DevFest 2026

From REST APIs to Autonomous Systems

Backend architecture patterns on GCP to decouple *probabilistic chaos* from your core, and a real swarm that applies them.



Hi, I'm

Sebastian Alvarez

Senior Software Engineer · GDG Quito Organizer

I build Python backends, cloud infrastructure and products with applied AI.

- GDG Quito
- Quito, Ecuador

@listerineh

What we'll cover

- **01** Why an agent breaks a classic backend
- **02** Four patterns to put a boundary around it
- **03** CodeSentry Swarm: the patterns in a real system
- **04** Demo: a real PR, end to end
- **05** Production: costs, stumbles and what's missing

01

The problem

What works in the demo and falls over in production.

The contract changed

Classic backend

- One input, one known output
- Predictable latency
- Code you wrote
- State lives in your database

Agent

- Reason, act, observe, repeat
- N steps: you don't know how many
- Code nobody reviewed
- Context that grows at every step

An agent isn't a slow endpoint: it breaks your backend's contract.

Request → DB → Response assumes one input, one result and a predictable time. An agent promises none of those.

The false promise of the wrapper

PYTHON

```python
@app.post("/api/chat")
def chat(body: ChatRequest):
    # A worker blocked while the model thinks
    reply = llm.generate(body.message)
    # The model's code, inside your process
    result = exec_tool(reply.code)
    # A client retry bills everything again
    return db.save(reply, result)
```

Eight lines that pass any demo and no incident.

What breaks in production

Timeouts

The proxy cuts off and paid work is lost.

Thread starvation

Every call holds a worker. Everything falls.

Uncapped costs

Each step pays for all the context again.

No idempotency

A retry repeats the bill and the effects.

Untrusted code

It runs in your process, with your credentials.

Prompt injection

A diff can carry orders for the model.

And the platform has limits

5 min

Cloud Run default

60 min

Cloud Run maximum

600 s

Pub/Sub push maximum

30 s

load balancer default

Cloud Run, Pub/Sub and Load Balancing docs, November 2026.

Don't make the agent deterministic: put a boundary around it.

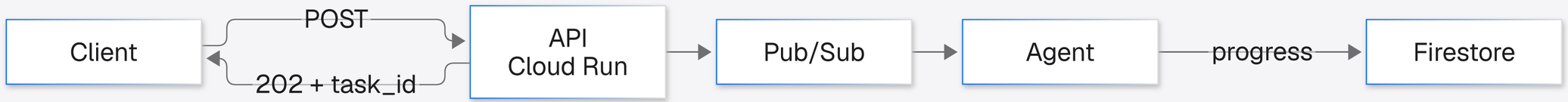
Chaos lives behind a queue and in a sandbox.
Code makes the decisions that matter.

02

Four patterns

Decouple, isolate, remember and validate.

Pattern 1: never block, enqueue



Async Request-Reply: `202` with a `task_id`, retries with backoff, dead letter and progress over SSE.

Pattern 2: all LLM code is hostile

- **Every run in its own environment**, ephemeral and disposable
- **No network:** egress denied by default
- **No identity:** no credentials, variables or metadata server
- **Hard quotas:** CPU, memory and time

Where to run generated code

Cloud Run sandboxes

`sandbox do` in your container. Preview. The demo uses it.

Code Execution

Fully managed, for any agent framework.

Agent Sandbox on GKE

Self-managed on Kubernetes, with gVisor or microVMs.

Local Docker

The demo's offline mode.

Pattern 3: hybrid memory

Working memory

Firestore or Memorystore. Fast, per task, with a TTL.

- Loop state and checkpoints
- Lets you pause for a Human-in-the-Loop
- Resume without repeating paid steps

Long term

Agent Retrieval, formerly Vertex AI Vector Search 2.0.

