2025.6.13

MCPにおける認証・認可

KDDI Agile Development Center

自己紹介

ソフトウェアエンジニア

大坪 悠 Yu Otsubo

職歴：

新潟県出身。大学卒業後、2016年にKDDIに新卒入社。
通信設備の運用、社内データ基盤構築のPM、Webチャットボット開発などを経験する。
スタートアップへの転職を経て2024年にKAG入社。
KAGでは生成AIを活用したアプリケーション(A-BOSS - 本部長AI)開発に従事。

著書：

「K.A.G Tech Book」（KDDIアジャイル開発センター）
「やさしいMCP入門」（秀和システム）

@tubone24

tubone-project24.xyz

mugimugi.cutedog

趣味：

愛犬と遊ぶこと
個人開発

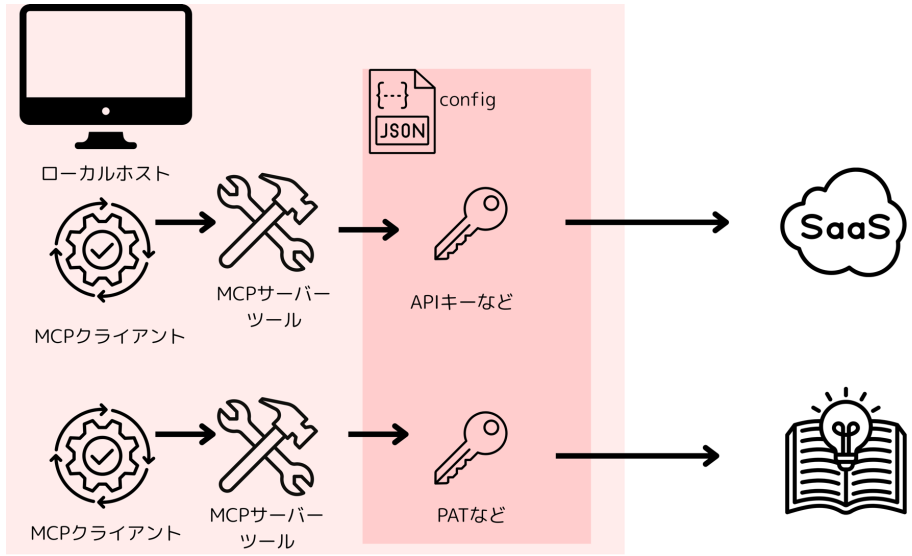
モットー：

早く小さく失敗する

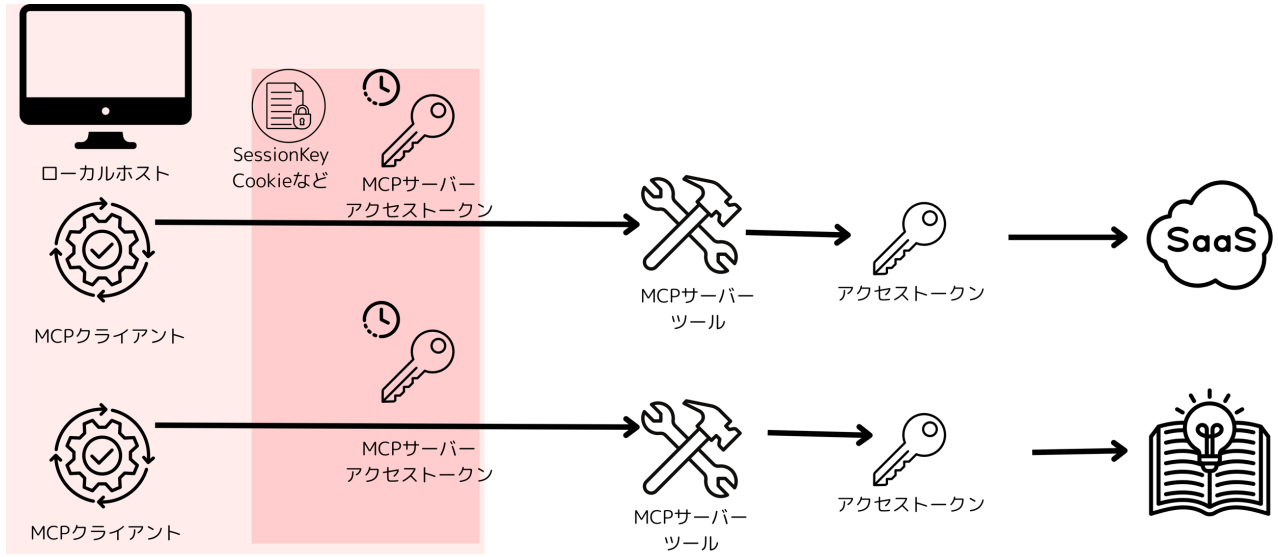
MCPにおける認証・認可の重要性

リモートMCPサーバーの普及や機微情報の扱いにおいて認証認可は重要

従来



これから



【💡ポイント💡】
ローカルMCPサーバーは接続コンフィグ(claude_desktop_config.jsonなど)に
強力なアクセストークンがそのまま記載されてしまう
一方でリモートMCPサーバーはMCPサーバー専用のアクセストークンを払い出す形

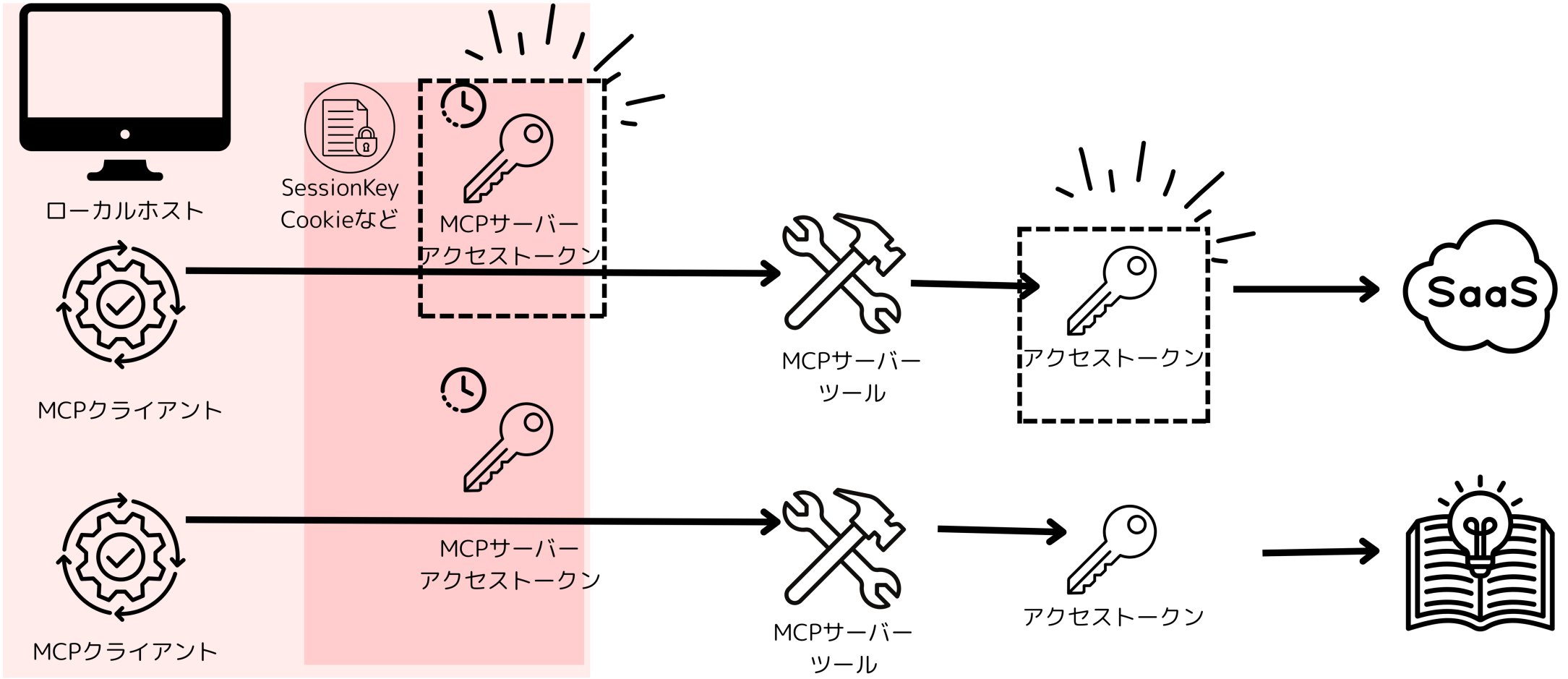
リモートMCPサーバーに強力なAPIキーを渡すことはできない!

APIは静的トークン = 漏洩時の影響が大きすぎる

- 定期的なキーローテをユーザーに求める必要がある
  - →誰もやらない
- (外部の)MCPサーバー内にキーを抱え込みにくい
  - DBのID/PWとかとてもじゃないけど持てない
- SaaS権限とは別にリソースサーバーとしてのMCPサーバーの権限制御ができない
  - SharePointを読み込めるけど、既存の記事は編集できない、など。

