나만의 MCP 만들기

MCP 직접 구축하기

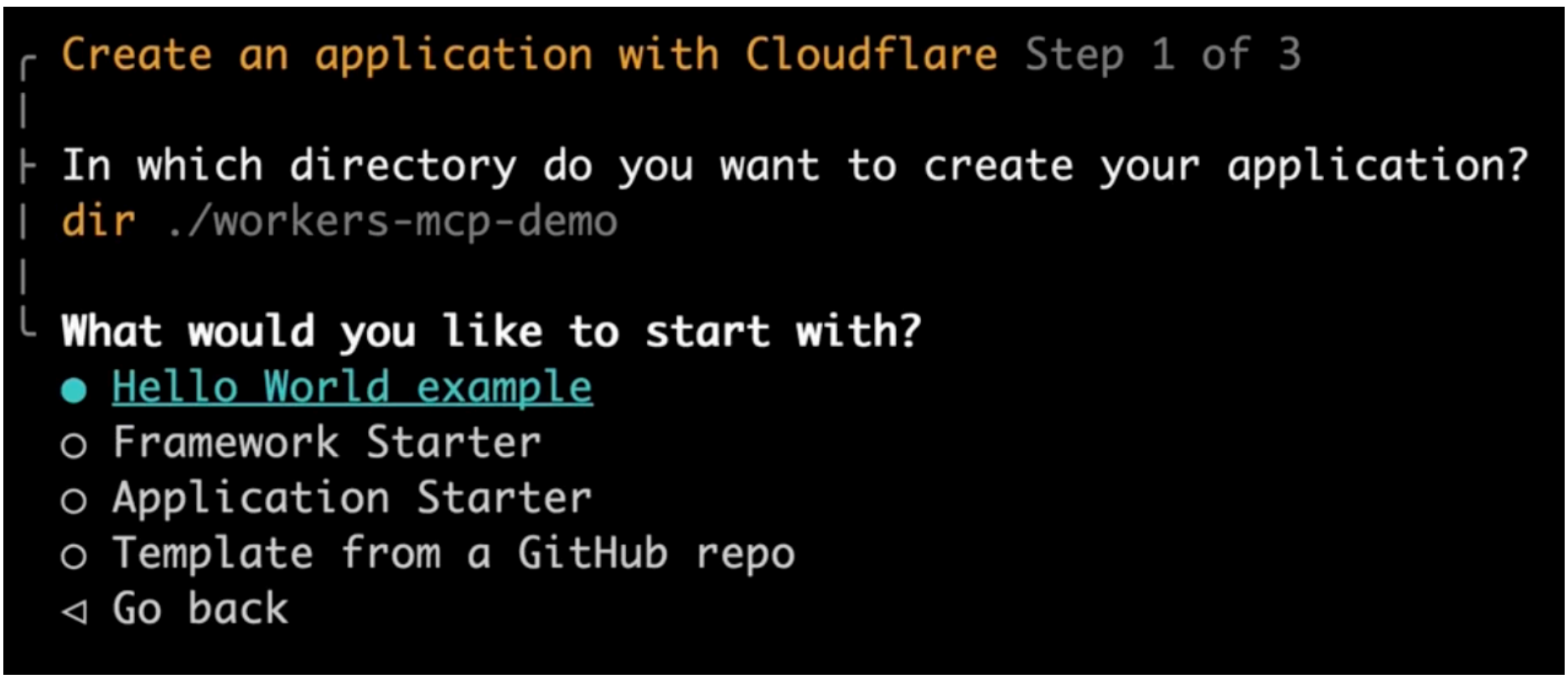
1. Cloudflare 프로젝트 설정

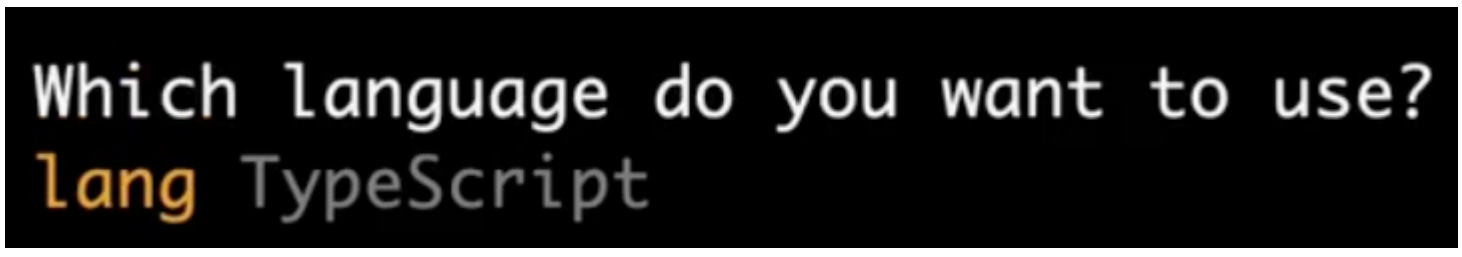
아래 명령어를 실행하여 Cloudflare 프로젝트를 생성합니다.

```
npm create cloudflare@latest
```

응용 프로그램의 이름을 입력합니다. (임의의 이름 사용 가능)

2. “Hello World” 예제 실행





Cloudflare Worker에서 “Hello World”를 실행하려면, **TypeScript**를 선택하고 **Git을 사용한 버전 관리**를 활성화합니다.

3. MCP 설정

프로젝트 폴더로 이동한 후, workers-mcp 패키지를 설치합니다.

```
npm install workers-mcp
```

그다음, 클라우드 플레어에 로그인을해야합니다.

Cloudflare 접속 방법과 API 토큰 관련 정보를 알려드리겠습니다.

Cloudflare 접속 방법

제가 사용한 Cloudflare 접속 방법은 OAuth 인증을 통한 브라우저 로그인 방식입니다:

```
npx wrangler login
```

이 명령어를 실행하면:

