Vol. 4 · for builders & AI consultants

Building *on Claude*

free to share!

A practical guide to the Claude API: the right model, prompts that hold up in production, tools, grounding, MCP, agents, evals, safety, and handing it over to clients.

MIAN SHAFIQ · SHAFIQ.PK



I build these for clients

Hello, builders.

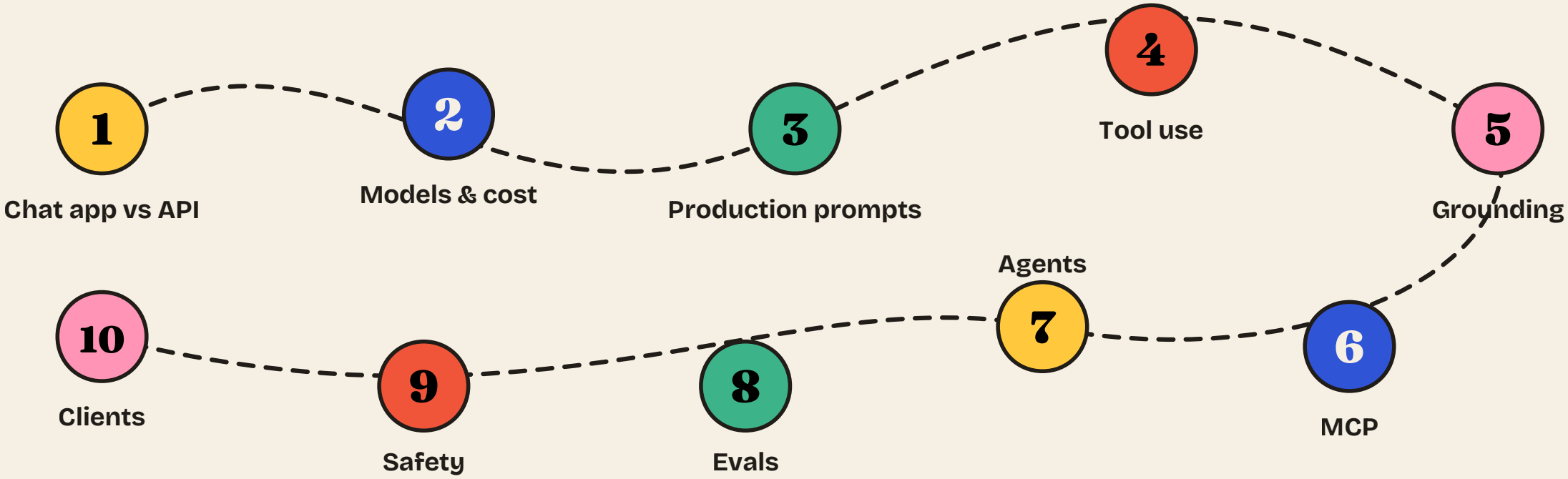
Volumes 1 to 3 were about using Claude. This one is about **building with it**: products, client projects and automations that run while nobody is watching.

The goal: something that's right, affordable and safe, and that you can prove is all three.

One idea per slide. About 20 minutes. Some Python helps.

The route

Ten stops. 1–5 get it working; 6–10 make it last.



1

the question I hear most...

"Can't we just use the chat app?"

For your own work, yes. For your customers, your data and your rules, inside your product, you need the API.

Chat app vs API

Four ways to build

From the most control to the least plumbing.

MESSAGES API

One call at a time

You send messages, you get a reply. Your code decides everything else.

AGENT SDK

An agent in your code

The loop behind Claude Code, running on your own servers.

MANAGED AGENTS

Agents, hosted

Anthropic runs the agent, its sandbox and sessions. In beta.

MCP SERVER

A plug for your system

Let every Claude use your data and actions. Stop 6.

start with the Messages API. Most client projects never need more.

Your first call

That's the whole API: a **model**, a **system prompt** with your rules, and the **messages** so far.

There's no memory between calls. Your app sends the history each time, so you decide what Claude sees.

first_call.py

```python
import anthropic
client = anthropic.Anthropic()  # reads ANTHROPIC_API_KEY

msg = client.messages.create(
    model="claude-sonnet-5-5",
    max_tokens=1024,
    system="You answer for Acme's help desk.",
    messages=[{"role": "user",
               "content": "Where's my order?"}],
)
print(msg.content[0].text)
```

2

Four models. Pick per step.

US dollars per million tokens, input / output. A million tokens is roughly 700,000 words.

HAIKU 5.5

$0.10 / $0.50

Sort, extract, route. Fast and cheap.

SONNET 5.5

$2 / $10

The everyday workhorse.

