Code Mode Example

- How it works
  - DynamicWorkerExecutor
  - createCodeTool

import { TypeScriptExample } from "~/components";

This example shows how to use the `@cloudflare/codemode` library with the Agents SDK to build an agent where the LLM writes code to orchestrate tool calls, instead of calling them one at a time.

This approach, called Code Mode, reduces tokens spent by up to 80%, returns better results, and avoids bloating the context window.

Deploy to Workers

This example shows you how to:

- Define tools as plain functions with Zod schemas
- Use `createCodeTool` to expose your tools to the LLM as a single "write code" tool
- Use `DynamicWorkerExecutor` to safely run LLM-generated code
- Wire it all together with `AIChatAgent` to handle chat over WebSockets

How it works

The agent uses three components from the `@cloudflare/codemode` library and the Agents SDK:

- **`AIChatAgent`** (**`@cloudflare/ai-chat`**), your agent's base class. Handles chat over WebSockets, persists messages, and calls the LLM.
- **`createCodeTool`** (**`@cloudflare/codemode/ai`**), wraps your tools into a single `codemode` tool that accepts `{ code: string }`.
- **`DynamicWorkerExecutor`** (**`@cloudflare/codemode`**), runs the LLM-generated code in an isolated Dynamic Worker.

The flow:

1. User sends a message over WebSocket.
2. `AIChatAgent` passes it to the LLM with a single tool available: `codemode`.
3. The LLM writes JavaScript, for example `const projects = await codemode.listProjects()`, instead of making individual tool calls.
4. `DynamicWorkerExecutor` spins up an isolated Worker and runs the code. Inside the sandbox, `codemode.listProjects()` calls your real `listProjects` implementation.
5. The result, any console output, and errors are returned to the LLM.
6. The LLM uses the result to respond to the user, or writes more code if it needs to.

`DynamicWorkerExecutor`

`DynamicWorkerExecutor` is part of the `@cloudflare/codemode` library. When the LLM writes code that orchestrates your tools, that code needs to run somewhere safe. `DynamicWorkerExecutor` spins up an isolated Dynamic Worker for each execution using the Worker Loader binding. Inside the sandbox:

- A `codemode` proxy object routes calls like `codemode.createTask(...)` back to your real tool implementations over Workers RPC
- Setting `globalOutbound` to `null` blocks `fetch()`, so the code can only reach the outside world through your tools
- `console.log` output is captured and returned alongside the result
- Each execution gets its own Worker instance with a 30-second timeout

```
import { DynamicWorkerExecutor } from "@cloudflare/codemode";

const executor = new DynamicWorkerExecutor({
  loader: env.LOADER, // WorkerLoader binding from wrangler.jsonc
  timeout: 30000, // default: 30s
  globalOutbound: null, // null = fetch blocked
});
```

`createCodeTool`

`createCodeTool` is part of `@cloudflare/codemode`. It takes your tools and an executor, and returns a single AI SDK `tool()`. It:

- Generates TypeScript type declarations from your tools' Zod schemas, so the LLM knows what is available and what the argument shapes look like.
- Puts those types in the tool's description, so the LLM sees a single tool with parameter `{ code: string }` and a description that includes the full typed API surface.
- On execution, normalizes the LLM's code (strips markdown fences, wraps bare statements in async functions, auto-returns the last expression), then passes it to the executor.

```
import { createCodeTool } from "@cloudflare/codemode/ai";

const codemode = createCodeTool({
  tools: myTools, // Record<string, tool()> with Zod schemas
  executor, // DynamicWorkerExecutor
});

// The LLM sees: one tool called "codemode" with input { code: string }
// The description includes TypeScript types for all your tools
```

The LLM writes an async arrow function. `createCodeTool` normalizes it and hands it to the executor. The executor builds a Worker module with a `codemode` proxy, runs the code, and returns `{ code, result, logs }`.

□□□ #2
□ □□□ □□□ 2026-11-01 02:04:30 UTC
□ □□□ □□□ 2026-11-01 02:04:32 UTC