- CATALOG
Plot Twist
Understanding What We …
The AI-Native Architectur…
Using Claude Code for R…
Conversation
Discovering Existing MC…
Choosing an Email Service
Alternative Email Services
System Architecture Des…
MCP Server Strategy
Technology Stack Summ…
The Development Workfl…
What Makes This AI-Nati…
Setting Up for Part 4
Conclusion: Why This M…
~ Finally

Generative AI (/tags/generative-ai) Claude Code (/tags/claude-code) Anthropic (/tags/anthropic) AI-assisted Coding (/tags/ai-assisted-coding)

Building an Event Management System with Claude Code: Part 3 - Architecting an AI-Native System

Posted by nielsb on Sunday, December 21, 2025

Welcome back to **Building an Event Management System with Claude Code** blog post series! If you've been following along, you know we've covered how to install Claude Code (which has gotten significantly simpler since we started - see edited Part 1 (/post/2025-07-29-building-an-event-management-system-with-claude-code-claude-code-installation-and-initialisation/) and Part 2 (/post/2025-08-13-building-an-event-management-system-with-claude-code-part-2---ide-integration-and-advanced-features/)) and how to integrate it with your IDE. We even built a basic `Node.js` application with some API endpoints. That was all great learning, but here's the thing: I've been so busy organising **Data & AI Community Day Durban** events that I haven't had time to write about building a system to manage them. The irony is not lost on me.

- To see all posts in the series, go to: Building an Event Management System with Claude Code (/contact-event-claude-code).

While not working on this blogpost series, but instead organising and prepping demos for a couple of events, I had time to reflect on what I would like the system to be. When I started this series, I had a clear picture in my head of what I wanted. However, when prepping event demos and seeing what LLMs together with MCP Servers are capable of, I wanted something different.

Plot Twist

Initially, I thought a simple web app with forms to manage events, attendees, and speakers would suffice. But as I explored the capabilities of LLMs and MCP servers more deeply, I realised that I could build something much more powerful, something that leverages AI not just to help me code faster, but to fundamentally change how I interact with the event management system. So, we're going to build an **AI-native system**, not just using AI to build a traditional web application.

What's the difference? An AI-native system uses artificial intelligence not just in its construction but also in its core operations. Instead of building a traditional CRUD application with web forms and dashboards, I'm building a system where the primary interface is conversational; where I can ask questions like "How many people attended last year's event?" or "Email all speakers from 2024 about the next conference", and the system just handles it.

This is the perfect use case for **Model Context Protocol** (**MCP**) servers, and that's what makes this blog series timely and valuable to you.

Understanding What We Actually Need

Before we dive into architecture and technology choices, let me clarify what this system actually needs to do. It's not a full-featured event management platform; we're not replacing Eventbrite or building registration systems. What we need is something more focused: a **Contact Relationship Management (CRM) and Communications Hub** specifically for the **Data & AI Community Day Durban** events.

Here's the reality of my workflow:

- **Attendee registration** happens through Quicket (a third-party ticketing platform)
- **Speaker submissions** happen through Sessionize (a speaker management platform)
- **Check-ins** happen through Quicket on the day of the event
- **Everything else** currently happens through Brevo or manual spreadsheet juggling

It is the last bullet, "Everything else", that I want this system for. More specifically:

1. **Stores contacts:** A unified database of all attendees and speakers with their participation history
2. **Tracks participation:** Who registered for which event, who actually attended, who spoke at which event

3. **Handles communications:** Email workflows for speakers and attendees, pulling contacts from our database
4. **Provides insights:** “Show me speaker retention”, “How many first-time attendees did we have?”, “Which topics had the highest attendance?”
5. **Imports data:** Batch upload registration data from Quicket, speaker data from Sessionize

Critically, I don’t want to build another web dashboard full of forms and tables. I want to ask questions and give commands in natural language, and have the system handle it. That’s where MCP comes in.

The AI-Native Architecture Vision

Let me start with how I envision using this system daily, because that vision drives the architecture:

Me: “Is Jane Smith registered for the upcoming conference?”
System: “Yes, Jane Smith (jane.smith@example.com (mailto:jane.smith@example.com)) registered on December 15, 2024. She also attended the 2023 and 2024 conferences as an attendee.”