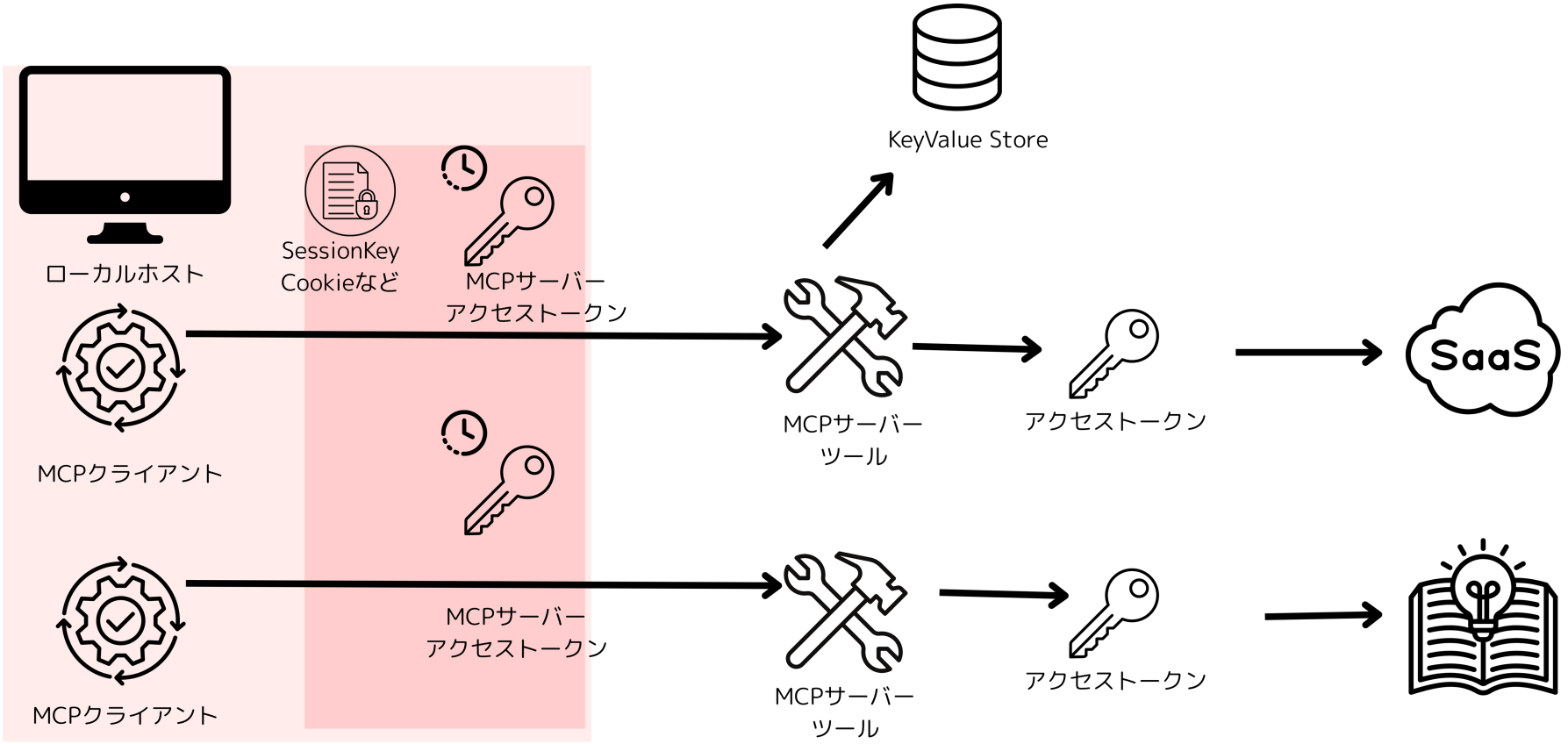
https://github.com/modelcontextprotocol/modelcontextprotocol/discussions/234
このあたりで議論されていました。

大きな特徴は、MCPクライアント vs サーバー間の認可と MCPサーバー vs 外部プロダクト間の認可の二つがあること



https://modelcontextprotocol.io/specification/draft/basic/security_best_practices#token-passthrough にて規定されます

なので、認証認可が完了すると、二つのアクセストークンが出来上がる状態。
加えて言うと、二つのアクセストークンを紐づける仕組みが必要になる。



よくある手法としてKeyValueでそれぞれの認証・認可のフローで必要な情報を保存しておく。
DBとしてTTLが設定できるとなおセッション感が出てよい。

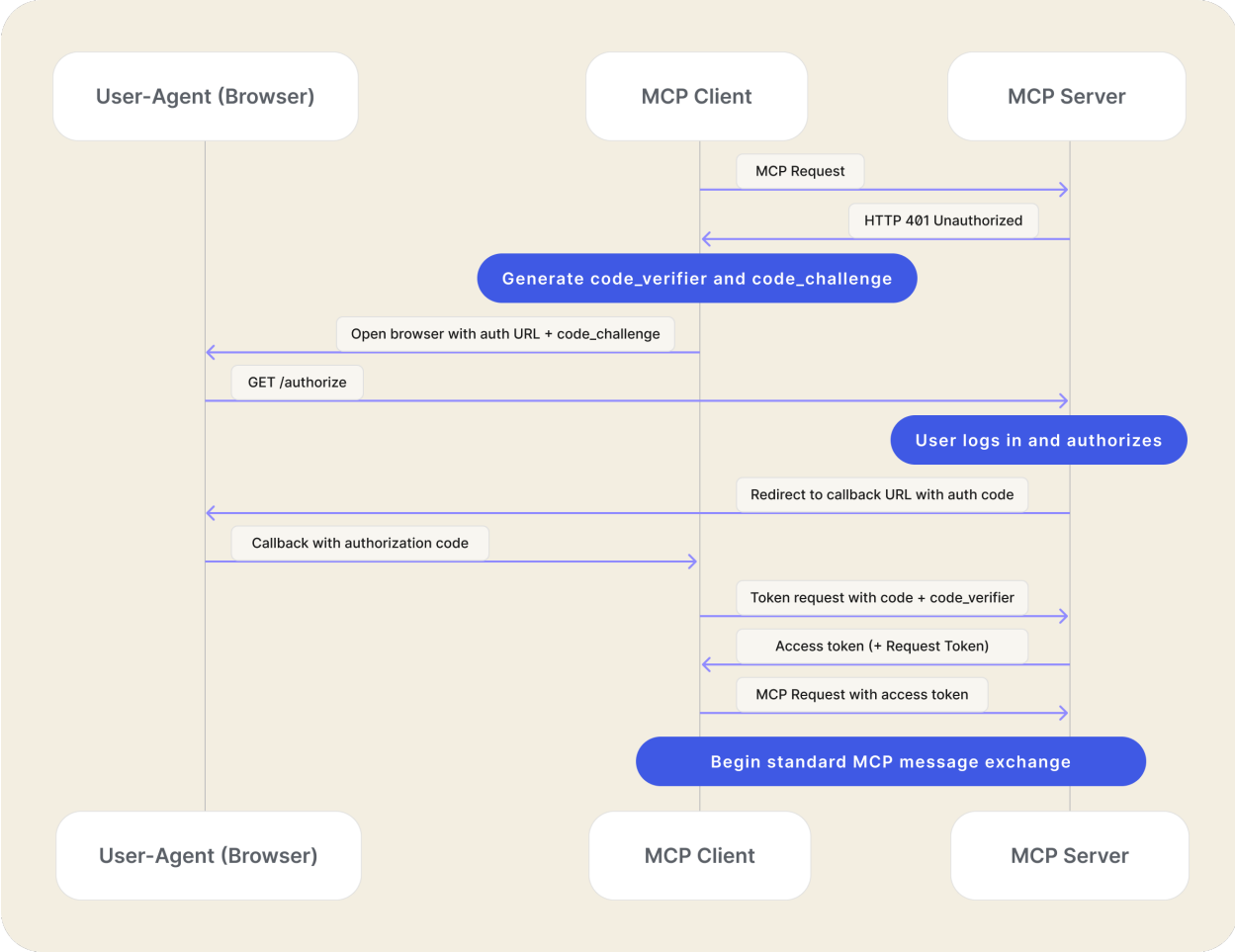
MCP(OAuth2.1)を支える技術

PKCE(ピクシーと読む)を用いた認可フロー(MUST)

Proof Key for Code Exchange by OAuth Public Clients

- 認可コードを横取りされない仕組み
  - 認可リクエスト者とトークンリクエスト者が同じである証明
    - 乱数とそのハッシュをそれぞれで突合させる

- パブリックなクライアントでも使える
  - クライアントを証明する方法=>クライアントクレデンシャル
  - Webページやスマホには埋め込めない欠点
  - クライアントを証明する代わりの方法



https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-12

図: https://auth0.com/blog/an-introduction-to-mcp-and-authorization/

メタデータディスカバリー(Authorization Server Metadata)

MCPサーバーにいざアクセスしても、どこを叩いて
認証・トークン取得すればいいかわからない。

なので、well-knownディレクトリとして、
認証に関する情報をまとめておくエンドポイントを用意する
ことが推奨される(ServerはSHOULD、ClientはMUST)

対応してない場合は、下記のデフォルトエンドポイントを
暗黙的に利用する

■デフォルトエンドポイント
- /authorize 認証エンドポイント
- /token トークンエンドポイント
- /register 動的クライアント登録エンドポイント