1. 기본 브라우저가 자동으로 열리며 Cloudflare 로그인 페이지로 이동합니다

2. Cloudflare 계정으로 로그인하면 wrangler가 로컬에 인증 정보를 저장합니다

3. 이후 wrangler 명령어를 사용할 때 저장된 인증 정보를 활용합니다

API 토큰 필요성

Cloudflare Worker를 배포하고 관리하기 위해서는 인증이 필요하며, 두 가지 인증 방식이 있습니다.

1. **OAuth 로그인 (권장)**:

- `npx wrangler login` 명령어로 브라우저 통해 로그인

- 인증 정보가 로컬에 안전하게 저장됨

- 개발 환경에서 가장 편리하고 권장되는 방식

2. **API 토큰 사용**:

- 환경 변수 `CLOUDFLARE_API_TOKEN`에 API 토큰 설정

- 스크립트나 CI/CD 파이프라인 등 자동화 환경에서 유용

- 명령어: `export CLOUDFLARE_API_TOKEN="your-token-here"`

- **개인 개발 환경**에서는 `npx wrangler login` 방식이 더 편리하고 권장됩니다
- **자동화된 환경**(CI/CD, 서버 등)에서는 API 토큰이 필수적입니다
- 두 방식 모두 가능하지만, 동시에 사용할 경우 API 토큰이 우선시됩니다