Me: “How many people attended last year’s event compared to 2023?"
System: “2024 had 287 attendees (92% of registrations). In 2023, 245 attendees (88% of registrations) attended. That’s a 17% increase in attendance.”

Me: “Send an email to all 2024 speakers announcing the call for papers for 2025.”
System: “I’ve queued an email to 23 speakers from 2024. Would you like to preview the email list before sending?”

This conversational interface isn’t science fiction; it’s absolutely achievable with MCP servers and Claude (or any LLM). The system consists of:

1. **Data Layer:** PostgreSQL database with our contacts, events, and participation records
2. **MCP Server Layer:** Tools that let Claude interact with the database and external services
3. **Conversational Interface:** Claude.ai/Gemini (initially), where I interact with the system

No web forms. No admin dashboards. Just conversations that get things done.

Using Claude Code for Requirements Analysis

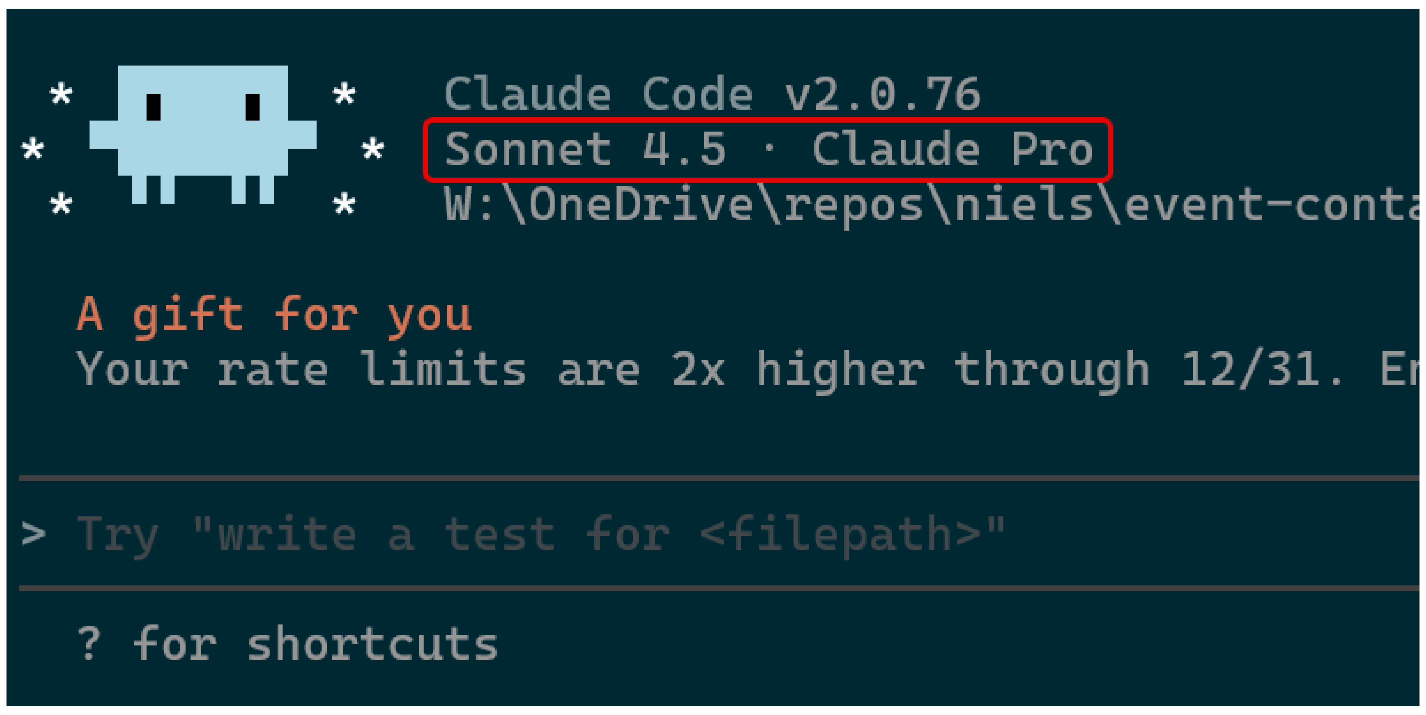
Now, here’s where Claude Code gets really interesting. Before I start building anything, I want to have a conversation with Claude Code about what we’re building. Not to write code yet, but to think through the architecture.