https://datatracker.ietf.org/doc/html/rfc8414

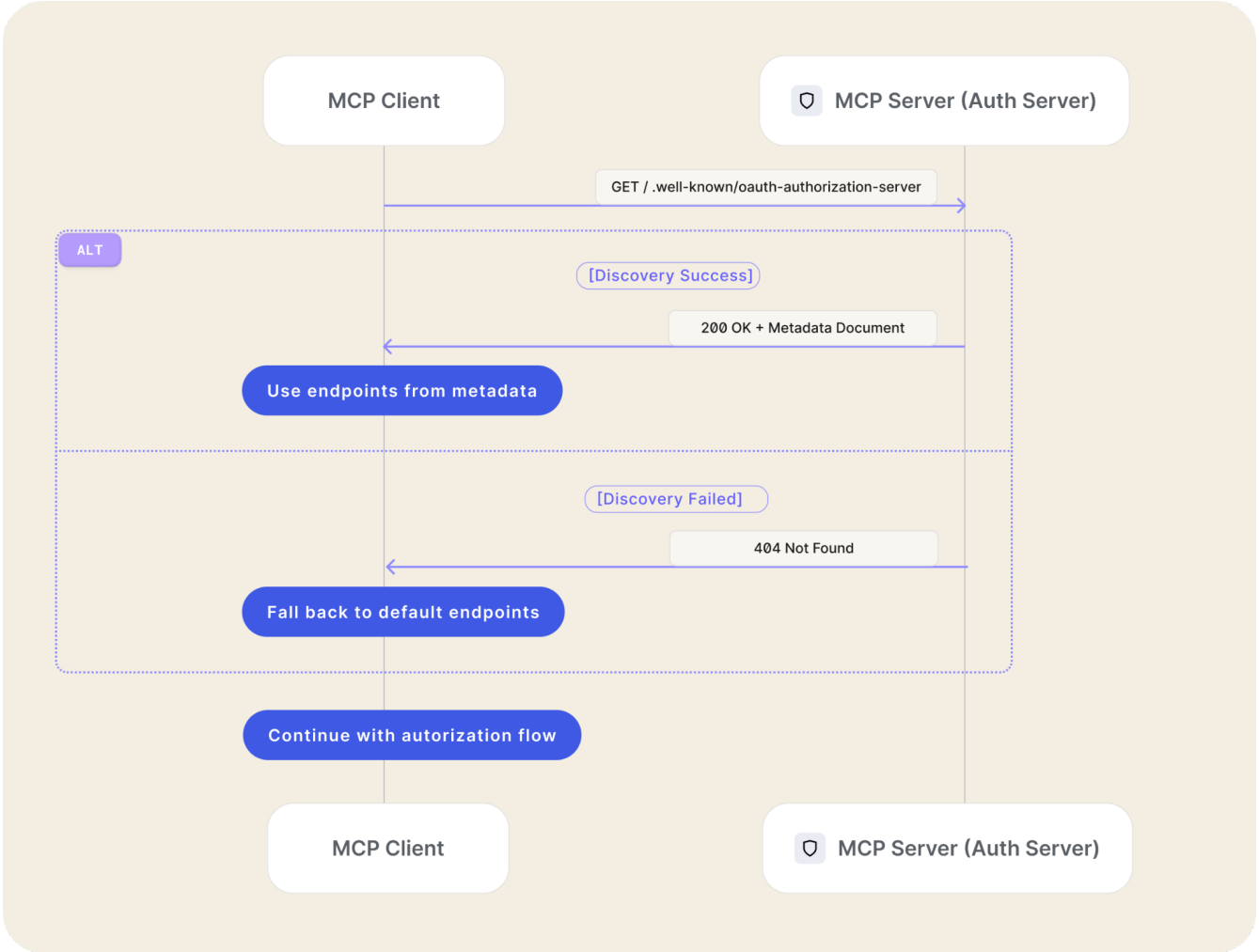


図: https://auth0.com/blog/an-introduction-to-mcp-and-authorization/

メタデータディスカバリー(Protected Resource Metadata - MUST)

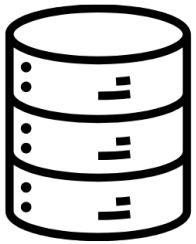
細かいですが、メタデータディスカバリーには、
- 認可サーバー側の機能
- リソースサーバー側の機能

の2つが別々のRFCで規定されている

MCPサーバーはリソースサーバーであるため、
リソースサーバー側のメタデータディスカバリの
実装はMUSTとなっている。

Authorization Server Metadata

認可サーバーが自身と対話するために
必要な情報を渡す



Protected Resource Metadata

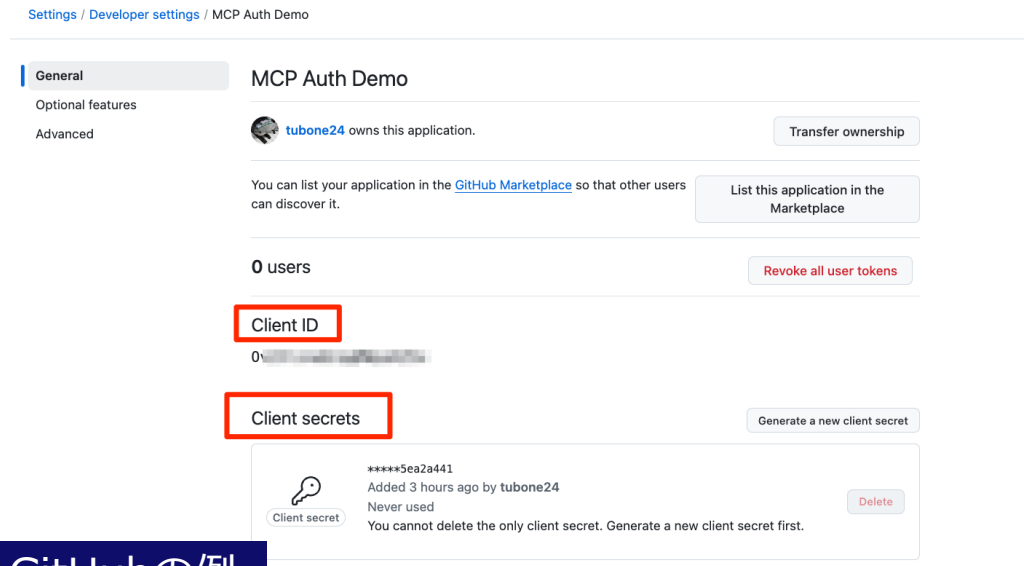
リソースサーバーが自身の保護リソースに
アクセスするために必要な情報を渡す

https://datatracker.ietf.org/doc/html/rfc9728

動的クライアント登録(DCR - SHOULD)

OAuthのクライアントを設定したことがある人はわかるかも

- 事前にクライアントIDとシークレットを登録する必要あり
- MCPクライアントに毎度設定するのは大変
- リクエストに応じてクライアントIDを発行する仕組み



GitHubの例

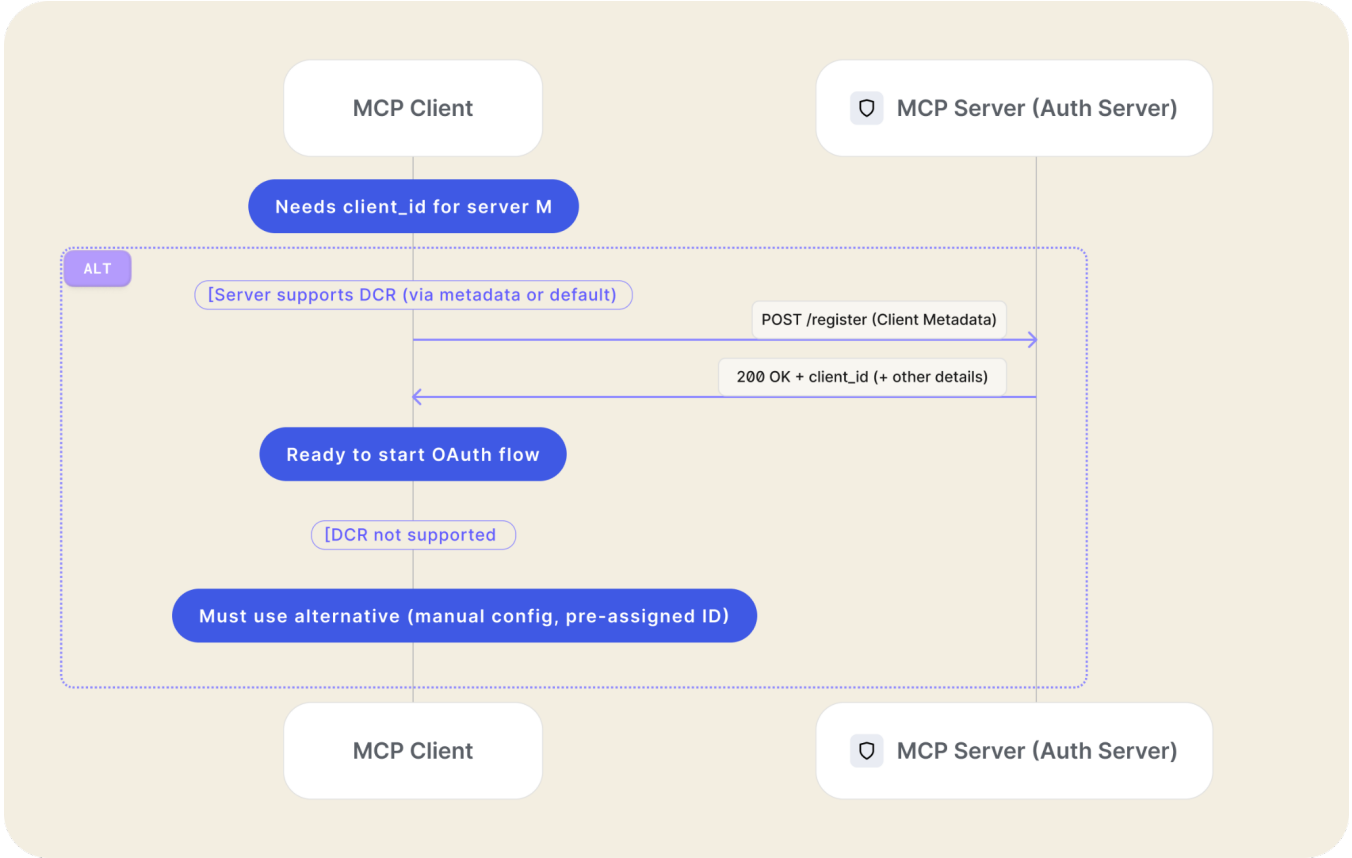


図: https://auth0.com/blog/an-introduction-to-mcp-and-authorization/

https://datatracker.ietf.org/doc/html/rfc7591

デモ

(自作のリモートMCPサーバー+Inspector)

今回のコードは次のレポジトリに保存してます
また、仕組みを作るのにCloudflare workersとKVを使いました

tubone24/**remote-mcp-oauth-github**

A Model Context Protocol (MCP) server for Claude.ai custom integrations, running on Cloudflare Workers with GitHub OAuth authentication.

