

Welcome back. You are signed into your member account
bg****@jaxondigital.com. Not you?

X

23blocks · [Follow publication](#)

Member-only story

My AI Agents have their own inbox and know how to use it. A Claude Code Skill Story.

12 min read · 2 days ago



Juan Pelaez

Follow

Listen

Share

More

```
AI Maestro
AI Agents
  authentication (Idle)
  jarvis (Idle)
  products (Idle)
  search (Idle)
  Apps
    app (Idle)
    prompthub (Idle)
  FLUIDMIND
    app (Idle)
  CUSTOMERS
    Startalking
      app (Idle)

Terminal
  customers-zoom-admin Connected
    > check your messages
      Use skill "agent-messaging"?
      Claude may use instructions, code, or files from this Skill.
      Send and receive messages between AI agent sessions using AI Maestro's messaging system. Use this skill when the user asks to "send a message", "check inbox", "read messages", "notify [session]", "tell [agent]", or any inter-agent communication. (project, gitignored)
      Do you want to proceed?
        1. Yes
        2. Yes, and don't ask again for agent-messaging in /Users/juanpelaez/3Metas/services/development/zoom/web-apps/zoom-admin
        3. No, and tell Claude what to do differently (esc)

Session Notes
[customers:node*] "Juans-MacBook-Pro.local" 15:55 17-Oct-25
```

I was running three AI coding agents simultaneously when I realized I had become the bottleneck in my own AI-assisted workflow.

The backend agent (Claude Code) had just finished implementing a REST API endpoint. Welcome back. You are signed into your member account [REDACTED] was waiting to implement tmux session switching. It was switching tmux sessions, pasting context, answering questions, switching back, getting more details, switching forward, pasting again.

I was a human relay server in an otherwise automated process.

That's when it hit me: if these agents are supposed to be intelligent collaborators, why can't they talk to each other directly?

A few days later, they could.

• • •

What to read this free? Use this link <https://juan-pelaez.medium.com/my-ai-agents-have-their-own-inbox-and-know-how-to-use-it-a-claude-code-skill-story-88ddc60d7062?sk=a5ecec433a0a2dfb1f24912fa5e93757>

• • •

This is the story of building AI Maestro's agent communication system. The technical decisions, the breakthrough moment with Claude Code skills, and what happens when AI agents get their own inboxes.

• • •

The Multi-Agent Coordination Problem

Let me paint the picture of what multi-agent development actually looks like.

I run AI Maestro, an open-source tmux dashboard for managing multiple AI coding agents. On any given day, I might have:

- `fluidmind-agents-backend-architect` (OpenAI Codex) — API development
- `fluidmind-agents-frontend-developer` (Claude Code) — React components
- `fluidmind-agents-devops-engineer` (Claude Code) — Infrastructure as code

- `ecommerce-dev-testing` (Claude Code) — Integration testing

Welcome back. You are signed into your member account

Each agent has their own session, isolated from others, and can only see what's in their own session.

bg***@jaxondigital.com**

and in

isolated silos.

Here's what a typical coordination flow looked like:

9:47 AM — Backend agent finishes `/api/users` endpoint

Backend session output:

- ✓ Implemented GET /api/users
- ✓ Added pagination support (?page=1&limit=20)
- ✓ Returns: { `users`: User[], total: number, page: number }
- ✓ Rate limited: 100 requests/hour per IP

9:48 AM I manually copied those specs

9:49 AM Switch to frontend session (Cmd+Tab, click, wait for context)

9:50 AM Paste specs, explain context

Me: "The backend team finished the users endpoint. Here are the specs..."

9:51 AM Frontend agent asks clarifying questions

Frontend agent: "What's the error response format for rate limiting?"

9:52 AM Switch back to backend session

9:53 AM Ask backend agent about error responses

9:54 AM Get answer, switch to frontend

9:55 AM Paste answer

Result: 8 minutes of context switching for something that should take 30 seconds.

Welcome back. You are signed into your member account
Multiply this by 100 agents...
just being a message relay.