OPUS 5.5

$4 / $20

Hard reasoning, agents, code.

FABLE 5.2

$8 / $40

The top tier, for the toughest work.

Models & cost

Haiku 5.5 prices rise for prompts over 100,000 tokens. Prices from platform.claude.com, November 2026. Check before you quote a client.

from identity-verification work...

Open-source checks ran first. A paid model only saw the documents they couldn't decide.

Most documents never reached the expensive step, and the bill followed. Cheap first, clever second.

Three levers on the bill

LEVER 1

Right model per step

Haiku sorts the request, Sonnet answers it, Opus only for the hard cases.

LEVER 2

Prompt caching

Reuse a long fixed prefix. On Sonnet and Opus 5.5, cached reads cost 5% of normal input.

LEVER 3

Batch

Work that can wait (reports, backfills, tagging) costs half through the Batch API.

Know your cost per task, not just your monthly bill. "Each answer costs about a cent" is a number a client can plan around.

3

A prompt in production is code.

Keep it in git. Review changes. Test before it ships. One changed word can quietly break a hundred conversations.

Production prompts

Four parts that stay put

Role, **rules**, **examples**, **format**. The user's question changes; these don't.

Need JSON for your code? Use **structured outputs**: the reply always matches your schema, so nothing breaks when the model gets chatty.

system_prompt.txt

```
<role>You answer questions for a
training institute's website.</role>

<rules>
- Answer only from <documents>.
- Not there? Say so, and give
  the contact page.
- Never promise fees or dates.
</rules>

<examples> 2-3 real Q&A pairs </examples>
<format>Short answer, then the
page title you used.</format>
```

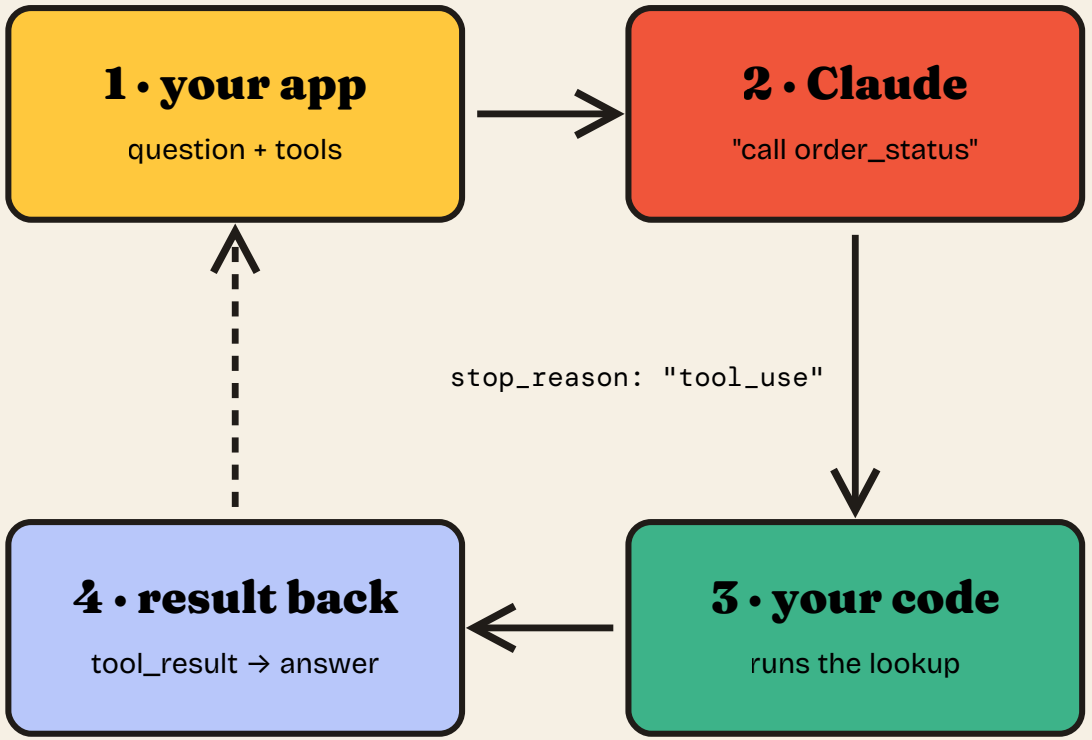
4

Claude doesn't run your code. It asks you to.

You describe a tool. Claude decides when to call it and with what. Your code runs it and hands back the result.