1 Contributor　0 Issues　0 Stars　0 Forks

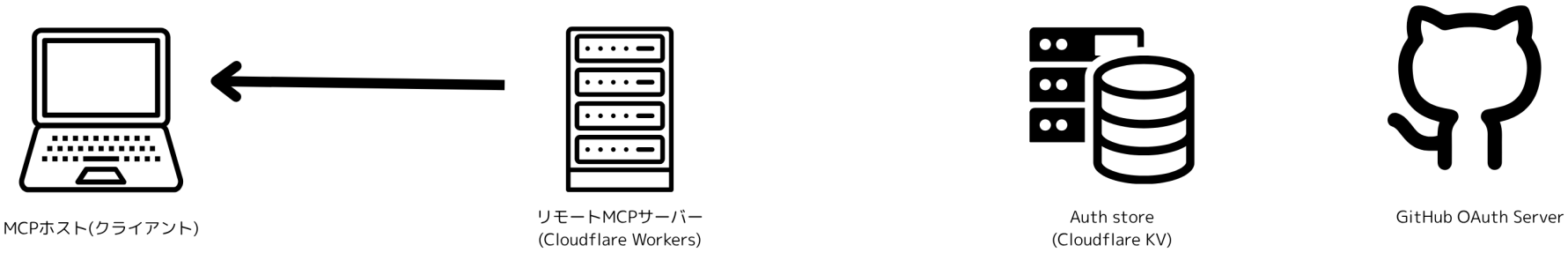
https://github.com/tubone24/remote-mcp-oauth-github

Streamable HTTPのエンドポイント(/mcp)にアクセスすると認証を求められる

Streamable HTTPでアクセスします～
https://mcp-github-oauth-demo.tubone24.workers.dev/mcp

ちゃんと認証してからにしてください～(401 Unauthorized)

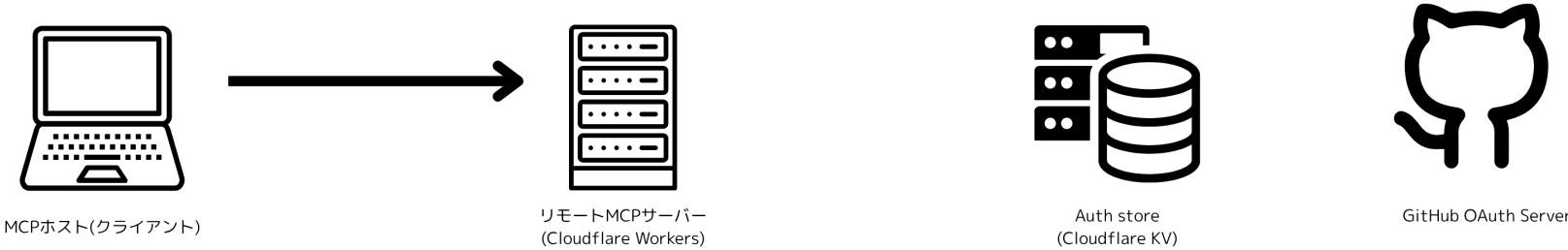
{"jsonrpc":"2.0","error":{"code":-32000,"message":"Authentication required"}}



認証に関するエンドポイントを一覧もらう
これがメタデータディスカバリー

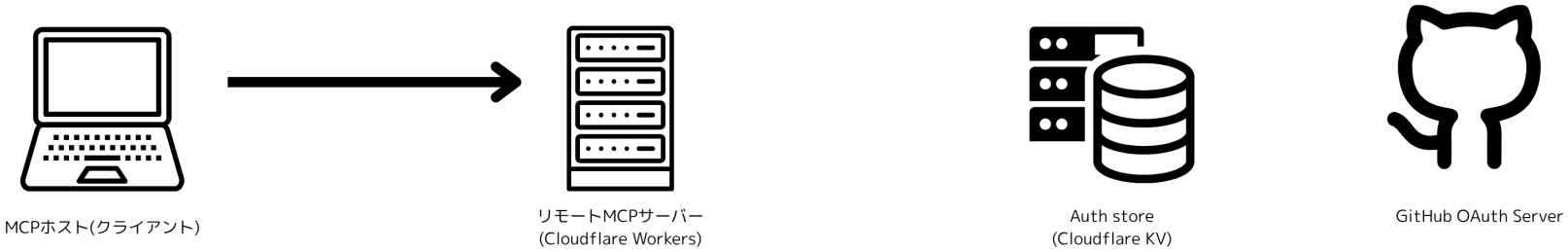
Wellknownにアクセスして認証のエンドポイント一覧もらいます～
/.well-known/oauth-authorization-server

一覧あげます～
{"issuer":"https://mcp-github-oauth-demo.tubone24.workers.dev"
," authorization_endpoint":"https://mcp-github-oauth-demo.tubone24.workers.dev/auth",
"token_endpoint":"https://mcp-github-oauth-demo.tubone24.workers.dev/token",
"registration_endpoint":"https://mcp-github-oauth-demo.tubone24.workers.dev/register" .... (中略)....
}



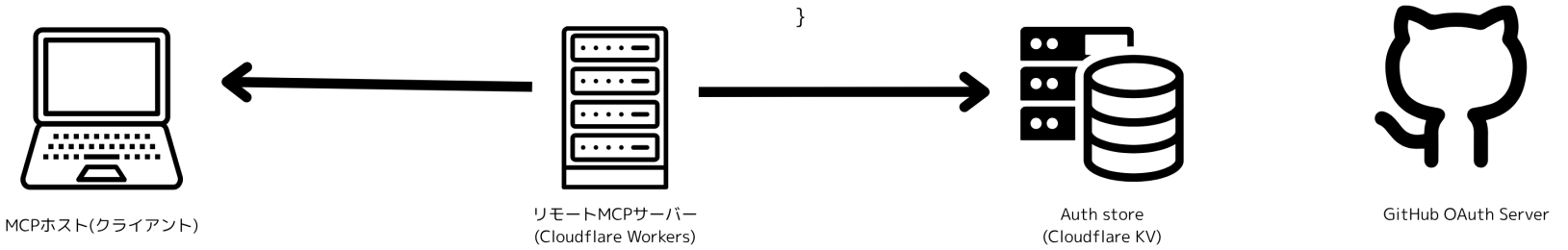
クライアントを自動で登録して以降のOAuthクライアントとして利用
これがDCR

今後OAuthで認証するためにもクライアントとして登録したいです！
/register

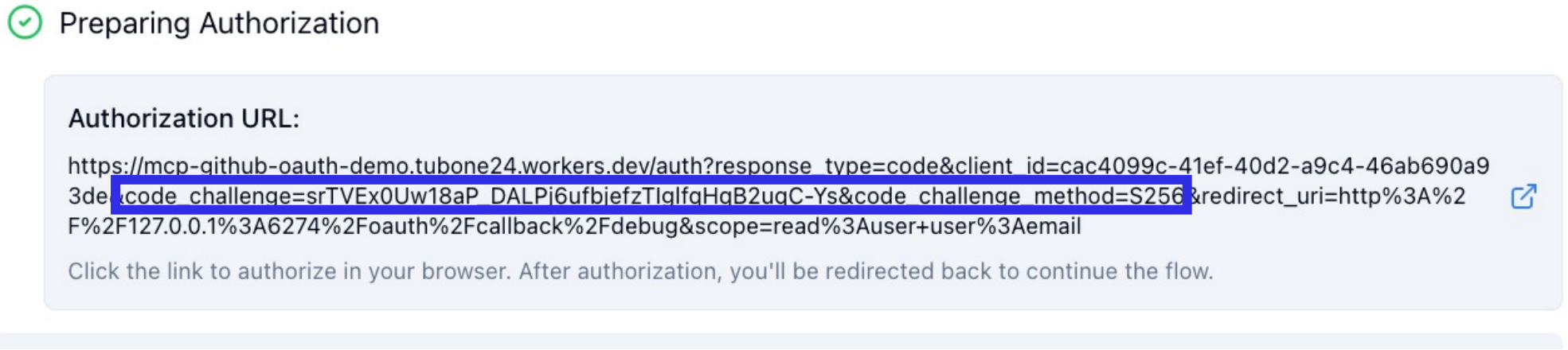


わかりました！あなたをクライアントxxxxとして登録しておきます。
スコープなども登録しておきました。よろしく！

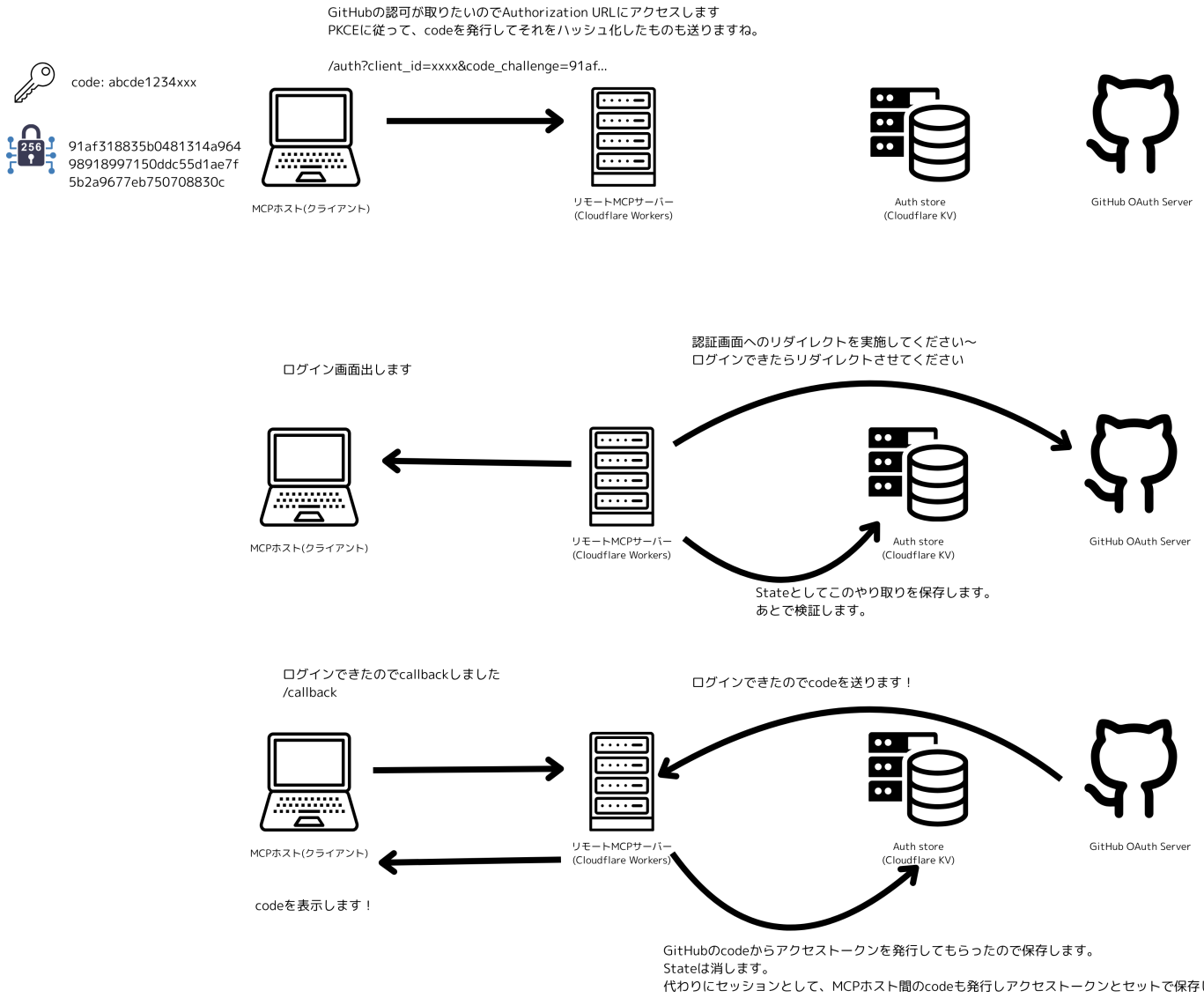
保存
{
“client_id”: xxxx
“client_secret”: yyyy
}