2–3 hours

The agents weren't the bottleneck — I was. It was a problem I had to solve, and I built the solution.

• • •

First Attempt: Share Files

My first instinct was simple: share markdown files.

```
~/.aimaestro/shared/
└── backend-to-frontend.md
└── frontend-to-backend.md
└── general-updates.md
```

Each agent could write or update these files. Other agents could read them when needed.

It failed spectacularly.

Problem 1: No notifications

Agents didn't know when new messages arrived. I still had to tell them, "Check the shared file."

Problem 2: Still manual

I still had to tell Agent A to write, then tell Agent B to read. The coordination problem remained.

After a week, I scrapped it.

• • •

Second Attempt
Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

Okay, I thought, let me build this properly. A real messaging system.

Architecture v1:

```
Agent A → HTTP POST → Direct File Write → JSON files  
Agent B → HTTP GET → Read JSON files → Display
```

I built:

- REST API endpoints (`POST /api/messages`, `GET /api/messages`)
- JSON storage in `~/.aimaestro/messages/inbox/{sessionName}/`
- Message metadata (from, to, subject, priority, type, timestamp)
- Read/unread status tracking

This actually worked.

Backend agent could POST a message:

Open in app ↗

≡ Medium



```
"subject": "API Ready",  
"content": {"message": "GET /api/users implemented at routes/users.ts:45"}  
}'
```

Frontend agent could GET messages:

```
curl http://localhost:23000/api/messages?session=frontend-dev
```

But it still had a critical flaw: the agents didn't know how to use it

I had to write these parameters.

Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

It was better than shared files, but I was still the middleman. The agents weren't communicating, I was making them communicate.

• • •

The Breakthrough: Claude Code Skills

Then, in October 2024, #Anthropic introduced #Claude Code Skills.

Skills are capabilities you can teach Claude by creating markdown files. They're:

- **Model-invoked.** Claude decides when to use them (not user-triggered)
- **Portable.** The same skill works across all projects and interfaces
- **Natural language.** No command memorization required
- **Progressive disclosure.** Only loaded when relevant

I immediately saw the potential.

What if I could teach every AI agent *how to communicate* as a core skill?

• • •

Designing the Agent Messaging Skill

I started writing `~/claude/skills/agent-messaging/SKILL.md`

The skill needed to teach Claude:

1. When to recognize messaging intent

```

## When      Welcome back. You are signed into your member account
**Send:    bg****@jaxondigital.com.
- User
- User says "notify [session]" or "alert [agent]"
- User wants YOU to communicate with ANOTHER agent session
**Receiving:**
- User says "check my inbox" or "check for messages"
- User says "read my messages" or "show messages"
- User asks "any new messages?"
```

2. Which tools to use

I had two tools:

- `send-aimaestro-message.sh` — File-based persistent messaging
- `send-tmux-message.sh` — Instant tmux popup notifications

The skill needed is decision logic:

```

## Decision Guide
**Use file-based when:**
- Message contains detailed requirements or context
- The recipient needs to reference it later
- Communication is structured (priority, type)
**Use instant when:**
- Urgent attention needed (minutes)
- Quick FYI ("build done", "tests passing")
- Production emergency
**Use both when:**
- Critical AND detailed information needed
```

3. How to execute the tools

Complete examples for every scenario:

```

# Request work from another agent
send-aimaestro-message.sh backend-api \
"Need GET /api/users endpoint" \
"Building user list UI. Need endpoint returning array of users with {id, name,
```

```
high \
request
# Urgent \
send-tr
send-aimaestro message.sh backend/api \
"Production: API failing" \
"All /users endpoints returning 500. Database timeout. ~100 users affected." \
urgent \
notification
```

4. Critical context boundaries

This was crucial. I needed agents to understand:

- **YOUR inbox** = Messages sent TO YOUR SESSION by OTHER AGENTS
- You do NOT read the operator's messages
- You do NOT read other agents' private inboxes
- This is agent-to-agent communication only

```
## CRITICAL: Inter-Agent Communication
**YOU ARE AN AGENT** - This skill is for agent-to-agent communication.
- **Your inbox** = Messages sent TO YOUR SESSION by OTHER AGENTS
- **Your session name** = The tmux session you're running in
- **Your inbox location** = `~/.aimaestro/messages/inbox/YOUR-SESSION-NAME/`  

**You do NOT read:**
- X The human operator's messages
- X Other agents' private inboxes
- X Messages not addressed to your session
```

The skill file ended up being 338 lines of carefully crafted instructions, examples, and decision trees.

