LIVE DEMO | MCP | POWER AUTOMATE

Claude builds, runs and debugs your Power Automate flows

An MCP server. Ten tools. One file. It tells you WHY a flow failed, not just WHAT.

```
> It failed. Why?
```

`explain_run()`
Compute_kpis divided by 0 staff from Dataverse.

Elliot Margot
Microsoft MVP, M365 Copilot & Copilot Studio | e-margot.ch

Elliot Margot

Team Lead Jumpstart, Copilot & Agents at Witivio, Microsoft Partner

- **Microsoft MVP** M365 Copilot & Copilot Studio
- **AI Specialist** Power Platform Solutions Architect
- **Lausanne, Switzerland** German, English, French

60+ AI use cases designed across 17 industries

15+ production AI architectures deployed

A flow failed. Now count the clicks.

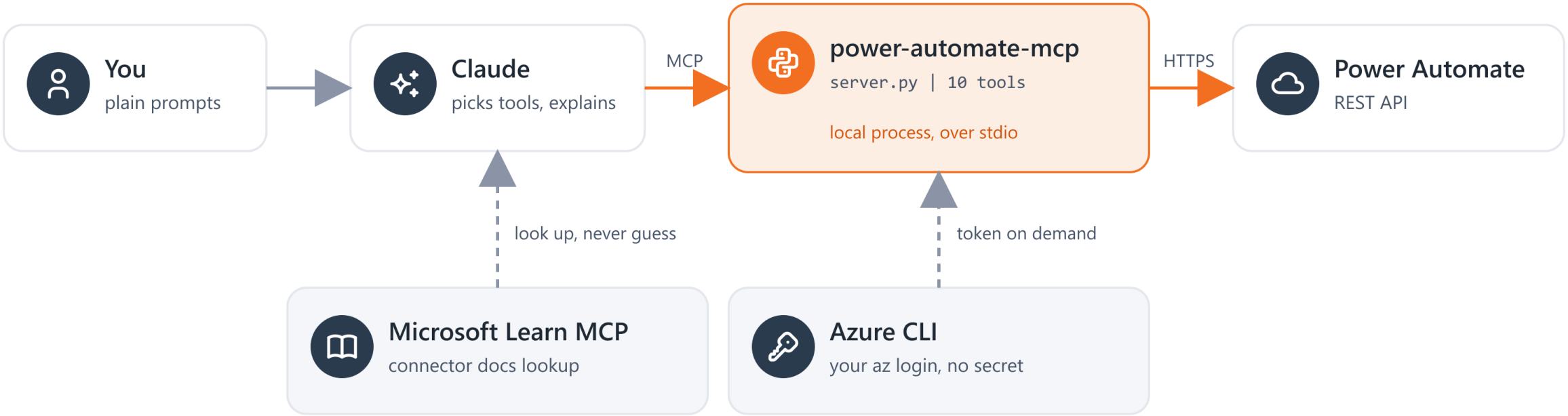
1. Open the flow
2. Run history
3. Open the failed run
4. Expand the red action
5. Read the error
6. Walk upstream to find the value

And the API is worse. The run says "An action failed". For connector failures the action record carries `error: null`: the real message hides behind a short-lived signed link.

The API tells you WHAT broke. A good tool tells you WHY, because it follows the links the portal follows and puts the upstream values next to the error.

An MCP server between Claude and Power Automate

ONE PROMPT, END TO END



No app registration, no secret: it borrows your az login

Runs locally over stdio: nothing to host, no token on the wire

Pinned to one tenant with PA_TENANT_ID: fails loudly elsewhere

Wrapping the API is commodity. The value is above it.

1 Auth

one az CLI call, plus caching

`~45 lines` Claude writes it in one prompt

2 Transport

retry 401 / 429 / 5xx, follow nextLink

`~50 lines` Claude writes it in one prompt

3 Shaping

60 API fields down to the few that matter

`~160 lines` **You. This is the work.**

4 Docstrings

what the API will never tell the model

`~200 lines` **You. This is the moat.**

Everything you get wrong twice belongs in a docstring. The docstrings are the prompt the model reads before every call. The code is regenerable; that knowledge is the asset.