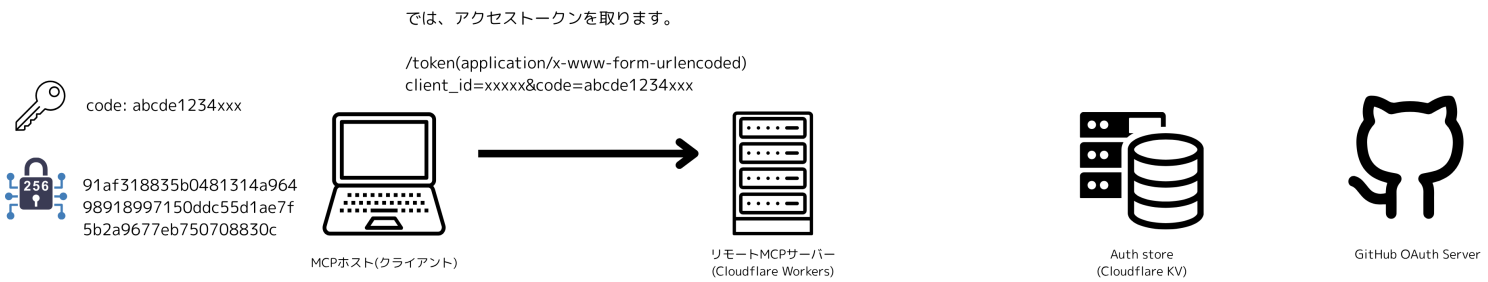
Authorization URLを発行
DCRで作成されたclient_idに加え、code_challengeが設定されている
これがPKCE

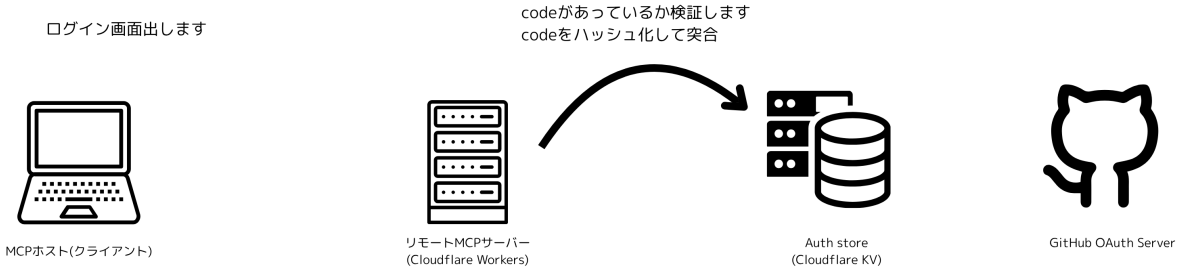


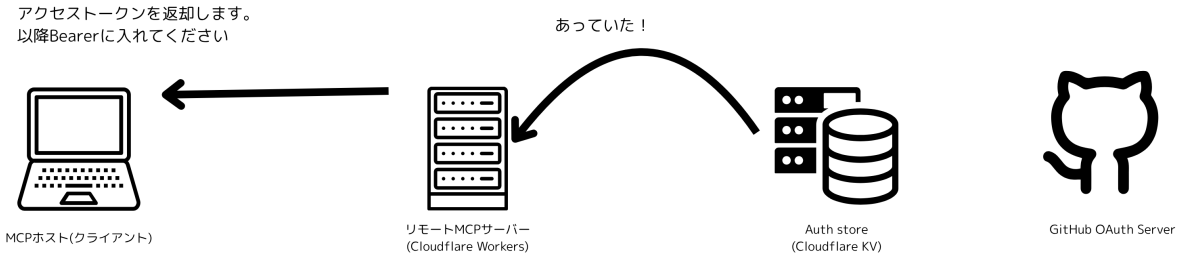
フローにするとこんな感じですが、詳細は割愛



トークン取得処理でcodeの検証を実施
これがPKCE







閑話休題

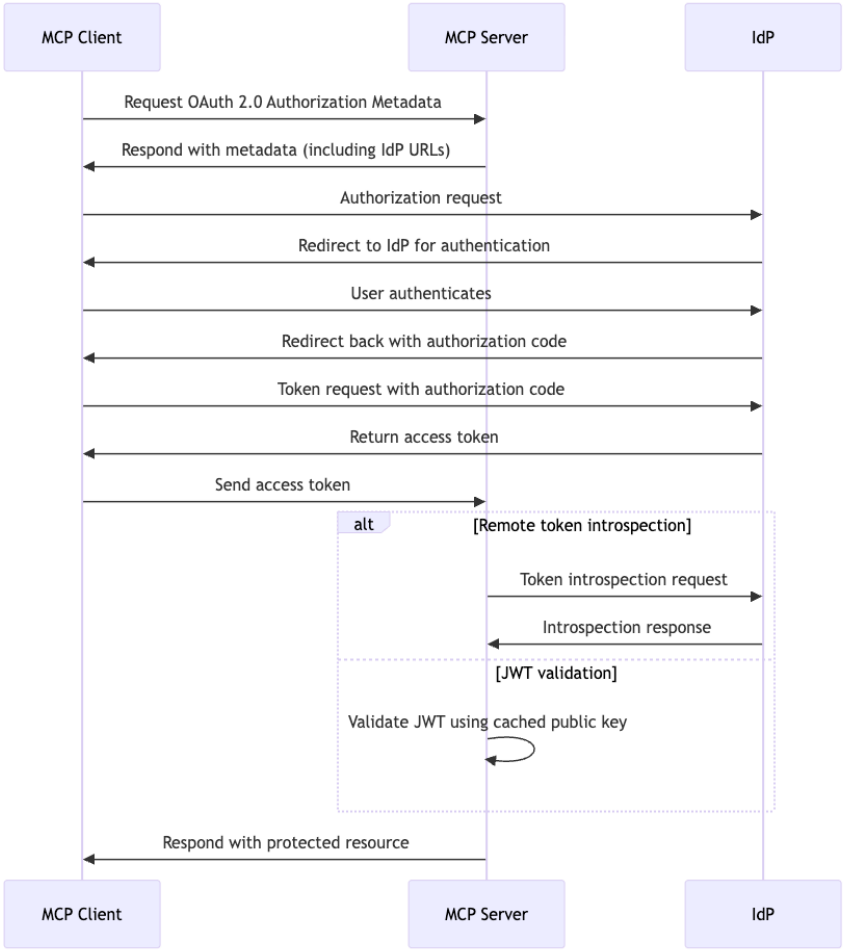
認可サーバーとリソースサーバーってくっつけていいの？

「ベストプラクティスじゃない」

が現状の規格だと、MCPサーバーがトークンの発行まで実施
ベストプラクティスに従えば、認可サーバーとリソースサーバーは分けるべき。

よって下記のIssueなどでより良い方式が今なお、
議論がされているところです。

https://github.com/modelcontextprotocol/modelcontextprotocol/issues/205



もっとMCPについて学びたい人は…

話題のMCPにやさしく入門しよう

技術のしくみ ＆ ビジネス展望がわかる！
図解たっぷりのコンパクトな本です。

7/1発売。アマゾンで予約受付中！

KDDIアジャイル開発センター
御田 稔（みのるん）、大坪 悠

KDDI Agile Development Center

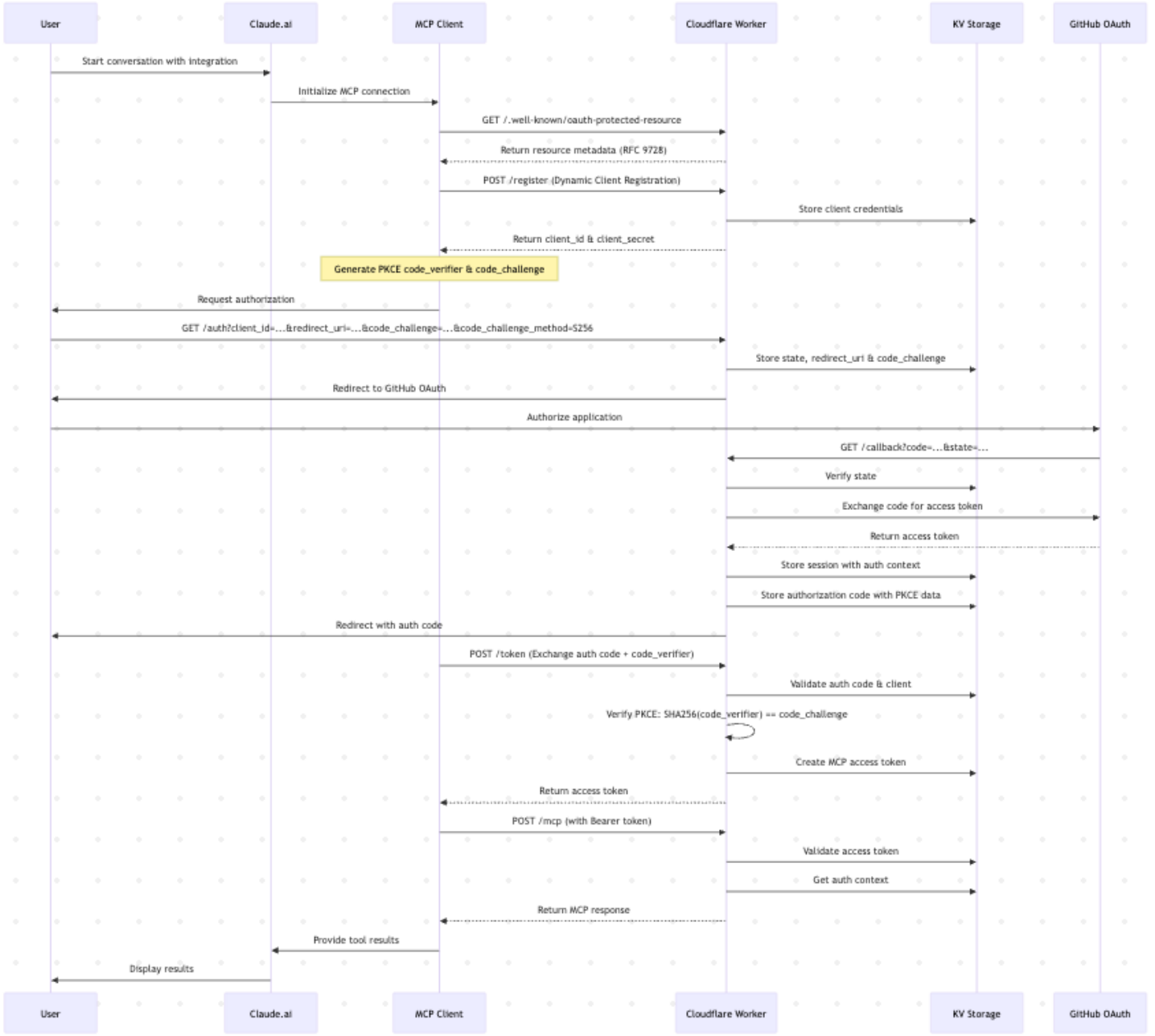
Be a Change Leader.