그다음, MCP 프로젝트를 설정합니다.

```
npx workers-mcp setup
```

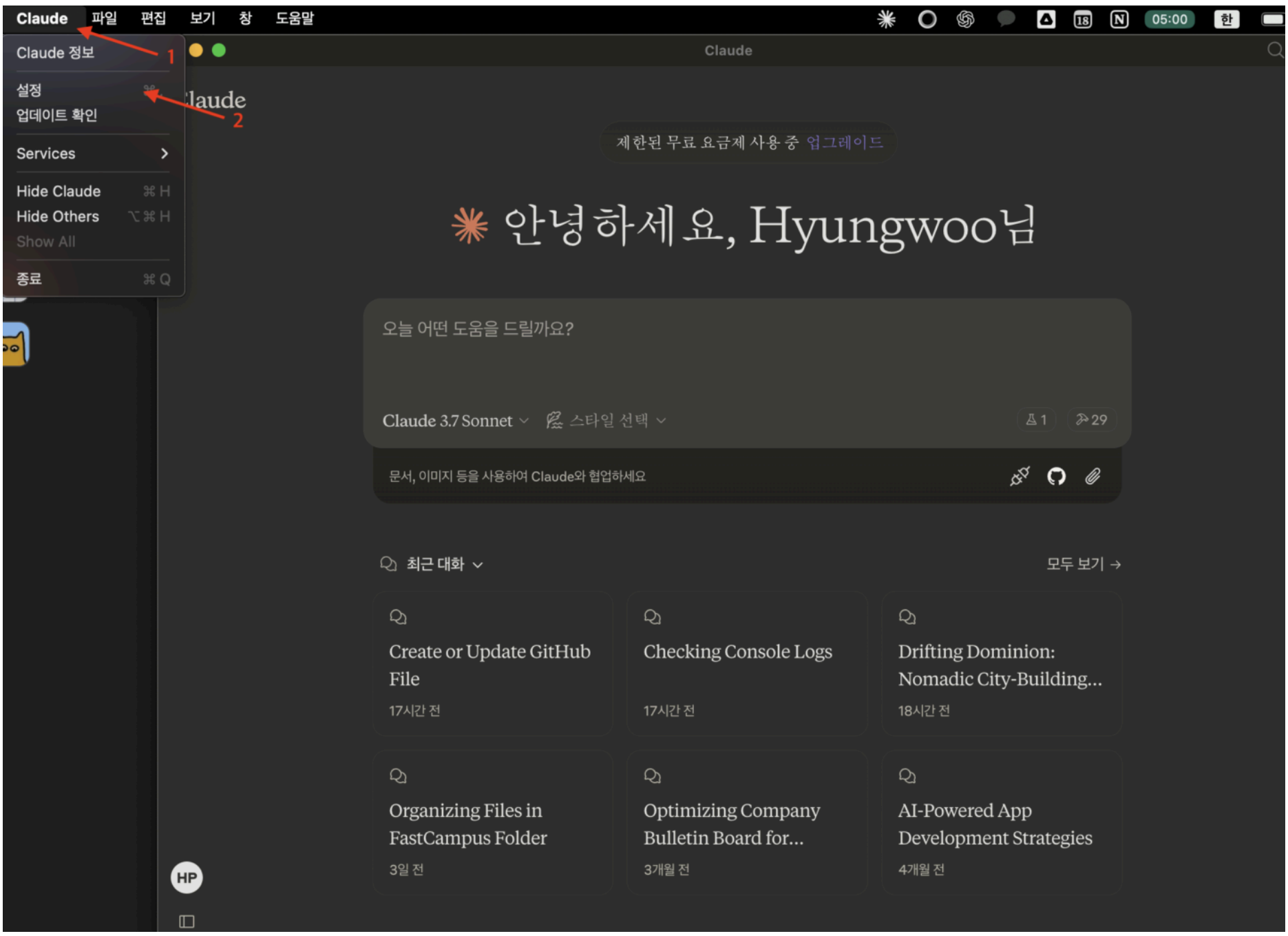
설정 과정에서 src/index.ts 파일을 대체할 것인지 묻는 질문이 나오면 **“Yes”**를 선택합니다.

MCP Worker의 이름을 입력하면 실행 명령어가 생성됩니다. 예를 들어, 다음과 같은 형식의 명령어 생성됩니다. (복사해두세요)