So let us do that. The code for this will be in a new directory `event-contact-management-2` :

```
# create a new directory for the project
$ mkdir event-contact-management-2
$ cd event-contact-management-2
$ claude update
# start a new Claude Code session
$ claude
```

Code Snippet 1: *Starting a new Claude Code session for Requirements Analysis*

Notice in *Code Snippet 1* that I call `claude update` to ensure I have the latest version of Claude Code before starting a new session. When I execute the code, I eventually see something like so:



```
Claude Code v2.0.76
Sonnet 4.5 · Claude Pro
W:\OneDrive\repos\niels\event-conta

A gift for you
Your rate limits are 2x higher through 12/31. Er

> Try "write a test for <filepath>"

? for shortcuts
```

CATALOG
Plot Twist
Understanding What We …
The AI-Native Architectur…
Using Claude Code for R…
Conversation
Discovering Existing MC…
Choosing an Email Service
Alternative Email Services
System Architecture Des…
MCP Server Strategy
Technology Stack Summ…
The Development Workfl…
What Makes This AI-Nati…
Setting Up for Part 4
Conclusion: Why This M…
~ Finally

Figure 1: *Claude Code up and Running*

The reason I am showing you this is what is outlined in red in *Figure 1*. You can see the model and which subscription plan I am on. This is important because Claude Code's capabilities depend on the model you are using. Sonnet 4.5 is the default model, but for the upcoming discussion, I want to use the Opus 4.5 model because it has better reasoning capabilities, which are crucial for architectural discussions.

To change the model, I simply type `/model` in Claude's input textbox, press enter and select `Opus 4.5` from the list. Now I'm ready to discuss the architecture.

> ***NOTE:*** *If you don't see the model selection option, ensure your Claude Code installation is up to date and that your subscription plan supports multiple models. Also, be aware that the Opus model is more costly to use, so monitor your usage accordingly.*

Conversation

When "talking" to Claude Code, you get the best results by providing clear context and specific requirements. The naive approach would be to just say something like:

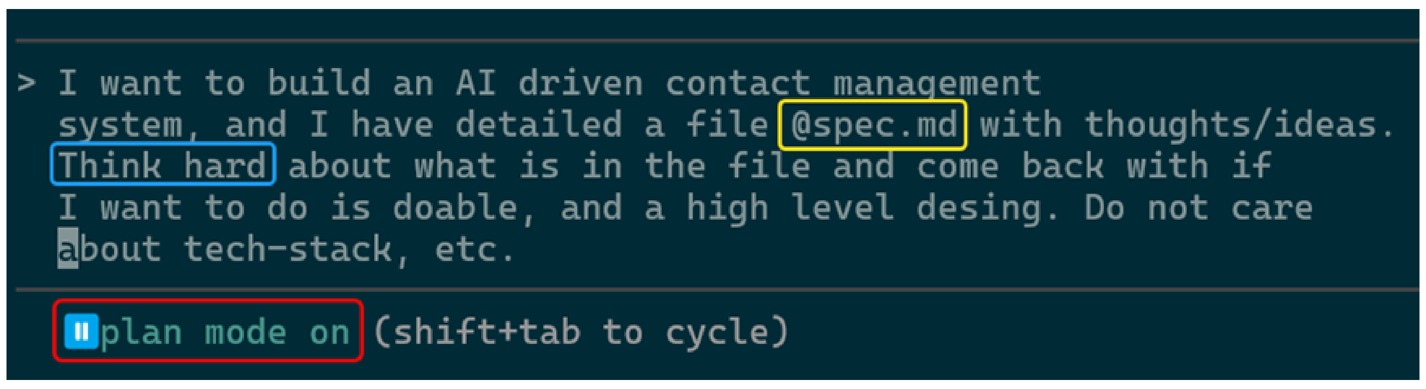
```
I need to build an event management system, but it's really more of a CRM and communications hub.

What I want is to be able to ask questions like "How many people attended last year?" and have the
```

Code Snippet 2: *Naive Prompt to Claude Code for Architectural Discussion*

Why do I say "naive"? I actually used that prompt initially 😄, and Claude Code's response was very generic 😞. So I altered my approach and tried to be very specific about what I wanted.