- Semantic search over what was learned
- Embeddings: `gemini-embedding-2`
- Changing model means re-embedding

Pattern 4: Structured Outputs

PYTHON

```python
response = await client.aio.models.generate_content(
    model="gemini-3.5-flash-lite",
    contents=prompt,
    config=GenerateContentConfig(
        response_mime_type="application/json",
        response_schema=Verdict,
    ),
)
```

A schema gives you valid JSON, not the truth. Gemini docs

03

CodeSentry Swarm

The four patterns in a system you can clone.

CodeSentry audits a PR, tests its hypothesis in a sandbox and proposes a typed fix.

An LLM reviewer has opinions. This one has to prove it with a test that fails and then passes.

Each agent does one thing

1. Triage

Reads the diff and an AST outline. Returns `hypothesis`, `test_code` and `edits`.

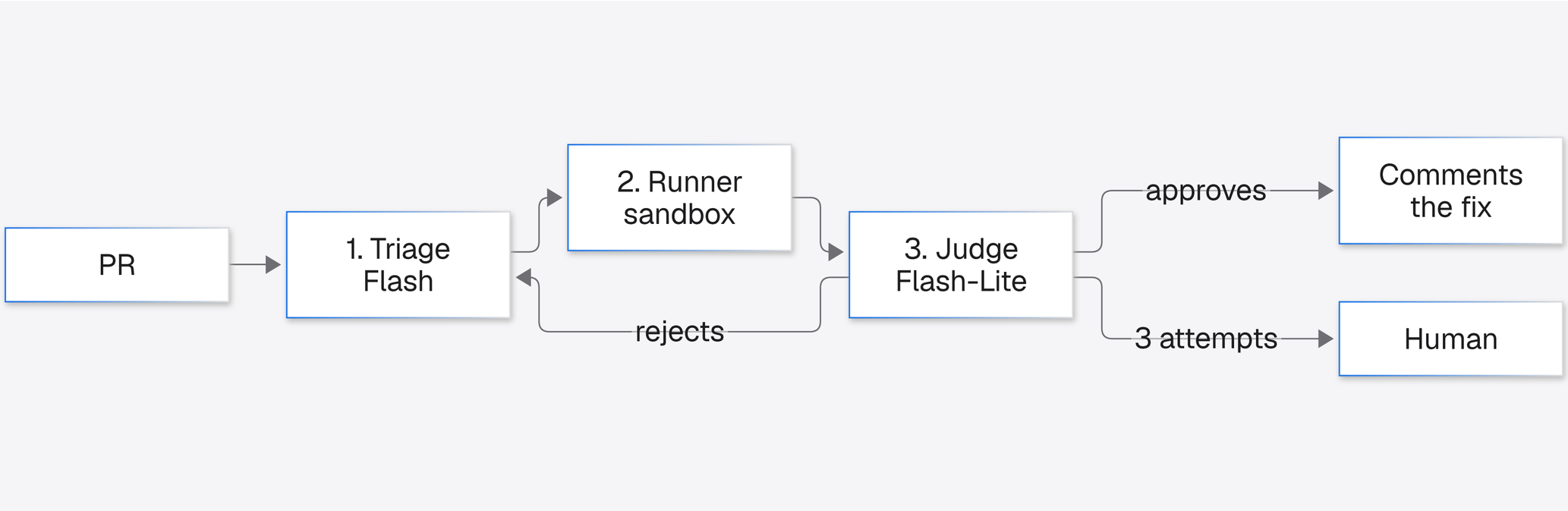
2. Sandbox Runner

Uses no model. Runs pytest and returns exit codes, stdout and tests run.

3. Judge

Receives those facts. Returns `approved`, `summary`, `risk` and `feedback`.

How the swarm works



`gemini-3.8-flash` and `gemini-3.5-flash-lite`: several small models, one job each, with code between them.