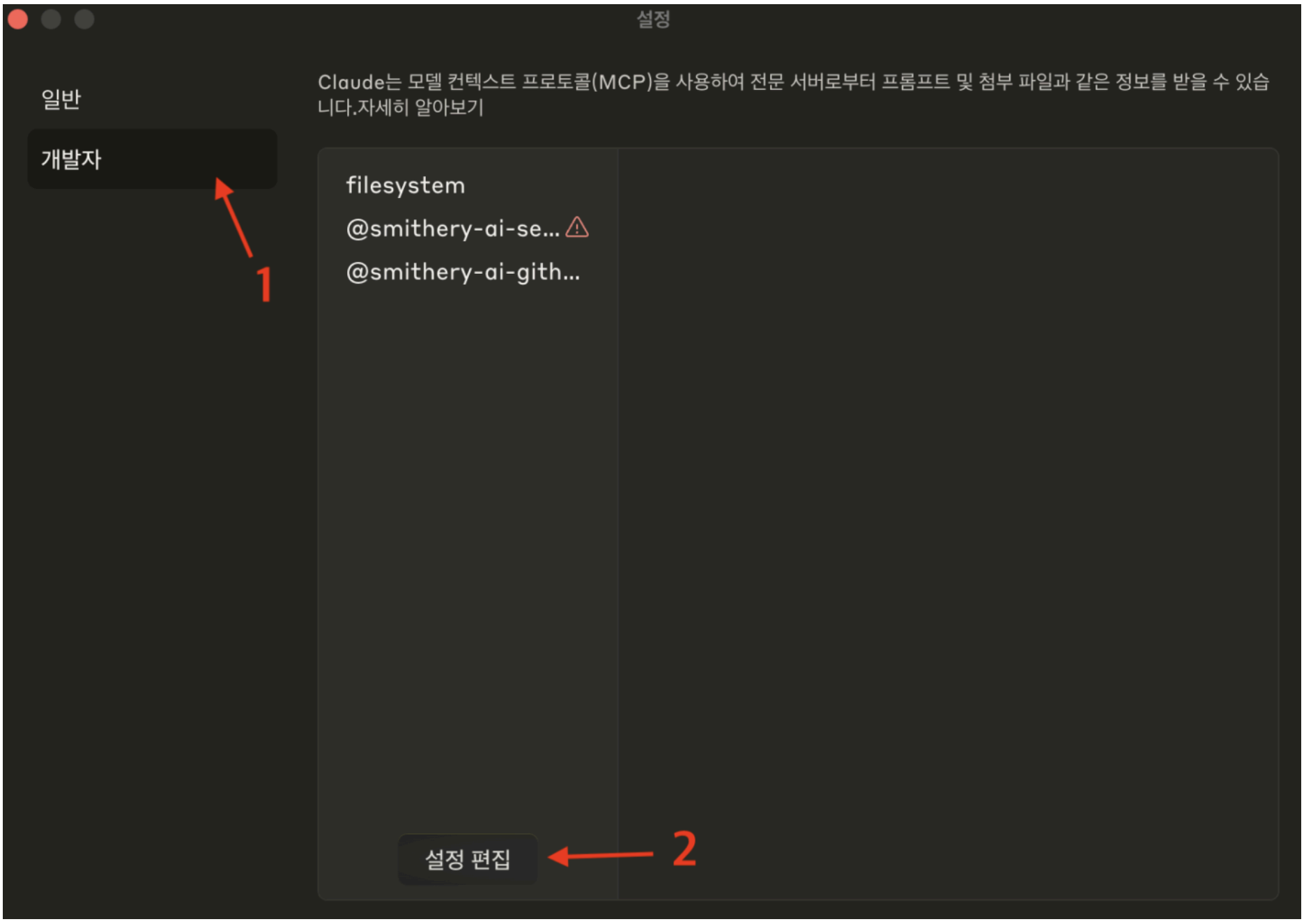
문제 1 출력 디버그 콘솔 터미널 포트 npx workers-mcp docgen src/index.ts && npx wrangler deploy

```
◇  What name should we use for this worker within Claude?
│  my-mcp
Looking for existing config in: /Users/browoo/Library/Application Support/Claude
my-mcp configured & added to Claude Desktop!
Wrote config to /Users/browoo/Library/Application Support/Claude/claude_desktop_config.json:
{
  "mcpServers": {
    "filesystem": "...",
    "@smithery-ai-server-sequential-thinking": "...",
    "@smithery-ai-github": "...",
    "my-mcp": {
      "command": "/Users/browoo/mcp/my-mcp/node_modules/.bin/workers-mcp",
      "args": [
        "run",
        "my-mcp",
        "https://my-mcp.kr2idiots.workers.dev",
        "/Users/browoo/mcp/my-mcp"
      ],
      "env": {}
    }
  }
```

이 명령어를 클로드 데스크탑에서 설정하여 mCP 테스트를 해보아햡니다.



클로드 데스크탑을 실행하고 설정 화면을 열어줍니다.



설정 편집을 누르면 "claude_desktop_config.json" 파일이 나옵니다. 이 파일을 열어줍니다.