CAPABILITIES

Ten tools, three jobs, plus the docs

Author

`list_flows`
what exists, newest first

`get_flow`
the full definition, Code view

`create_flow`
validated before it is sent

`update_flow_definition`
keeps bindings, snapshots the old

`bind_connection`
finds and binds a connection

Operate

`run_flow`
trigger it now

`list_runs`
status of recent runs

Diagnose

`explain_run`
which action, what error, why

`compare_runs`
diff a failed run against a good one

`analyze_flow_health`
flaky or broken, and where

Look up

`microsoft_docs_search`
Microsoft Learn MCP

`microsoft_docs_fetch`
connector reference pages

THE TOOL THAT JUSTIFIES THE EXERCISE

explain_run: from WHAT to WHY in one call

WHAT explain_run DOES THAT THE API DOES NOT

1 Read the run actions

`GET /runs/{id}/actions`

`Compute_kpis: Failed`

The API says:
"An action failed."
A naive wrapper stops here.

2 Follow the hidden link

connector errors are not
on the action record:
`error: null`
`outputsLink -> signed blob`

The tool fetches the real text.

3 Pair with upstream outputs

the failed action, next to
what ran before it:
`Get_store (Dataverse) ->`
`{ numberofemployees: 0 }`

The cause is one step up.

4 Claude answers WHY

"Compute_kpis divides
revenue by staff, and
Lausanne had 0 staff
on a closed Sunday."

1 call, not ~12 clicks.

Expression errors come back inline; connector errors hide behind the link. The tool handles both.

You will see exactly this in the demo. One tool call instead of all those clicks, and an answer that names the cause, not just the symptom.

The traps the model never hits twice

error: null

Connector errors live behind a signed link. explain_run follows it.

Two magic parameters

$connections and $authentication, or a 400 that wrongly blames the trigger.

Bind every connector at once

A flow cannot start, or even save, with one connector unbound.

Connections are OAuth

No API can create one. Portal once, bind forever: the tool hands you the link.

Designer-safe definitions

A bare Button schema crashed the new designer. create_flow now fixes it.

No secrets in definitions

They are stored in plain text. Use environment variables or Key Vault.

56 tools vs 10, on the same failed run

Question	Microsoft flowagent (56 tools)	power-automate-mcp (10 tools)
Which action failed?	Yes	Yes
Error text on a connector failure	`"message": ""`	`404 NotFound: LocationLookupFailed ...`
The failing action's inputs	Not returned	Resolved from the content link
Upstream value that caused it	Unreachable	**Returned next to the error**
Diff against a working run	No such tool	`compare_runs`

Microsoft's plugin is broader: use it. **Build your own for the one question that keeps costing you hours.**

Four steps. One rule.

1. **`git clone`**
   github.com/OwnOptic/power-automate-mcp

2. **`pip install`**
   -r requirements.txt (Python 3.10+)

3. **`az login`**
   your normal Azure CLI sign-in

4. **`register`**
   add server.py to your MCP client config

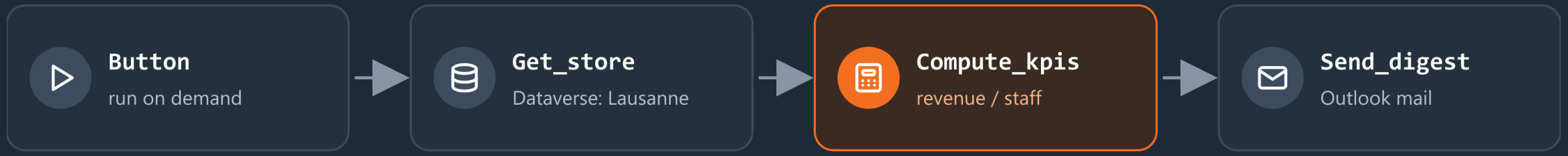
It acts as you.

- Create, edit and run are live, with no confirmation step.
- Point it at a test environment, pinned with PA_TENANT_ID.
- No delete tool in the core server, on purpose.
- For production: split read tools and write tools into two servers.

Or give it to your Vibe Coding favourite platform and let it do it for you

WHAT YOU ARE ABOUT TO SEE

A Dataverse sales digest that breaks on Sunday



1 Build + connect

Claude creates the flow. I create the Dataverse connection live; Claude binds both and starts it.

2 Run

Dataverse says Lausanne was closed: 0 staff. The run fails, and the API only says "An action failed".

3 Debug

Claude explains why, fixes the logic, reruns it, and the digest lands in the inbox.

I only type prompts. Claude does every step and explains it. About 10 minutes, live, no safety net.

Let's build it.

1. **Auth and transport are commodity.**
   az login and one request helper. Do not spend your talent there.

2. **Shape the payload for the model.**
   Upstream outputs next to the error turn WHAT into WHY.

3. **Every bug you hit twice belongs in a docstring.**
   That is the part you actually own.

</> `github.com/OwnOptic/power-automate-mcp`