アジャイルに力を与え

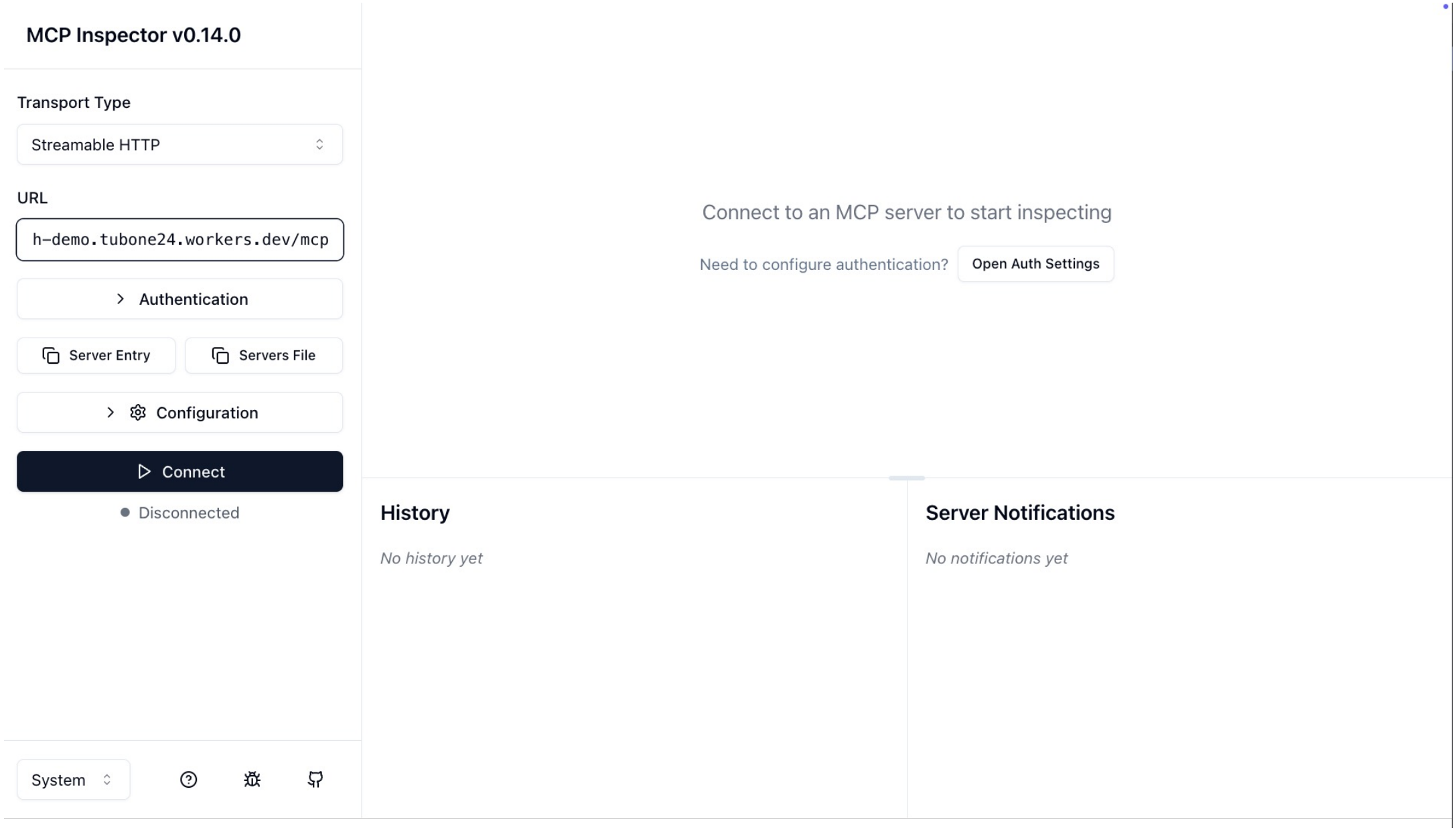
共に成長し続ける社会を創る

Appendix

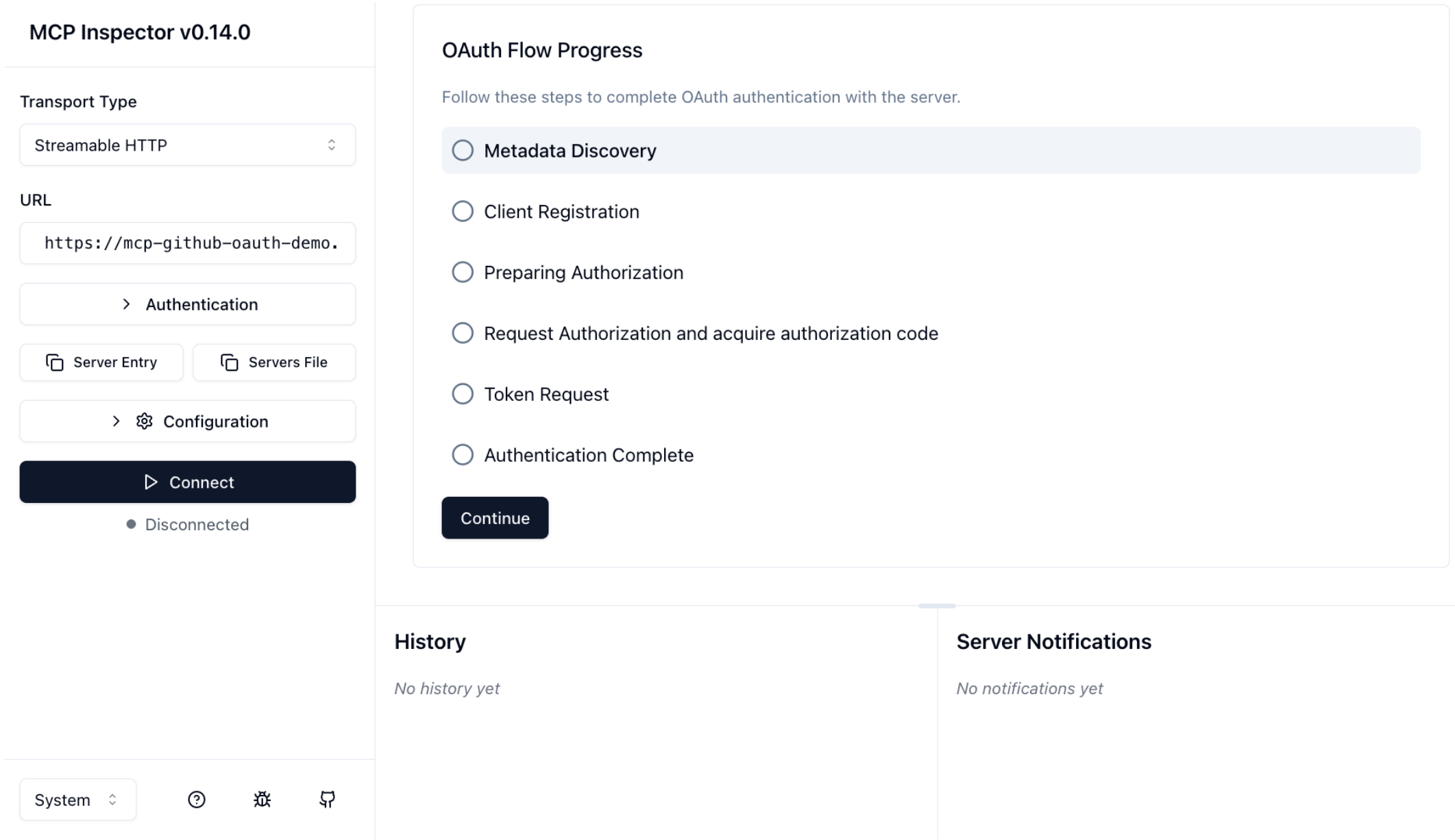
デモ - シーケンス


User
Claude.ai
MCP Client
Cloudflare Worker
KV Storage
GitHub OAuth
Start conversation with integration
Initialize MCP connection
GET /.well-known/oauth-protected-resource
Return resource metadata (RFC 9728)
POST /register (Dynamic Client Registration)
Store client credentials
Return client_id & client_secret
Generate PKCE code_verifier & code_challenge
Request authorization
GET /auth?client_id=...&redirect_uri=...&code_challenge=...&code_challenge_method=S256
Store state, redirect_uri & code_challenge
Redirect to GitHub OAuth
Authorize application
GET /callback?code=...&state=...
Verify state
Exchange code for access token
Return access token
Store session with auth context
Store authorization code with PKCE data
Redirect with auth code
POST /token (Exchange auth code + code_verifier)
Validate auth code & client
Verify PKCE: SHA256(code_verifier) == code_challenge
Create MCP access token
Return access token
POST /mcp (with Bearer token)
Validate access token
Get auth context
Return MCP response
Provide tool results
Display results
User
Claude.ai
MCP Client
Cloudflare Worker
KV Storage
GitHub OAuth