Tool use

The tool loop



a tool, described

```
{
  "name": "order_status",
  "description": "Status and delivery
    date for ONE order. Use when
    the customer gives an order
    number like A-10432.",
  "input_schema": {
    "type": "object",
    "properties": {
      "order_id": {"type": "string"}},
    "required": ["order_id"]}
}
```

web search & code execution run on Anthropic's side: no loop needed

5

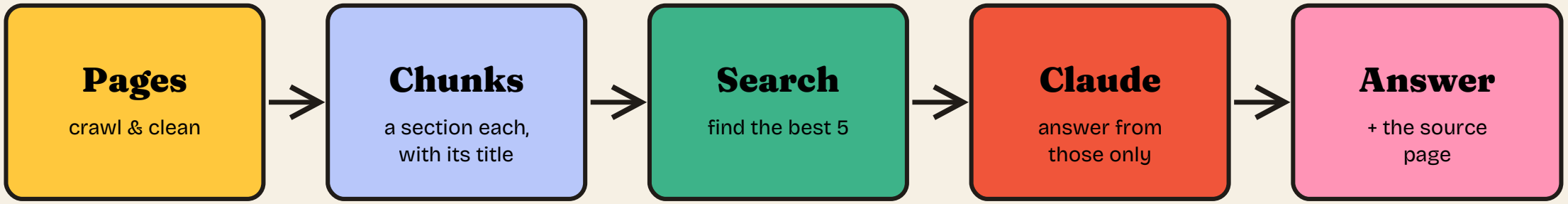
building a client's chatbot...

Their first question: "What if it makes things up about our programmes?"

So it answers only from their own website, names the page it used, and says "I don't know" when the answer isn't there.

Grounding

From pages to answers with sources



Citations on: Claude points to the exact passage it used.

Stale pages, wrong answers: re-crawl on a schedule.

6

Build the connection once. Use it everywhere.

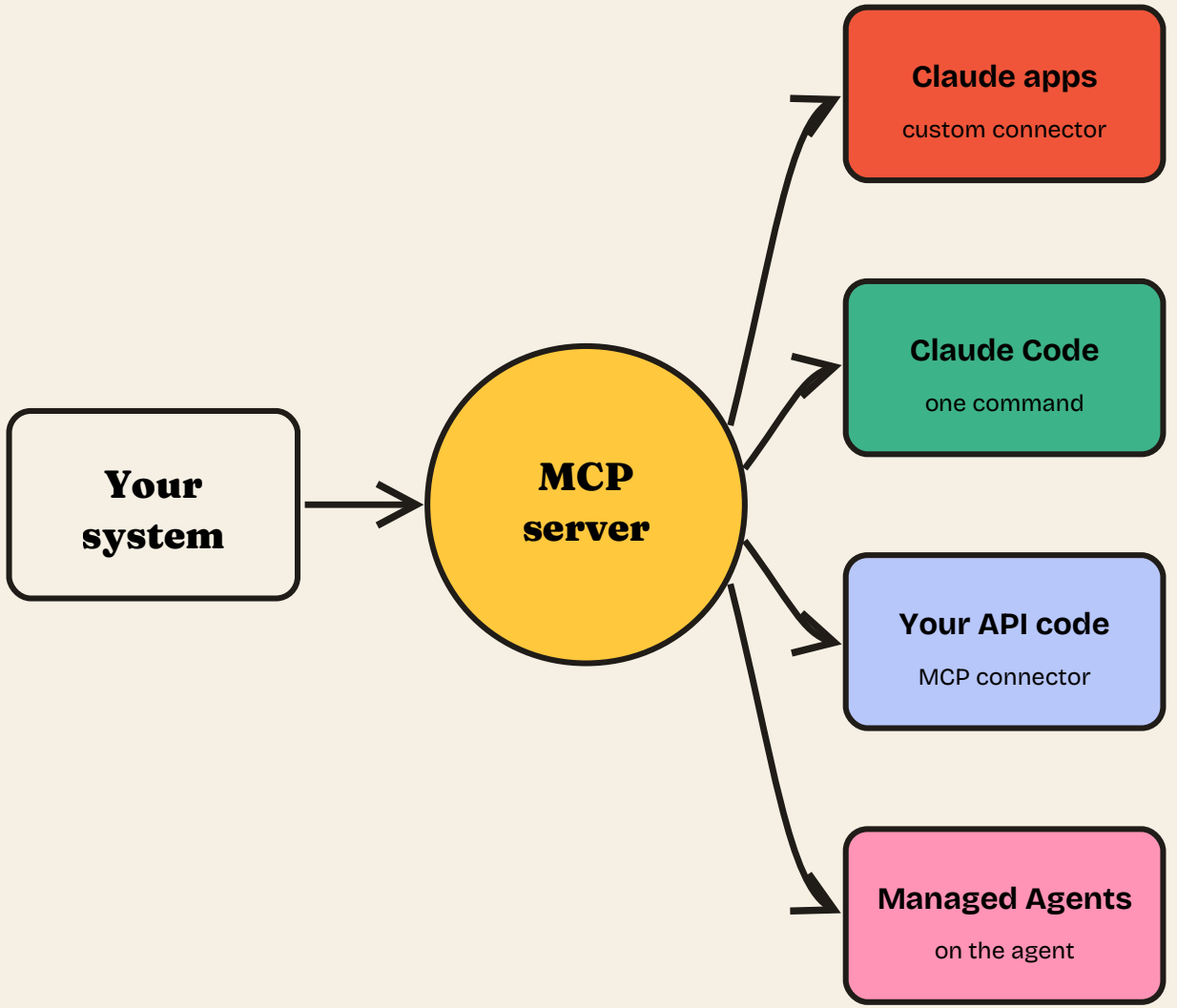
An MCP server wraps your system (orders, CRM, tickets) in a standard plug. The Claude apps, Claude Code, your API code and agents can all use the same one.