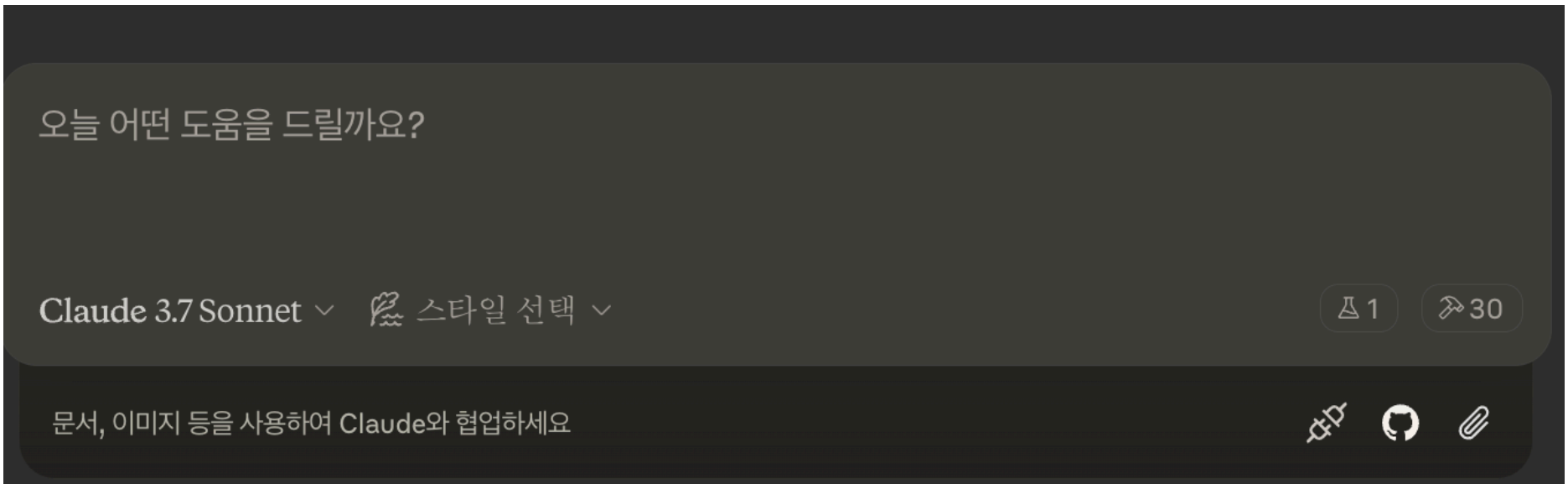
claude_desktop_config.json

claude_desktop_config

claude_desktop_config 〉 No Selection

```
    "@smithery-ai-github": {
      "command": "npx",
      "args": [
        "-y",
        "@smithery/cli@latest",
        "run",
        "@smithery-ai/github",
        "--config",
        "\"{\\\"githubPersonalAccessToken\\\":\\\"
            {\\\\\\\"method\\\\\\\":\\\\\\\"initialize\\\\\\\",\\\\\\\"params\\\\\\\":
            {\\\\\\\"protocolVersion\\\\\\\":\\\\\\\"2024-11-05\\\\\\\"
            ,\\\\\\\"capabilities\\\\\\\":
            {\\\\\\\"tools\\\\\\\":true,\\\\\\\"prompts\\\\\\\":false
            ,\\\\\\\"resources\\\\\\\":true,\\\\\\\"logging\\\\\\\":false,\\\\\\\"roots\\\\\\\":
            {\\\\\\\"listChanged\\\\\\\":false}},\\\\\\\"clientInfo\\\\\\\":
            {\\\\\\\"name\\\\\\\":\\\\\\\"cursor-vscode\\\\\\\",\\\\\\\"version\\\\\\\":\\\\\\\"1
            .0.0\\\\\\\"}},\\\\\\\"jsonrpc\\\\\\\":\\\\\\\"2
            .0\\\\\\\",\\\\\\\"id\\\\\\\":0}\\\"}\""
      ]
    },
    "my-mcp": {
      "command": "/Users/browoo/mcp/my-mcp/node_modules/.bin/workers-mcp",
      "args": [
        "run",
        "my-mcp",
        "https://my-mcp.kr2idiots.workers.dev",
        "/Users/browoo/mcp/my-mcp"
      ],
      "env": {}
    }
  }
}
```

파일에 여러분이 만든 mCP 프로젝트 내용이 자동으로 작성되어 있다면 다음 단계로 넘어가고, 작성되어 있지 않다면 이전에 복사한 내용을 붙여넣으세요. 그 다음 클로드 데스크탑을 완전히 종료한 뒤 재시작합니다.