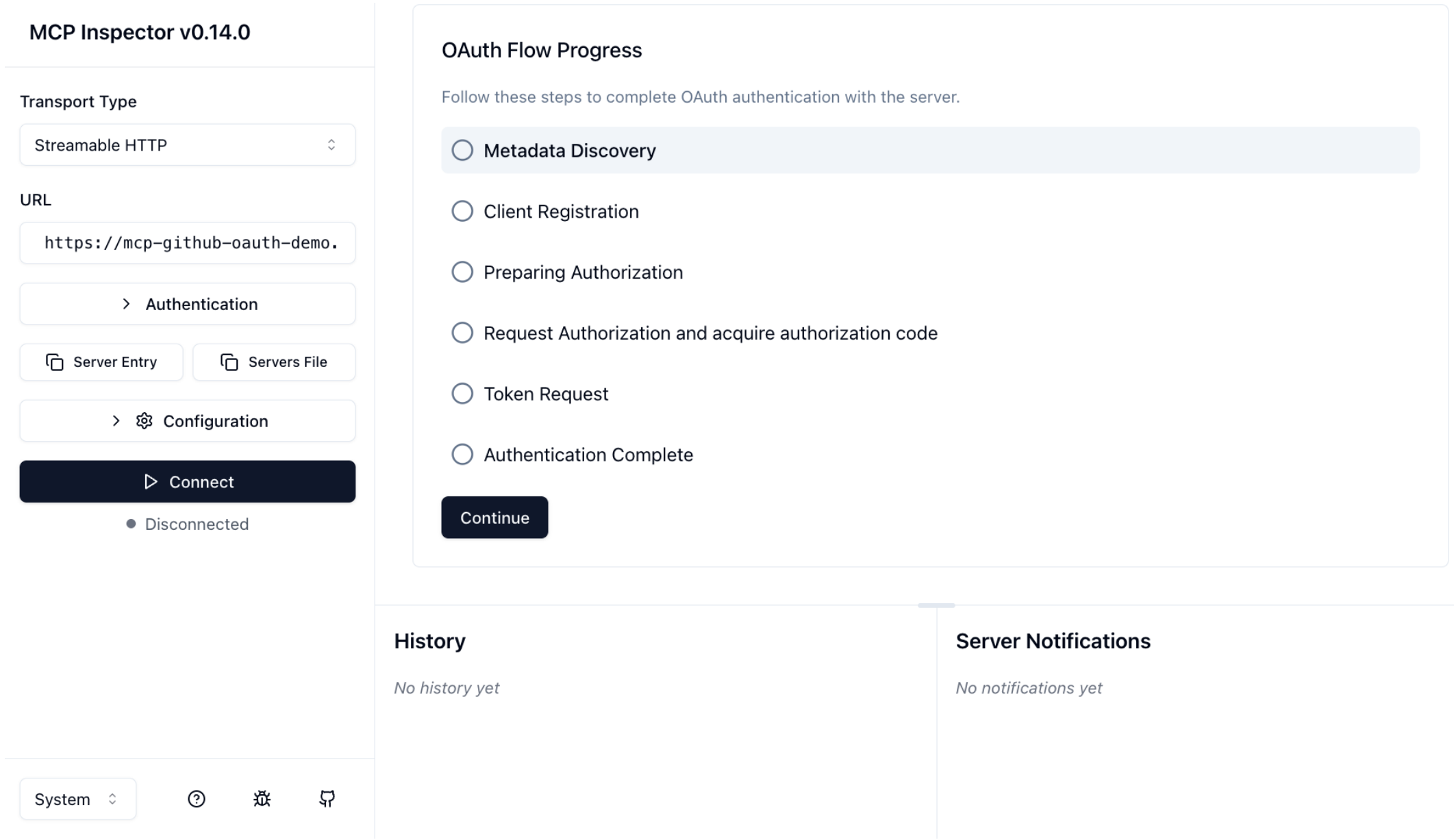
デモがうまく行かなかった用



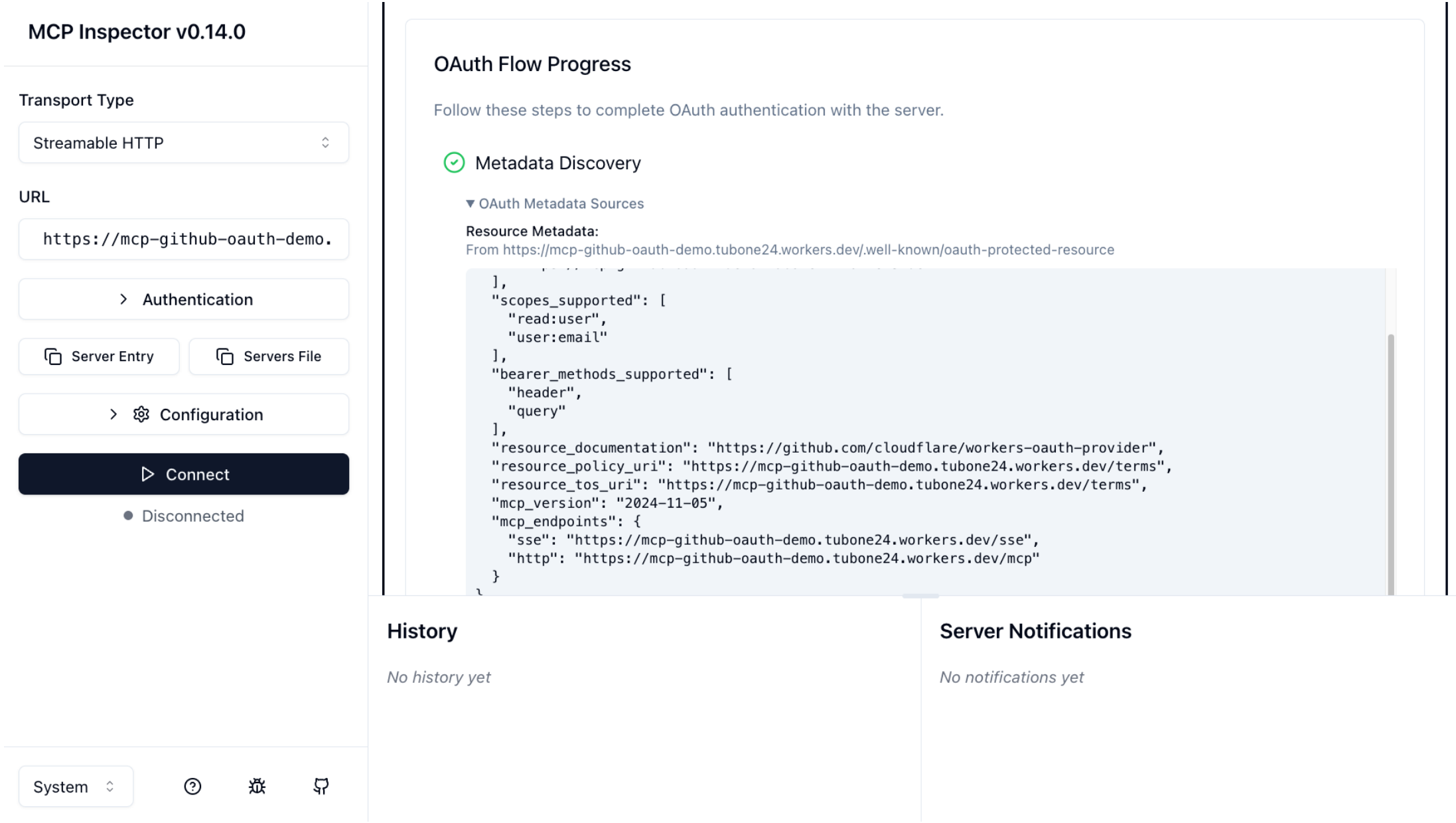
デモがうまく行かなかった用



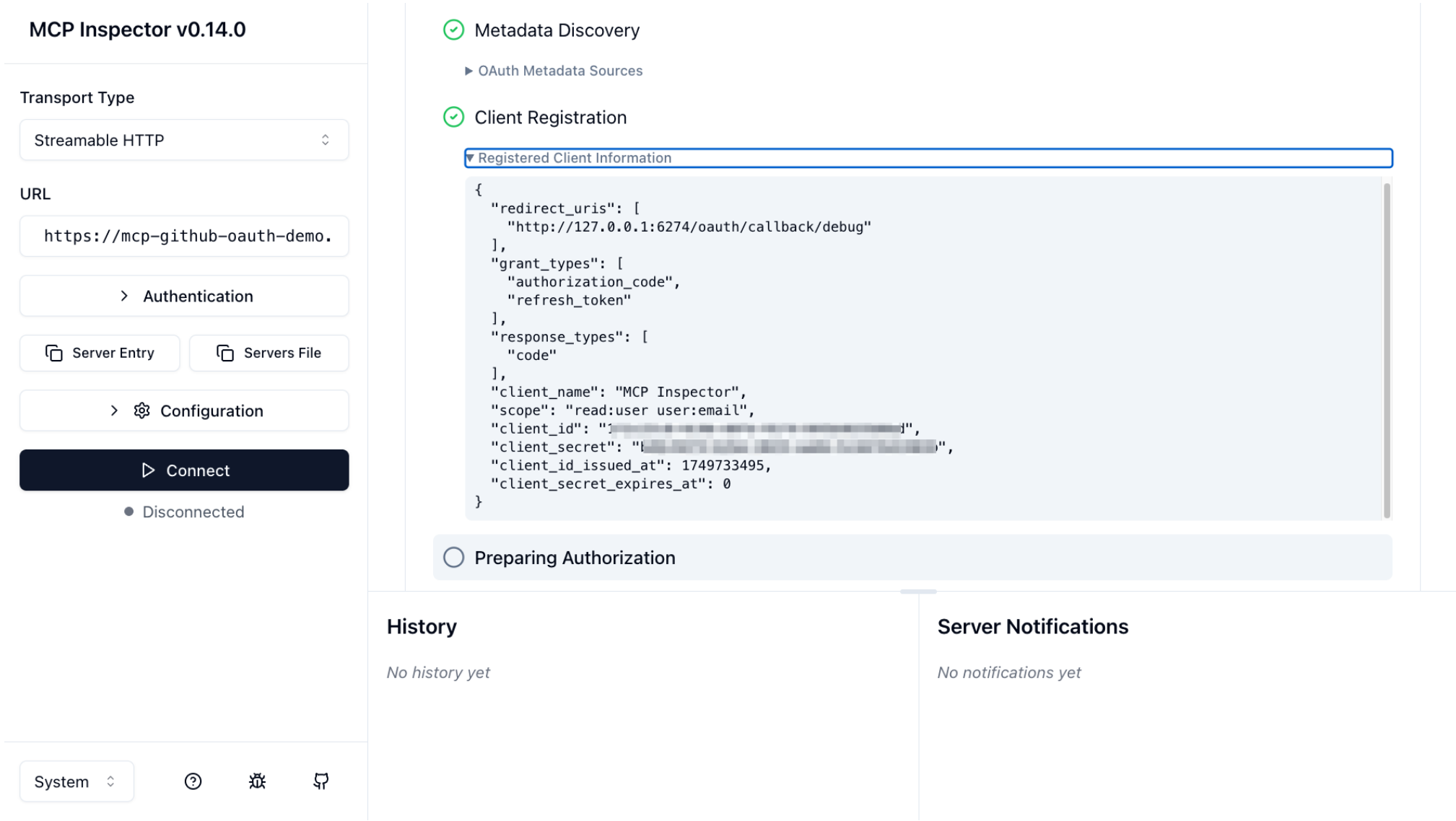
デモがうまく行かなかった用



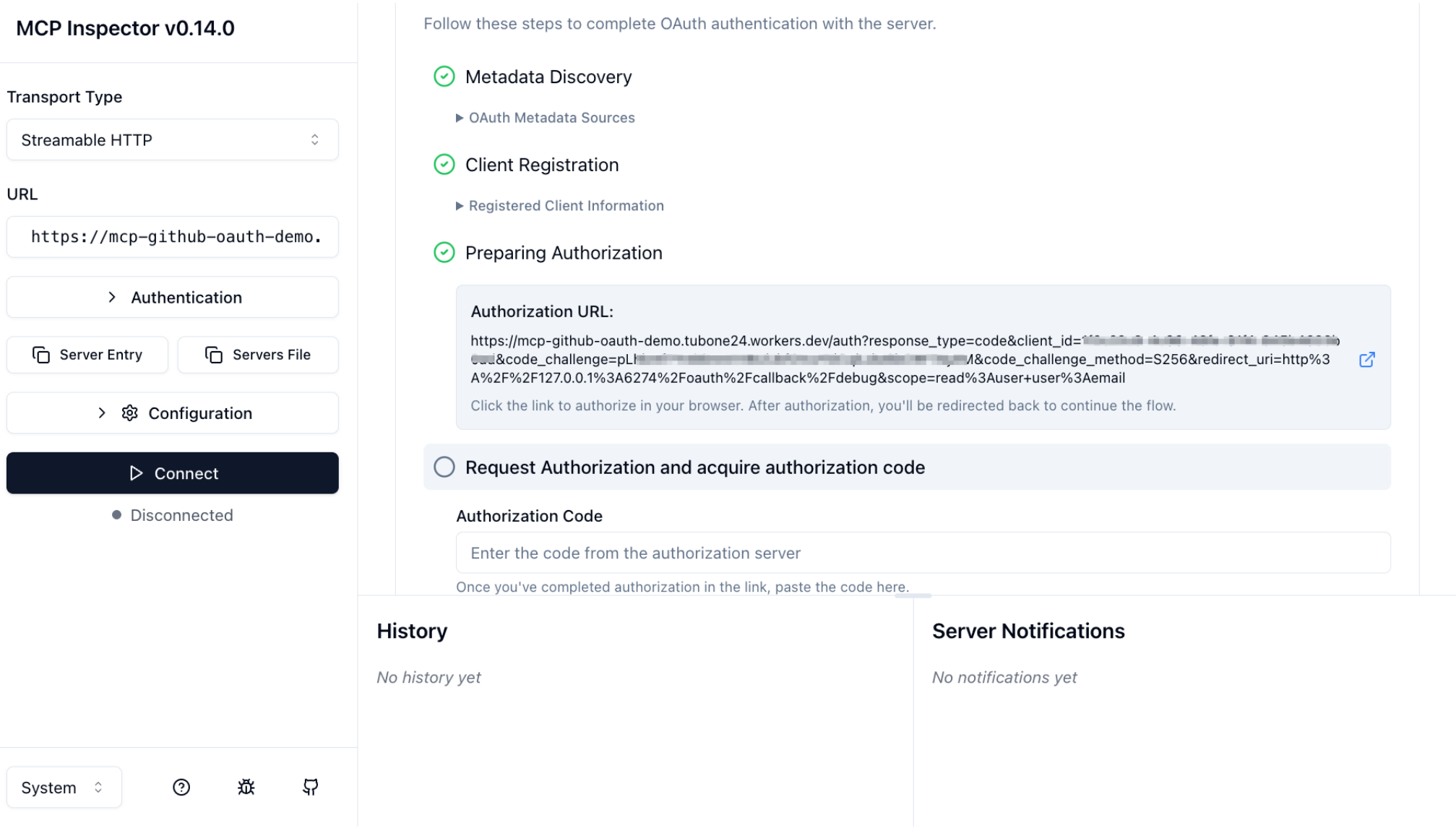