MCP

One server, four doors

Write the integration once, instead of once per app.

Keep it small: **three to five well-named tools**, read-only first. Sign people in with OAuth so each person sees only their own data.



7

for a small food shop's website...

Four stops in the code pipeline: a triage agent, a worker, a reviewer... and one human.

Only documentation changes merge on their own. Everything else waits for a person to say yes.

Agents

Do you even need an agent?

Use the simplest one that does the job. Each step right costs more and is harder to test.

ONE CALL

Most jobs

Classify, extract, summarise, draft. One prompt in, one answer out.

WORKFLOW

Fixed steps

Your code chains the calls: find, draft, check. Predictable and easy to test.

AGENT

Open-ended

Claude picks its own steps and tools. For work you can't map out in advance.



more freedom · more cost · harder to test

8

Without evals, every change is a guess.

Fifty real questions with known good answers will tell you more than a week of clicking around in a demo.

Evals

An eval set in four moves

1 · COLLECT

Real inputs

30 to 100, including the rude, the vague and the off-topic.

2 · DEFINE PASS

What good means

An exact answer, a rule in code, or a checklist Claude grades.

3 · RUN

On every change

New prompt, new model, new data: run it again.

4 · TRACK

One score

Pass rate plus cost per task, over time.

Start in the Console's **Evaluate** tab, then move the set into your repo once it grows. When a new model comes out, your eval set tells you in an hour whether to switch.

9

from running a messaging gateway...

It could reach thousands of phones. So nothing a model wrote went out without a template and a rate limit.

The more a system can do, the smaller the part the model should control directly.

Safety

Five guards for production

INPUT IS DATA

Web pages, emails and files can hide instructions. Treat them as content, never as orders.

LEAST POWER

Each tool does one thing, with the narrowest access. Read before write.

HUMAN YES

Sending, paying, deleting: a person approves anything you can't undo.

LESS DATA

Send the minimum. Mask ID and card numbers before they leave your system.

WATCH IT

Log every call. Alert on cost. Read 20 real conversations a week.

a clever attacker only needs one tool you forgot to limit

10

Clients don't buy a model. They buy something that keeps working.

Most of the work comes after launch: new questions, new content, new models. Price for that from day one.

Clients

Scope, prove, launch, run

SCOPE

One job

One success number. Real sample data before any quote.

PROVE

A small pilot

Their data, their eval set, a score they agree on.

LAUNCH

With guards

Limits, logs, and a human fallback from day one.