재시작 후 우측 망치 모양의 아이콘을 눌러 아래 내용처럼 mCP 명령어가 적용 되었는지 확인합니다.

> **sayHello**
>
> A warm, friendly greeting from your new Workers MCP server.

> *서버: my-mcp*

4. MCP 기능 구현

모든 함수는 src/index.ts 파일에서 정의됩니다.

아래 예제와 같은 형식으로 **도구 설명, 입력값 및 반환값을 포함한 함수**를 작성하면 됩니다.

예제: 이미지 생성 기능

아래는 **Gemini-2.0-Flash-Exp-Image-Generation** 모델을 사용하여 이미지 생성하는 예제 코드입니다.

다운로드 방법

- 페이지 하단의 **리소스 섹션**에서 다운로드 가능합니다.

아래 내용을 **index.ts**에 복사하여 붙여넣은 후, **API 키를 설정하거나 환경에 맞게 수정해야 합니다**.

이 코드는 **즉시 실행 가능한 코드가 아니라**, 기능 구현 방법과 설정 방법을 안내하는 코드입니다.

사용자의 환경에 맞게 적절한 설정을 진행한 후 활용하시기 바랍니다.

```
import { WorkerEntrypoint } from 'cloudflare:workers';
import { ProxyToSelf } from 'workers-mcp';
import { GoogleGenerativeAI } from '@google/generative-ai';

/**
 * 이미지 생성 API 교육용 예제 코드
 *
 * 이 코드는 Cloudflare Worker를 사용해 이미지 생성 API를 구현하는 예제입니다.
 * Google의 Gemini API를 활용하여 텍스트 프롬프트를 기반으로 이미지를 생성합니다.
 *
 * 설정 방법:
 * 1. Cloudflare 계정 생성 및 Workers 활성화
 * 2. Google AI Studio에서 API 키 발급 (https://makersuite.google.com/app/apikey)
 * 3. 아래 GEMINI_API_KEY에 발급받은 키 입력
 * 4. YOUR_WORKER_DOMAIN을 자신의 Worker 도메인으로 변경
 * 5. wrangler.toml 파일 설정 (환경 변수 등)
 * 6. `npx wrangler deploy` 명령으로 배포
 */

// TypeScript 환경 변수 정의
interface Env {
    // Workers MCP에 필요한 필수 환경 변수
    SHARED_SECRET: string;
    // 선택적으로 API 키를 환경 변수로 설정 가능
    // GEMINI_API_KEY: string;
}

// API 요청 형식 정의
interface ImageGenerationRequest {
    prompt?: string;
    direct?: boolean;
}

// API 키 설정
// 실제 구현 시에는 환경 변수로 설정하는 것이 안전합니다!
// const GEMINI_API_KEY = env.GEMINI_API_KEY; 형태로 사용하세요.
const GEMINI_API_KEY = 'YOUR_GEMINI_API_KEY_HERE';

// Worker 도메인 설정
// 자신의 Cloudflare Worker 도메인으로 변경하세요
```

```
const WORKER_DOMAIN = 'YOUR_WORKER_DOMAIN.workers.dev';

export default class ImageGenerationWorker extends WorkerEntrypoint<Env> {
    /**
     * 이미지 생성 URL을 반환하는 함수
     * @param prompt {string} 이미지 생성 프롬프트
     * @return {string} 이미지 생성 URL
     */
    getImageGenerationUrl(prompt: string): string {
        const encodedPrompt = encodeURIComponent(prompt);
        return `https://${WORKER_DOMAIN}/generateimage?prompt=${encodedPrompt}&direct=true`;
    }

    // 캐시 저장을 위한 프로퍼티 선언
    // 참고: Worker는 요청 간에 상태를 유지하지 않으므로
    // 실제 프로덕션 환경에서는 KV나 R2 스토리지 사용을 권장합니다
    private cache?: Map<string, string>;

    /**
     * Worker의 요청 처리 메인 함수
     */
    async fetch(request: Request): Promise<Response> {
        // MCP 프록시 요청인지 확인
        const url = new URL(request.url);

        // CORS 프리플라이트 요청 처리
        if (request.method === 'OPTIONS') {
            return new Response(null, {
                headers: {
                    'Access-Control-Allow-Origin': '*',
                    'Access-Control-Allow-Methods': 'GET, POST, OPTIONS',
                    'Access-Control-Allow-Headers': 'Content-Type',
                },
            });
        }

        // 모든 API 요청을 처리하는 단일 엔드포인트
        if (url.pathname === '/generateimage' || url.pathname === '/') {
            let prompt: string | null = null;
            let directImage = false; // 이미지 직접 반환 여부

            // GET 요청 처리 (URL 쿼리 파라미터)
            if (request.method === 'GET') {
                prompt = url.searchParams.get('prompt');
                // direct 파라미터가 있으면 이미지를 직접 반환
                directImage = url.searchParams.get('direct') === 'true';
            }
            // POST 요청 처리 (JSON 본문)
            else if (request.method === 'POST') {
                try {
                    // 요청 본문이 JSON인지 확인
                    const contentType = request.headers.get('content-type');
                    if (contentType && contentType.includes('application/json')) {