デモがうまく行かなかった用



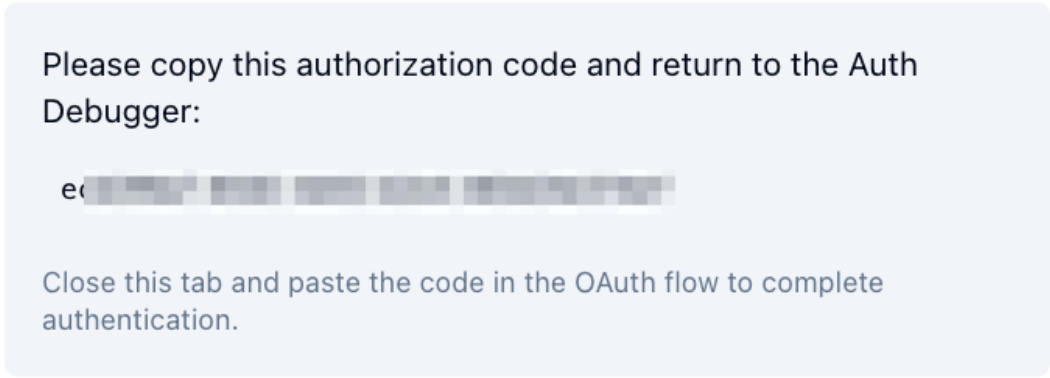
デモがうまく行かなかった用



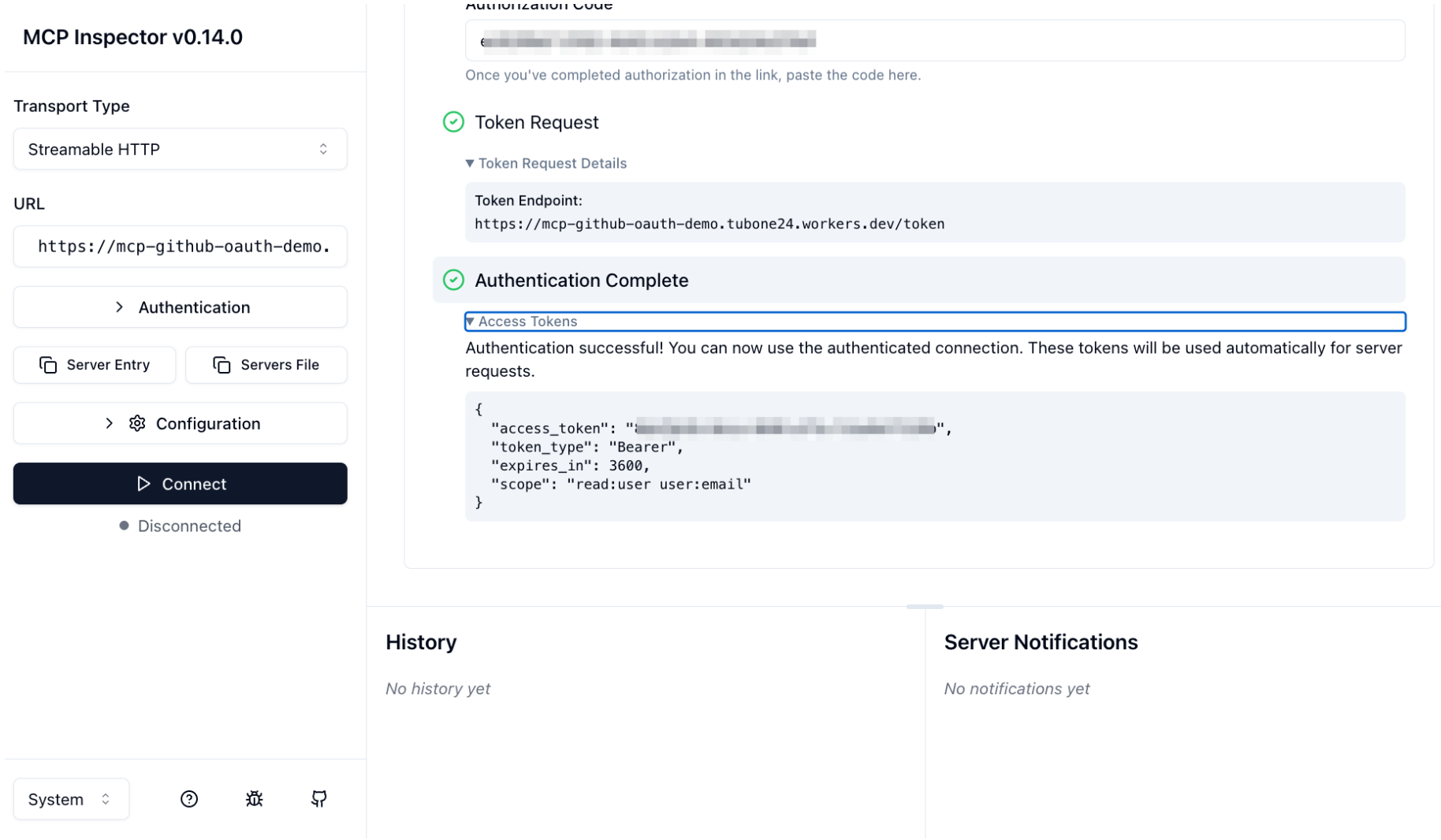
デモがうまく行かなかった用



デモがうまく行かなかった用



デモがうまく行かなかった用



デモがうまく行かなかった用