Testing the Skill

I installed the skill globally:

```
~/.clau Welcome back. You are signed into your member account  
bg****@jaxondigital.com.
```

Restarted my Claude Code sessions. And tried it.

Backend session:

```
Me: "Send a message to frontend-dev that the /api/users endpoint is ready at ro
```

Claude's response:

```
I'll send a message to the frontend-dev agent about the completed API endpoint.  
[Executing: send-aimaestro-message.sh frontend-dev "API Endpoint Ready" "GET /a  
✓ Message sent to frontend-dev  
I've notified the frontend developer that the /api/users endpoint is ready  
with details about the implementation and pagination support.
```

I didn't write a single line of integration code. I just talked to Claude in natural language.

I switched to the frontend session.

Frontend session:

```
Me: "Check your inbox"
```

Claude's response:

Let me [Executive] You Welcome back. You are signed into your member account
[Executive] **bg****@jaxondigital.com.**

Message: msg_1737148625_abc123
From: backend-architect
To: frontend-dev
Subject: API Endpoint Ready
Priority: normal
Type: response
Status: unread
Timestamp: 2025-01-17 14:23:45
Content: GET /api/users endpoint implemented at routes/users.ts:120.
Returns paginated user list with {id, name, email}. Supports ?page=1&limit=20.
Would you like me to start integrating this API endpoint into the user list com

It worked. Perfectly.

The agents were communicating without me being the middleman.

Real-World Results

After deploying the system across all my agent sessions, here's what changed:

Quantitative Impact

Before:

- Average coordination time: 8 minutes per interaction
- Context switches per day: 40–60
- Time spent relaying messages: 2–3 hours/day
- Lost context events: 5–10/day

After:

- Average coordination time: 30 seconds
- Context switches per day: 5–10 (only for complex decisions)
- Time spent relaying messages: ~15 minutes/day
- Lost context events: 0–1/day

Time saved: ~2.5 hours per day

Welcome back. You are signed into your member account
More important: bg****@jaxondigital.com.

Qualitative Changes

1. Agents coordinate proactively

Backend finishes a feature:

```
Backend (to frontend): "Auth middleware ready. All /api/* routes now require va  
'Authorization: Bearer {token}'"
```

Frontend gets context immediately and can ask clarifying questions *directly to the backend agent*:

```
Frontend (to backend): "The auth middleware - does it handle token refresh  
automatically or should I implement client-side refresh logic?"
```

2. Async collaboration actually works

I can start agents working on different parts of a feature, go get coffee, and come back to them having coordinated the integration points without me.

3. Context preservation

All coordination is logged. When I return to a session days later, I can read the message history and instantly understand what happened:

```
Inbox (3 messages):  
1. From: devops-engineer - "Deployment succeeded to staging"  
2. From: backend-architect - "Migration scripts ran successfully"  
3. From: frontend-dev - "Verified UI changes on staging, looks good"
```

4. Reduced cognitive load

Welcome back. You are signed into your member account
I'm no longer **bg****@jaxondigital.com**.
...
it.

Design Considerations & Lessons Learned

1. Natural Language > Commands

The initial design had agents using explicit commands:

```
send-message backend-architect "content here"
```

Current design uses natural language:

```
Tell backend-architect about the API error
```

Why it matters: Cognitive load. Natural language fits the conversational flow of working with AI agents. Commands break that flow.

The skill system made this possible. Without skills, I'd have needed command parsing. Skills give Claude the context to understand intent.