RUN

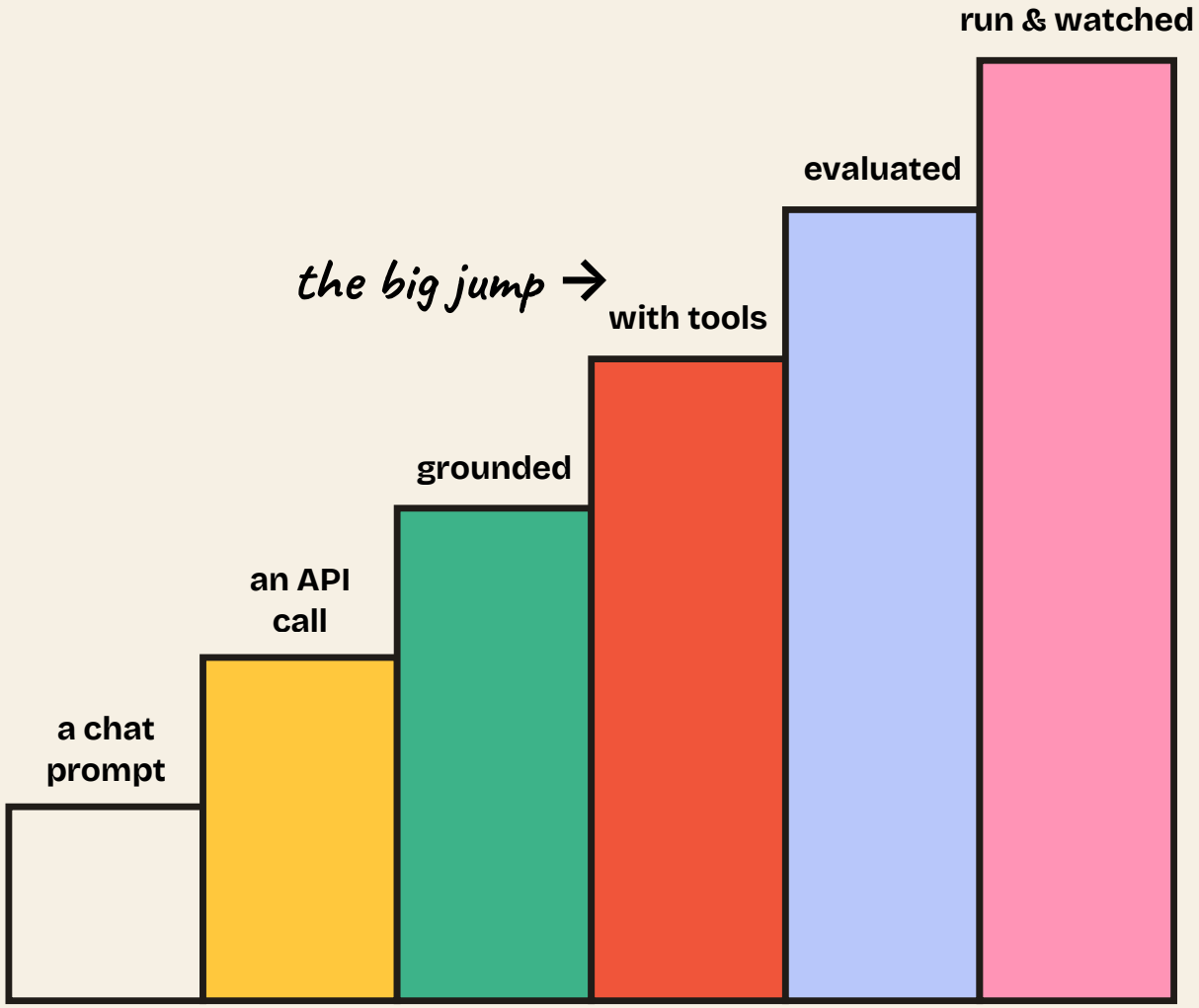
A care plan

Monthly report, evals on model updates, content refresh.

price = build fee + running costs + care plan. Never one number that hides the other two.

Where is your build?

Most demos stop at step three: it works on the questions you tried. The jump that matters is evaluated, when you can prove it works on the questions you didn't.



The builder's checklist

BUILD

- ☐ Simplest surface
- ☐ Model per step
- ☐ Caching on
- ☐ Prompt in git
- ☐ Structured output

CONNECT

- ☐ Tools well described
- ☐ Knowledge base fresh
- ☐ Citations on
- ☐ MCP for reuse
- ☐ Agent only if needed

PROVE

- ☐ 50+ eval cases
- ☐ Run on every change
- ☐ Cost per task
- ☐ Baseline saved
- ☐ Model-update plan

PROTECT

- ☐ Input treated as data
- ☐ Least-power tools
- ☐ Human yes
- ☐ Data minimised
- ☐ Logs & alerts

Now go build it.

Ship one small thing this month, with an eval set from day one.

Mian Shafiq is a technology leader, software engineer and AI practitioner with over 25 years of building software, products and teams. Today he designs and ships AI systems for clients, from chatbots to verification services and agent pipelines.

The Field Manuals: **The Claude Code Field Manual** · **Claude at Work** · **Claude for the Company** · **Building on Claude**

Free to share. Not affiliated with Anthropic. Checked against Anthropic's documentation, November 2026. Models, prices and features change, so check platform.claude.com before you quote.

shafiq.pk



yes, a real human reads every message

Questions? Building something? *Say hi.*

Tell me what you're building, what broke, and what the next Field Manual should cover.

EMAIL

talk-to-a-human@shafiq.pk

X / TWITTER

@shafiq

LINKEDIN

in/phpgurru

WEB

shafiq.pk