> 💡 **TIP**
>
> When using Claude Code for architectural discussions, always provide as much context as possible. Use external files for detailed specs, and encourage thoughtful analysis by prompting for step-by-step plans.

I created a `spec.md` (https://gist.github.com/nielsberglund/357e06005ae3ef43db6c2166557c836c) file in the project directory that outlines all the context and high-level requirements. Then I told Claude Code to read that file and help me design the architecture based on it:



```
> I want to build an AI driven contact management
  system, and I have detailed a file @spec.md with thoughts/ideas.
  Think hard about what is in the file and come back with if
  I want to do is doable, and a high level desing. Do not care
  about tech-stack, etc.

  ⏸ plan mode on (shift+tab to cycle)
```

Figure 2: *Claude Code Prompt*

You see in *Figure 2* my prompt to Claude Code. Notice how I:

- Refer to the `spec.md` file for context (outlined in yellow).
- Tell Claude Code to *Think hard*, which encourages Claude Code to engage in a more thoughtful analysis of the problem (I mentioned this in Part 2 (/post/2025-08-13-building-an-event-management-system-with-claude-code-part-2---ide-integration-and-advanced-features/)).
- Outlined in red, you see how I switched `plan mode on` (**shift+tab**) so that Claude Code can outline a step-by-step plan for the architecture.

When executing the prompt in *Figure 2*, Claude Code "goes away", it asks some clarifying questions, and then comes back with a plan:

- CATALOG
Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
Conversation
Discovering Existing MC...
Choosing an Email Service
Alternative Email Services
System Architecture Des...
MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

```
[snipped for brevity]

## Conclusion

**Your vision is not only feasible - it's architecturally superior for your use case.**

The LLM/MCP approach gives you:
- **Flexibility**: Ask any question, not just pre-built reports
- **Speed**: No frontend development for Phase 1
- **Intelligence**: Context-aware, conversational responses
- **Safety**: Preview/confirmation workflows
- **Extensibility**: Add capabilities without UI changes
- **Future-proof**: Same backend works with any frontend

**Recommended Next Step:** Start with Phase 1 (Claude.ai/Claude Code + MCP servers). This lets yo

The database schema and MCP servers you build for Phase 1 will work unchanged in Phase 2 - you're
```

Code Snippet 3: *Claude Code Plan*

In *Code Snippet 3*, you see the last part of the plan. You can see the complete plan here (https://gist.github.com/nielsberglund/c0b16d2a83b81956b0e1a59c957fc928). When you read the entire plan, you see that Claude Code has outlined a comprehensive architecture. It breaks down the data model, integration points, and query patterns. Is the plan 100%? No, but it's a fantastic starting point. I can now iterate on this plan, ask follow-up questions, and refine the architecture further.

> 💡 **TIP**
>
> You may ask yourself, "How did you get the plan, so it could be uploaded as a gist?" Good question! After Claude Code said that the plan we done I asked it to write the plan to a file: `plan.md`. Once that was done, I used the command line to create a gist from that file:
>
> ```
> $ gh gist create plan.md --public
> ```

This is powerful, despite the plan being nowhere near perfect. Because Claude Code isn't just writing code, it's helping you **think through the problem** before committing to an implementation. It's forcing you to articulate your requirements clearly, and it's suggesting architectural patterns you might not have considered.

At Derivco, we've found that spending time in this planning phase with Claude Code significantly speeds up development, because you catch design issues before they become code problems.

Coming back to the plan. When reading the plan, I noticed that Claude Code suggested building custom MCP servers for everything. Is that the best solution?

Discovering Existing MCP Servers

Here's where things get really practical. Before I build a single custom MCP server, I need to know what already exists. The MCP ecosystem is growing rapidly, and there's a good chance someone has already built tools for everyday operations, such as database access or email sending.