2. Agent Identity is Critical

Early versions didn't clearly establish agent identity. This confused:

- Agents tried to read other agents' private messages
- Unclear who "you" referred to
- Ambiguous inbox ownership

Solution: The skill explicitly teaches identity:

YOU = T
YOUR in>Welcome back. You are signed into your member account
YOUR se**bg****@jaxondigital.com**.

4. Metadata Enables Workflows

Messages have rich metadata:

```
{  
  "priority": "urgent",  
  "type": "request",  
  "timestamp": "2025-01-17T14:23:45Z"  
}
```

This enables:

- Filtering by priority
- Automatic urgent message highlighting
- Type-based UI rendering
- Time-based sorting

5. Skills Need Comprehensive Examples

The skill file is 40% examples. This is intentional.

Claude learns from examples better than from abstract instructions. The skill includes:

- 4 receiving scenarios (check inbox, quick count, respond to request, handle urgent)
- 4 sending scenarios (request work, urgent alert, progress update, reply)
- Error handling examples
- Edge cases

More examples = Better skill activation accuracy

Welcome back. You are signed into your member account
6. Security **bg****@jaxondigital.com**.

Agents have significant power. They execute shell commands, modify files, and access APIs.

The messaging skill needed clear boundaries:

- Agents can only read THEIR OWN inbox
- Messages are agent-to-agent only (not agent-to-operator) Yet?
- No access to other agents' private messages

This required explicit teaching in the Skill:

- **You do NOT read:****
- ✗ The human operator's messages
 - ✗ Other agents' private inboxes
 - ✗ Messages not addressed to your session

The Web UI: Making Messages Accessible

While CLI tools work great for agents, humans need a UI.

Each session in AI Maestro now has a **Messages tab** with:

Inbox View:

- All messages sent TO this agent
- Filter by: read/unread, priority, type, sender
- Unread count badge
- Click to mark as read

Sent View:

- All messages FROM this agent

- Delivery confirmation

Welcome back. You are signed into your member account

- Response **bg****@jaxondigital.com**.

Compose View:

- Target agent selector (dropdown of all sessions)

- Subject line

- Message content (textarea with markdown support)

- Priority selector (urgent/high/normal/low)

- Type selector (request/response/notification/update)

- Send button

The UI bridges the gap between agent automation and human oversight. Agents use CLI tools, humans use the web interface, and both work with the same underlying message store.

• • •

Future Implications

This system opens up possibilities I'm only beginning to explore:

1. Agent Task Delegation

Me (to project-manager agent): "Coordinate the team to implement user authentication."
Project manager:

- Sends to backend-architect: "Implement JWT auth middleware" (high-priority request)
- Sends to frontend-dev: "Create login/signup forms" (high priority request)
- Sends to devops-engineer: "Set up Redis for session storage" (normal priority request)
- Sends to testing-agent: "Create auth integration tests" (normal priority request)

Each agent:

- Checks inbox
- Sees request from project manager
- Works on their part
- Sends progress updates back
- Sends completion notification

Project manager:

- Aggregates updates
- Reports

Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

2. Agent Collaboration Patterns

Pair Programming Pattern:

Backend: "I'm stuck on this database query optimization."
Database-Expert: "Let me look at your query..."
[Reviews code]
Database-Expert: "Try adding an index on user_id and created_at."
Backend: [Implements suggestion]
Backend: "That reduced query time from 2s to 40ms. Thanks!"

Code Review Pattern:

Frontend: "PR ready for review: user-profile-redesign"
Security-Reviewer: "Found XSS vulnerability in profile bio rendering"
Frontend: "Fixed. Now using DOMPurify for sanitization."
Security-Reviewer: "LGTM. Approved."

Deployment Coordination:

Backend: "API v2 deployed to staging"
Frontend: "Detected breaking changes in /api/users response format"
Backend: "Ah, we renamed 'username' to 'displayName'"
Frontend: "Updated. Ready for production deployment."
DevOps: "Both deployed. Monitoring for issues..."

