が容易
- 自動ビルド・デプロイ

8. スライド

8.1 スライド

本セクションでは、Marpで作成したプレゼンテーション資料を掲載する。

8.1.1 スライド一覧

- Marp - Marpの基本構文、スタイリング、VS Code連携、出力方法のガイド
- 生成AI時代のドキュメント基盤 - Markdown中心のドキュメント管理と基盤構築についての発表資料

8.2 生成AI時代のドキュメント基盤

8.2.1 ドキュメントの重要性

生成AIの活用が進み、ドキュメントの重要性の再認識

8.2.2 生成AI活用にも注目

- AWS - Kiro
- GitHub - Spec Kit
- Fission-AI - OpenSpec
- gotalab - cc-sdd

8.3 新たな課題の出現

8.3.1 ドキュメントの2つの課題

- ドキュメント体系
- ドキュメント基盤

8.3.2 ドキュメント体系

ソフトウェア開発プロセスの中で

- どのタイミングで
- 何を文章化し
- どうコードとコラボレーションするか

8.3.3 ドキュメント基盤

- どのように作成し
- どのようにレビューし
- どのように閲覧し
- どのように配布するか？

8.3.4 2つの視点

・ドキュメント体系 : どう構成し、どのように役割分担するか
・**ドキュメント基盤** : どのように作成・レビューし、どう閲覧し、どう配布するか

後者が本発表のスコープです。

8.3.5 本ドキュメント基盤の発端

・7年前、MS Officeから脱却を決意

8.3.6 本ドキュメント基盤の発端

・7年前、MS Officeから脱却を決意
・WordやExcelで「きれいに書く事」に時間をかける事が負担
・複数人で並列作業する事の難しさ
・コードと文書のバージョン同期の難しさ
・平行作業時のレビューの難しさ

8.3.7 6時間かけて書いたWordが崩壊してキレた

8.3.8 Markdownベース文書への移行を決意

8.3.9 本ドキュメント基盤の実績

・大手金融3社のSIプロジェクト
・Markdownベース
・800ページ超
・最長7年の持続性

これらの事例とノウハウを紹介します。

8.3.10 自己紹介

- T.B.D
- T.B.D

8.3.11 ドキュメント基盤要件

- AIフレンドリー - 単純な図の簡便な作成 - 複雑な図を正確に描画 - 表の簡便な作成 - 単一PDF出力
- git管理 - PRベースレビュー - PR時のステージング - CI/CD
- 検索を含めた高い閲覧性 - 高い拡張性 - 十分なセキュリティ - 低コスト - 保守ステージのコスト0

8.4 デモンストレーション

8.4.1 デモの流れ

1. Markdown ファイルの作成・閲覧
2. Mkdocs によるサイト閲覧・PDF出力
3. Mermaid / draw.io による作図
4. Copy To Markdown Excel アドインによる 表の管理
5. ソース と ドキュメント の一元管理
6. レビュー と ステージング環境
7. CI / CD
8. おすすめ３機能（MkDocs の検索 / Csv の埋め込み / Marp によるスライド作成）

8.4.2 PDF 出力の流れ

ドキュメンテーション戦略.pdf

8.5 Marp

8.5.1 概要

MarpはMarkdown形式でスライドを記述し、HTML/PDF/PPTXなどに変換できるツール。

本プロジェクトでの特徴：

・MkDocsのページとしてWeb上で閲覧可能

・PDF出力はMkDocsのto-pdfプラグインで統合的に行う

・スライド単体のPDF出力も可能

8.5.2 基本構文

スライドの区切り

スライドは `---`（水平線）で区切る。

```
# スライド1

最初のスライドの内容

---

# スライド2

2枚目のスライドの内容
```

フロントマター

スライドの先頭にフロントマターを記述して設定を行う。

```
---
marp: true
theme: default
paginate: true
header: 'ヘッダーテキスト'
footer: 'フッターテキスト'
---
```

主な設定項目：

設定	説明
`marp: true`	Marpスライドとして認識させる
`theme`	テーマ（default, gaia, uncover）
`paginate`	ページ番号を表示
`header`	ヘッダーテキスト
`footer`	フッターテキスト
`size`	スライドサイズ（16:9, 4:3）

8.5.3 スタイリング

ディレクティブ

スライドごとにスタイルを変更できる。

```
---

<!-- _class: lead -->
# タイトルスライド

中央寄せのリードスライド

---
```

```
<!-- _backgroundColor: #123 -->
<!-- _color: white -->
# 背景色を変更

このスライドは背景が暗い

---
```

画像の配置

```
![width:300px](../assets/images/my-diagram.drawio.svg)

<!-- 背景画像として使用 -->
![bg right:40%](../assets/images/background.png)
```

画像の指定方法：

- `width:300px` - 幅を指定
- `height:200px` - 高さを指定
- `bg` - 背景画像として使用
- `bg right:40%` - 右側40%に背景画像

8.5.4 VS Code拡張

Marp for VS Code拡張を使用すると、エディター内でプレビューを確認できる。

1. VS Codeを開く
2. 拡張機能で「Marp for VS Code」を検索
3. インストール

8.5.5 PDF出力

MkDocs統合でのPDF

MkDocsサイト全体をPDF化する場合は、to-pdfプラグインを使用する。

```
MKDOCS_PDF=1 uv run mkdocs build
```

スライド単体のPDF

Marp CLIを使用してスライド単体をPDF化できる。

```
# Marp CLIのインストール
npm install -g @marp-team/marp-cli

# PDFに変換
marp slides.md --pdf
```

HTMLに変換

```
marp slides.md --html
```

8.5.6 参考リンク

- Marp公式サイト
- Marp for VS Code
- Marpit Markdown