Let's use Claude Code to research what's available:

- CATALOG
Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
  Conversation
Discovering Existing MC...
Choosing an Email Service
  Alternative Email Services
System Architecture Des...
  MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

```
Me: In your plan you suggested to use MCP servers. Good! However it sounded you suggested to buil
  servers. Before we do that, can you please check if there are any existing PostgreSQL MCP serve
  what is the best one in your opinion.

Claude Code: [searches and finds several options]

Excellent question! I found several existing PostgreSQL MCP servers. Here's my analysis:

  Available PostgreSQL MCP Servers

  1. Postgres MCP Pro (Crystal DBA) ⭐ Best Option

  Repository: https://github.com/crystaldba/postgres-mcp

  Features:
  - ✅ Configurable read/write access (unrestricted for dev, restricted for production)
  - ✅ 8 comprehensive tools including:
    - execute_sql - Run queries with safety controls
    - list_schemas, list_objects, get_object_details - Schema exploration
    - explain_query - Query performance analysis
    - analyze_db_health - Database health monitoring
    - analyze_workload_indexes - Index recommendations
  - ✅ Production-ready with safety features
  - ✅ Actively maintained
  - ✅ Easy installation (Docker, pipx, or uv)

  2. Official @modelcontextprotocol/server-postgres (Anthropic)

  Status: ⚠️ ARCHIVED (May 2025)
  - Read-only access only
  - Basic schema inspection and queries
  - Still available but deprecated
  - 20,000 weekly NPM downloads (legacy usage)

  [snipped for brevity]

  My Verdict

  Start with Postgres MCP Pro directly (Option 1) to validate your architecture quickly. If you f

[snipped for brevity]
```

Code Snippet 4: *Claude Code Researching Existing PostgreSQL MCP Servers*

This discovery phase is critical. It's tempting to just start coding, but by finding existing MCP servers first, I can:

- **Save development time** - Why build what already exists?
- **Learn patterns** - Study how others have solved similar problems
- **Combine tools** - Use multiple MCP servers together for complex workflows

> ***NOTE:*** *After Claude Code came back with the above, I asked it to update the plan in* `plan.md` *and' CLAUDE.md' to reflect this new information about existing MCP servers.*

I then did the same for CSV/Excel import. In this case, Claude Code came back with a recommended hybrid option:

```
My Verdict

Go with Option 1: Use csv-mcp-server for foundation + FastMCP for custom Quicket/Sessionize impor

Why:
- 80% of functionality (validation, quality, duplicates) comes free from csv-mcp-server
- 20% custom code handles domain-specific formats
- Best balance of leverage vs. control
- FastMCP makes custom development fast and Pythonic

Implementation Plan:
1. Install csv-mcp-server for general CSV operations
2. Build custom "import-mcp-server" using FastMCP with 5-6 tools for Quicket/Sessionize
3. Custom server internally uses Pandas (same as csv-mcp-server) for CSV parsing
4. Custom server calls Postgres MCP Pro to insert validated data

Code Estimate: ~500-800 lines of Python for custom import server (FastMCP + Pandas + validation l

Would you like me to update CLAUDE.md and plan.md with these findings and recommendations?
```

Code Snippet 5: *Claude Code Researching Existing CSV/Excel MCP Servers*

It is interesting to see in *Code Snippet 5* how Claude Code now asks me if I also want to update `CLAUDE.md` and `plan.md` with these findings and recommendations (I said yes).

Choosing an Email Service

It would now be tempting to ask the same for email MCP servers. However, in the original plan, Claude Code mentioned SendGrid as an email service. I am not sure that is the best option, so I asked Claude to research the email service options a bit more:

CATALOG

Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
Conversation
Discovering Existing MC...
Choosing an Email Service
Alternative Email Services
System Architecture Des...
MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

```
Can you now research what email sending servers are out there. I do have some requirements:
1. I have the contact database. I do not want to be reliant on a database sitting with the servic
 schemas that I do not want.
2. It would be great if the service has a drag and drop template editor, alternatively email edit
Markdown
3. Cost is always a factor. I do not see me sending more than 6,000 emails per year, ~1,500 per q
```