3. Multi-Agent Workflows

Imagine a bug fix workflow:

1. The monitoring agent detects a production error

2. Sends urgent alert to on-call agent

3. On-call
bg****@jaxondigital.com.

4. Sends a request to the backend architect for a fix

5. Backend architect implements a fix, sends it to the testing agent

6. Testing agent runs regression suite, confirms fix

7. Sends approval to devops engineer

8. DevOps engineer deploys to production

9. Sends confirmation to the monitoring agent

10. The monitoring agent verifies fix has resolved the issue

11. Sends a success notification to all involved agents

All of this could happen while I sleep.

. . .

When to Use This Skill

Sending (Agent-to-Agent):

- User (operator) says “send a message to [another-agent-session]”
- User says “notify [another-agent]” or “alert [another-agent]”
- User wants YOU to communicate with ANOTHER agent session
- You need to send urgent alerts or requests to OTHER AGENTS

Receiving (Check YOUR OWN Inbox):

- User says “check my inbox” = Check messages sent TO YOUR SESSION
- User says “read my messages” = Read messages sent TO YOU (this agent)
- User asks, “Any new messages?” = Check YOUR inbox for new messages

More specific language reduced false positives dramatically

Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

Takeaways for Building AI Agent Systems

If you're building multi-agent systems, here's what I learned:

1. Communication is Infrastructure

Don't treat agent coordination as an afterthought. It's foundational infrastructure, like logging or monitoring.

Build it early. Build it well.

2. Skills > Scripts

Claude Code skills let agents understand intent, not just execute commands.

```
# Script approach (rigid)
send-message.sh backend-architect "message"
# Skill approach (flexible)
"Tell backend-architect about the error"
"Notify the backend team"
"Send a message to backend explaining the issue"
```

All of these work because the skill understands intent.

3. Two Channels > One Channel

Urgent situations need instant notifications. Detailed collaboration needs structured messages.

Don't try to force one channel to do both jobs.

4. Agent Identity Must Be Explicit

Agents need to know:

- Who they are (session name)
- Where
Welcome back. You are signed into your member account
bg**@jaxondigital.com**.
- Who they're talking to (other agents, not operators)

Ambiguity causes chaos.

5. Examples Train Better Than Rules

The skill is 40% examples for a reason. Claude learns from examples.

Show, don't just tell.

6. Build for Humans AND Agents

Agents use CLI tools. Humans use web UIs. Both need access to the same data.

Design your data layer to support both interfaces.

• • •

The Bigger Picture

We're in the early days of multi-agent AI systems.

Right now, most people use AI as a single assistant. Ask a question, get an answer.
Write some code, get a review.

But the future is collaborative AI teams:

- Backend architect
- Frontend developer
- DevOps engineer
- Security auditor
- Performance optimizer
- Documentation writer
- Testing specialist

Each specialized. Each expert is in their own domain. Each is coordinating with the others.

Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

For that future to work, agents need to communicate.

Not through you.

Not with you relaying messages.

Directly. Agent-to-agent.

This system is a step toward that future.

• • •

Try It Yourself

AI Maestro is open source (MIT license) and available on GitHub.

Repository: <https://github.com/23blocks-OS/ai-maestro>

• • •

Documentation

- Send your first message in 2 minutes. [Quickstart Guide](https://github.com/23blocks-OS/ai-maestro/blob/main/docs/AGENT-COMMUNICATION-QUICKSTART.md)(<https://github.com/23blocks-OS/ai-maestro/blob/main/docs/AGENT-COMMUNICATION-QUICKSTART.md>).
- Technical details. [Architecture Deep-Dive](https://github.com/23blocks-OS/ai-maestro/blob/main/docs/AGENT-COMMUNICATION-ARCHITECTURE.md)(<https://github.com/23blocks-OS/ai-maestro/blob/main/docs/AGENT-COMMUNICATION-ARCHITECTURE.md>)
- Skills setup. [Claude Code Configuration](https://github.com/23blocks-OS/ai-maestro/blob/main/docs/CLAUDE-CODE-CONFIGURATION.md)(<https://github.com/23blocks-OS/ai-maestro/blob/main/docs/CLAUDE-CODE-CONFIGURATION.md>)

• • •

What's Next

I'm continuing to explore multi-agent coordination patterns:

- Task delegation hierarchies
- Long-term agent collaboration

Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

- Agent collaboration workflows
- Automated code review chains
- Coordinated deployment pipelines

If you're working on multi-agent systems, I'd love to hear what you're building.