                        const json = (await request.json()) as ImageGenerationRequest;
                        if (json.prompt) prompt = json.prompt;
                        if (json.direct) directImage = json.direct;
                    } else {
                        // URL 인코딩된 폼 데이터 처리
```

```
            const formData = await request.formData();
            prompt = (formData.get('prompt') as string) || null;
            directImage = formData.get('direct') === 'true';
        }
    } catch (error) {
        return new Response(
            JSON.stringify({
                success: false,
                error: 'Invalid request body format',
            }),
            {
                status: 400,
                headers: {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*',
                },
            }
        );
    }
}

if (!prompt) {
    // 메인 페이지 표시 (간단한 안내 정보)
    if ((url.pathname === '/' || url.pathname === '/generateimage') && request.method === 'GET') {
        return new Response(
            `
Image Generation API 예제

사용법:
- 이미지 직접 보기: /generateimage?prompt=이미지_설명&direct=true
- JSON 응답 받기: /generateimage?prompt=이미지_설명
- URL만 받기: /generateurl?prompt=이미지_설명

예시:
https://${WORKER_DOMAIN}/generateimage?prompt=cute%20puppy&direct=true
https://${WORKER_DOMAIN}/generateurl?prompt=cute%20puppy
`,
            {
                headers: { 'Content-Type': 'text/plain' },
            }
        );
    } else {
        return new Response(
            JSON.stringify({
                success: false,
                error: 'Missing prompt parameter',
            }),
            {
                status: 400,
                headers: {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*',
                },
            }
        );
    }
}
```

```
        // direct=true이면 이미지를 직접 생성하고 반환
        if (directImage) {
          // 이미지 직접 생성 및 반환
          try {
            // Gemini API 초기화
            // API 키 확인
            if (GEMINI_API_KEY === 'YOUR_GEMINI_API_KEY_HERE') {
              return new Response(
                '설정 오류: Gemini API 키가 설정되지 않았습니다. example.ts 파일에서 GEMINI_API_KEY를 설정하거나 환경 변수로 제공하세요.',
                {
                  status: 500,
                  headers: { 'Content-Type': 'text/plain' },
                }
              );
            }

            const genAI = new GoogleGenerativeAI(GEMINI_API_KEY);
const model = genAI.getGenerativeModel({
  model: 'gemini-2.0-flash-exp-image-generation',
  generationConfig: {
    // @ts-ignore - responseModalities는 최신 버전에서 사용 가능
    responseModalities: ['Text', 'Image'],
  },
});

// 프롬프트로 이미지 생성 요청
const response = await model.generateContent(prompt);
const candidates = response.response?.candidates || [];

// 이미지 데이터 추출
let imageData: string | null = null;
if (candidates.length > 0 && candidates[0]?.content?.parts) {
  for (const part of candidates[0].content.parts) {
    if (part.inlineData && part.inlineData.data) {
      imageData = part.inlineData.data;
      break;
    }
  }
}

if (imageData) {
  // 이미지 데이터를 바이너리로 변환하여 반환
  const binaryData = atob(imageData);
  const buffer = new Uint8Array(binaryData.length);
  for (let i = 0; i < binaryData.length; i++) {
    buffer[i] = binaryData.charCodeAt(i);
  }

  return new Response(buffer, {
    headers: {
      'Content-Type': 'image/png',
      'Content-Disposition': 'inline; filename="generated-image.png"',
      'Cache-Control': 'public, max-age=86400',
    },
  });
} else {
  return new Response('이미지를 생성할 수 없습니다.', {
```

```