What it does and doesn't do

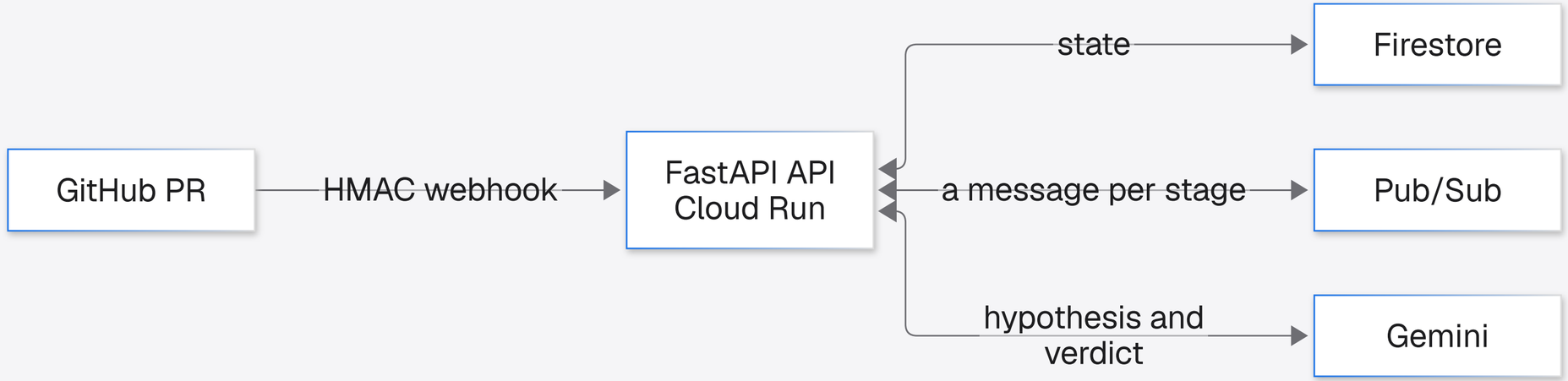
Does

- Forms a hypothesis about the diff
- Writes a test that reproduces the bug
- Tests the fix in a sandbox
- Comments the diff and a JSON report
- Tracks tokens, cost and a trace per task

Doesn't

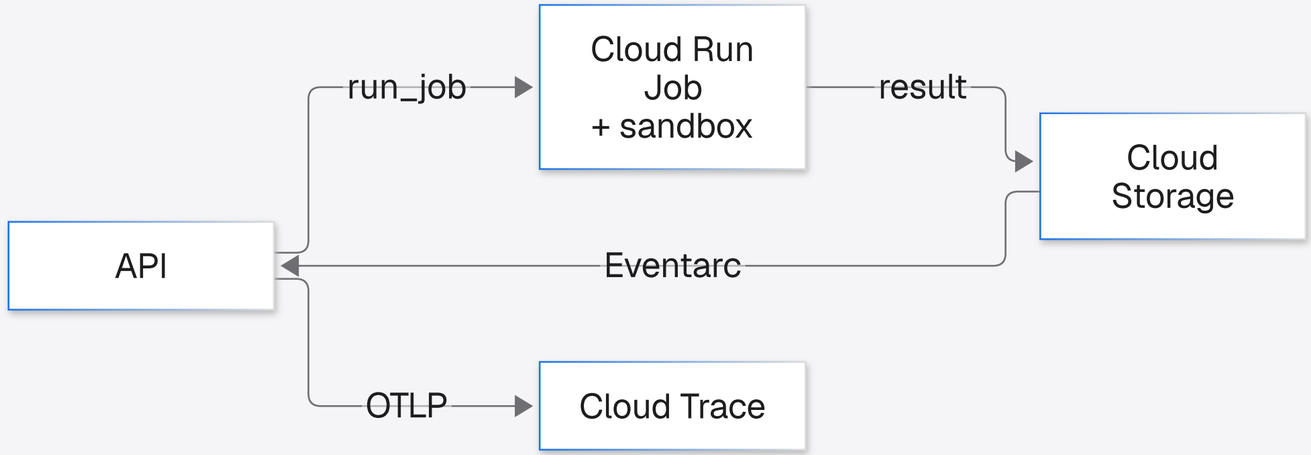
- Push or merge: merging is human
- Touch tests, CI or dotfiles
- Analyze other languages: Python only
- Learn between tasks
- Guarantee the fix: green is evidence

The real architecture: one API and one queue



The agents run inside the API: each stage is a Pub/Sub message that comes back to it.