Connect:

- Twitter/X: [@jkpelaez](https://x.com/jkpelaez) / [@aimaestro23](https://x.com/aimaestro23)
- GitHub: [23blocks-OS/ai-maestro](https://github.com/23blocks-OS/ai-maestro)
- LinkedIn: [Juan Peláez](https://www.linkedin.com/in/juanpelaez)

The future of AI isn't one assistant. It's a team of specialists coordinating seamlessly.

Let's build it together.

Built with ❤ in Boulder, Colorado

Juan Peláez • Founder, 23blocks • Coded with Claude

Ai Agent

Devtools

AI

Future Of Work

Follow

Published in 23blocks

4 followers · Last published 2 days ago

Enabling People and Companies to build a new generation of software solutions. Software, Infrastructure, and services by

Welcome back. You are signed into your member account
bg****@jaxondigital.com.



Follow

Written by Juan Pelaez

55 followers · 120 following

Dad. Rock Climber. Ironman and 70.3 finisher (6x). Marathon and Ultra Marathon runner (2x). Scuba Diver. Biker, also CTO and Founder at 3Metas and 23blocks.

No responses yet



What are your thoughts?

More from Juan Pelaez and 23blocks

Welcome back. You are signed into your member account
bg**@jaxondigital.com**.

 In 3Metas by Juan Pelaez

From WCF on-premise to Azure App Service. Part I

Moving legacy .Net 4 WCF Services to Azure App Service on the Cloud. Lessons Learned

Apr 11, 2023  1



...

 In 23blocks by Juan Pelaez

Using PGVector with a multitenant Ruby on Rails App.

This is a short one. When using the pgvector extension to enable vector searches on Ruby On Rail Applications, if the application is also...

Jan 7, 2024



...

Welcome back. You are signed into your member account
bg****@jaxondigital.com.

In 23blocks by Juan Pelaez

AWS Amplify, different build settings for different branches.

Using AWS Amplify there are cases when you want to use different build settings per branch, however, as all the branches share the same...

Aug 21, 2022



...

In 23blocks by Juan Pelaez

Building AI Maestro: From 47 Terminal Windows to One Beautiful Dashboard

Perfect for devops, SRE, and system administrators

Welcome back. You are signed into your member account

bg****@jaxondigital.com

Oct 10



...

See all from Juan Pelaez

See all from 23blocks

Recommended from Medium

In The Context Layer by Jannis

Claude's New "Skills" Show How Anthropic Is Layering Intelligence on Top of MCP

3d ago 202 1



...

Welcome back. You are signed into your member account
bg****@jaxondigital.com.

 aakash

Perplexity Just Unleashed 10 FREE AI Agents That Do Your Entire Job (The “Comet” Shortcut)

Stop what you are doing. The era of the simple AI search engine is officially over.

 Oct 7  561  15



...

 Ahmed Shika Shaker

The AI Development Revolution: Building Intelligent Agent Swarms with Claude Code

The landscape of AI-assisted development is rapidly shifting. We're not talking about AI-assisted

Welcome back. You are signed into your member account

bg**@jaxondigital.com**.

Oct 13  57



...

 Riccardo Tartaglia

5 Essential MCP Servers Every Developer Should Know

I've been experimenting with Model Context Protocol servers for a few months now, and I have to say, they've changed the way I work.

Oct 11  18  2



...

 Tosny

7 Websites

Welcome back. You are signed into your member account

bg****@jaxondigital.com.

If there is one

★ Sep 23

4.8K

175



...



In Block Magnates by Farhad Ali

I Tried Trading With Agentic AI, and It's Mind Blowing

My biggest enemy in the market wasn't volatility; it was my own emotional brain. This is the story of how I built an AI co-pilot to save me...

★ Oct 9

110



...

See more recommendations