                status: 500,
                headers: { 'Content-Type': 'text/plain' },
            });
        }
    } catch (error: unknown) {
        const errorMessage = error instanceof Error ? error.message : String(error);
        return new Response(`이미지 생성 실패: ${errorMessage}`, {
            status: 500,
            headers: { 'Content-Type': 'text/plain' },
        });
    }
} else {
    // 이미지 URL 반환
    const imageUrl = this.getImageGenerationUrl(prompt);
    return new Response(
        JSON.stringify({
            success: true,
            url: imageUrl,
        }),
        {
            headers: {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
            },
        }
    );
    }
}

// URL 생성 엔드포인트
if (url.pathname === '/generateurl') {
    let prompt: string | null = null;

    // GET 요청 처리
    if (request.method === 'GET') {
        prompt = url.searchParams.get('prompt');
    }
    // POST 요청 처리
    else if (request.method === 'POST') {
        try {
    // 요청 본문이 JSON인지 확인
    const contentType = request.headers.get('content-type');
    if (contentType && contentType.includes('application/json')) {
        const json = (await request.json()) as ImageGenerationRequest;
        if (json.prompt) prompt = json.prompt;
    } else {
        // URL 인코딩된 폼 데이터 처리
        const formData = await request.formData();
        prompt = (formData.get('prompt') as string) || null;
    }
} catch (error) {
    return new Response(
        JSON.stringify({
            success: false,
            error: 'Invalid request body format',
        }),
        {
            status: 400,
```

```
                headers: {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*',
                },
            }
        );
    }
}

if (!prompt) {
    return new Response(
        JSON.stringify({
            success: false,
            error: 'Missing prompt parameter',
        }),
        {
            status: 400,
            headers: {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
            },
        }
    );
}

// URL 생성 및 반환
const imageUrl = this.getImageGenerationUrl(prompt);
return new Response(
    JSON.stringify({
        success: true,
        url: imageUrl,
    }),
    {
        headers: {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*',
        },
    }
);
}

// 이미지 직접 제공 엔드포인트 - 이 기능은 생략했습니다
// 실제 구현 시에는 Cloudflare KV나 R2를 사용하여 이미지 데이터를 저장하는 것이 좋습니다

// 그 외 모든 요청은 MCP를 통해 처리
return new ProxyToSelf(this).fetch(request);
}
}

/**
 * 설정 방법 가이드
 *
 * 1. API 키 설정:
 *    - Google AI Studio(https://makersuite.google.com/app/apikey)에서 API 키 발급
 *    - 발급받은 키를 GEMINI_API_KEY 상수에 입력하거나
 *    - 환경 변수로 설정 (보안상 권장, wrangler.toml에 설정)
 *
 * 2. Worker 도메인 설정:
```

```
*    - WORKER_DOMAIN 상수를 자신의 Cloudflare Worker 도메인으로 변경
*    - 예: 'my-image-generator.username.workers.dev'
*
* 3. wrangler.toml 설정 예시:
*    ```
*    name = "my-image-generator"
*    main = "src/index.ts"
*    compatibility_date = "2023-10-02"
*
*    [vars]
*    GEMINI_API_KEY = "YOUR_API_KEY_HERE"
*
*    # Workers MCP 설정
*    [mcp]
*    SHARED_SECRET = "generate_a_random_string_here"
*    ```
*
* 4. 배포:
*    - `npm install` 명령으로 의존성 설치
*    - `npx wrangler deploy` 명령으로 배포
*
* 5. 사용 방법:
*    - 이미지 직접 보기: /generateimage?prompt=이미지_설명&direct=true
*    - JSON 응답 받기: /generateimage?prompt=이미지_설명
*    - URL만 받기: /generateurl?prompt=이미지_설명
*/
```

---

5. MCP 배포 및 적용

새로운 기능을 추가하거나 업데이트한 후에는 다음과 같은 과정으로 배포를 진행합니다.

```
npm run deploy
npx workers-mcp setup
```

그 후, **Cursor를 재시작하면** 새롭게 추가한 MCP 도구가 정상적으로 적용된 것을 확인할 수 있습니다.

MCP 직접 구축하기 예제.ts