The only thing running apart: untrusted code



The API doesn't wait for the job: the result comes back as an event. Terraform creates all of this infrastructure.

04

Demo

A real PR against a deployed system.

What we'll see

- `scripts/open-demo-pr.sh` opens a PR with a real bug
- The webhook answers `202` and enqueues the task
- Triage forms the hypothesis with Gemini
- The sandbox runs pytest with no network
- The judge decides and the swarm comments the fix
- Cloud Trace shows the whole task in one trace

When the hypothesis fails

```text
iter 1  hypothesis: the sign is flipped, use abs()
        before the fix  exit 1  assert 110.0 == 100
        with the fix    exit 1  assert 90.0 == 100
        judge           rejects

iter 2  hypothesis: the percent has no range, clamp it to 0..100
        before the fix  exit 1  assert 110.0 == 100
        with the fix    exit 0  4 passed
```

`uv run codesentry demo`: real pytest, scripted model answers.

The journey of a real PR

```text
06:38:08  task.submit
06:38:10  agent.triage    gemini-3.8-flash
06:38:58  agent.run       sandbox.execute
06:40:42  sandbox.result  arrives through Eventarc
06:40:43  agent.judge     rejects: iteration 2
06:40:45  agent.triage    gemini-3.8-flash
06:42:24  sandbox.result  arrives through Eventarc
06:42:25  agent.judge     approves and comments
```

Real spans from PR #5 in Cloud Trace: 2 iterations in 4 min 22 s.

I tested the sandbox from inside

```text
metadata_ip           blocked  Network is unreachable
metadata_host         blocked  Network is unreachable
internet              blocked  Temporary failure in name resolution
signed_urls_visible   []
credentials_visible   []
```

`scripts/open-escape-pr.sh` on Cloud Run, 4 November 2026. It blocks what I tested, not everything possible.

Measured on real GCP

106 s
fastest PR

262 s
slowest PR

$0.016
highest cost

15 spans
in one trace

3 real PRs on codesentry-demo-shop, 4 November 2026. Cost from the demo's price table.

05

Production

Costs, stumbles and what's missing.

Guardrails in code, not in the prompt

Budget per task

20,000 tokens, counted before each call.

3 iterations at most

Then it escalates to a human.

Hard timeouts

45 s per model call and a limit per run.

Circuit breaker

After 3 failures it stops insisting on the main model.

Light fallback

Better a Lite model than a crash.

A human at the end

The agent proposes; someone decides.

What it costs and what lowers it

Costs

- USD 0.011 to 0.016 per measured PR
- Triage output is nearly all of it: 3,630 of 3,690 tokens on PR #4
- Idle, Cloud Run scales to zero

Lowers it

- A light model for judging
- A token cap per task
- Ignoring pull requests from forks
- Not tested yet: limiting Gemini's thinking

The road for CodeSentry

Works today

- Deployed on GCP with real PRs
- Sandbox tested from inside
- One trace end to end
- Offline mode with replay
- 79 tests and green CI

Still missing

- Test the local Docker sandbox
- Clean up stuck tasks
- Process the dead letter
- A deployment pipeline
- Long-term memory

Your Monday checklist

- Treat the agent as an **untrusted external service**
- **Isolate** all dynamic compute
- Build on **events**, never on blocking HTTP
- **Type and validate** every input and output
- Keep the **final decision in code**

Questions?

The demo is open source: clone it, run it without a cloud and break it.

CodeSentry Swarm

Demo shop

linkedin.com/in/listerineh

listerineh.dev/slides