Code Snippet 5: *Claude Code Researching Email Services*

The question in *Code Snippet 5* is quite specific about my requirements. When Claude Code comes back with the answer, it is somewhat surprising (remember, initially Claude Code mentioned SendGrid):

```
Final Recommendation

 Recommended Stack:

 1. Email Service: Brevo (FREE plan)
    └ Covers 109,500 emails/year (18x your need)
    └ Drag-and-drop template editor
    └ API-first, no contact DB required

 2. MCP Integration: Build custom Email MCP Server
    └ Use FastMCP framework (Python)
    └ Integrate with Brevo API
    └ Tools: preview_recipients, send_email, get_templates, etc.
    └ ~300-500 lines of code

 3. Template Workflow:
    Option A: Use Brevo's drag-and-drop editor (easiest)
    Option B: Create MJML templates, store in repo, send via API
```

Code Snippet 6: *Claude Code Recommended Email Services*

That was unexpected! Claude Code is now recommending Brevo (formerly Sendinblue) instead of SendGrid. The reason is that Brevo's free plan covers my email volume, has a drag-and-drop editor, and doesn't require me to manage a contact database on their side. This is a perfect fit for my needs (at least at first glance).

Alternative Email Services

Above, I said Claude's recommendation was unexpected. The reason is that, in the original plan, Claude Code suggested SendGrid without me asking. However, when I specifically asked for email services that met my requirements, it returned Brevo instead. I am still not 100% that this is the best option, so I wanted to get a second option. I decided to ask Gemini Pro via the web interface. I asked the same question as in *Code Snippet 5* and I got the same answer: Brevo. So both Claude Code and Gemini Pro agree on this. However, it also has a new option: MailGun, which then led me to a third option: MailJet.

Since we are now going truly down the rabbit hole, I decided to as Claude Code to compare the three options: Brevo, MailGun, and MailJet. Here is the comparison I got back:

```
| Feature                    | Brevo            | MailJet                   | Mailgun
|----------------------------|------------------|---------------------------|----------------------
| Free Tier (emails/year)    | 109,500          | 72,000                    | 36,000
| Free Tier Cost             | $0               | $0                        | $0
| Actual Cost for Your Use   | ✅ $0             | ❌ $25/month ($300/year)  | ❌ $15/month ($180/y
| Database Independence      | ✅ Yes            | ✅ Yes                    | ⚠️ Yes (paid only)
| Drag-and-Drop Editor       | ✅ Yes            | ✅ Yes                    | ❌ No
| MJML Support               | ⚠️ Import only    | ✅ Native                 | ❌ No
| Transactional API (Free)   | ✅ Yes            | ❌ No (Premium only)      | ⚠️ 5 recipients max
| Unrestricted Sending       | ✅ Yes            | ⚠️ Yes (Essential+)       | ❌ No (Foundation+)
| Marketing + Transactional  | ✅ Both           | ✅ Both                   | ❌ Transactional onl
| Ease of Use                | ⭐⭐⭐⭐⭐           | ⭐⭐⭐⭐                      | ⭐⭐ (dev-only)
```

Code Snippet 7: *Claude Code Email Service Comparison*

So, according to *Code Snippet 7*, Brevo is the best option for my use case. It offers the highest free tier, supports both marketing and transactional emails, includes a drag-and-drop editor, and doesn't require a contact database on its end. MailJet is a close second, but it costs $25/month for my usage. MailGun falls short in several areas, especially since I want both marketing and transactional emails. The final question then is whether there are existing MCP servers for Brevo. Asking Claude Code again, I get the answer that there are a couple of existing Brevo MCP servers, and it recommended one: `houtini-ai/brevo-mcp`.

> 💡 **TIP**
>
> Please remember that Claude Code (or any other coding agent) is not infallible. If you are uneasy about something it suggests, do your own research. The research, as in this case, can be as simple as asking another LLM (like Gemini Pro) for a second opinion.

Based on all we have discussed so far, we should now be ready to look at the actual architecture.

- CATALOG

Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
  Conversation
Discovering Existing MC...
Choosing an Email Service
  Alternative Email Services
System Architecture Des...
  MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

- CATALOG
Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
Conversation
Discovering Existing MC...
Choosing an Email Service
Alternative Email Services
System Architecture Des...
MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

System Architecture Design & Strategy

Claude Code did a great job outlining the architecture in the plan. However, now that we have more information about existing MCP servers and services, we can refine that architecture further, so we ask Claude Code:

```
Both in the @CLAUDE.md and @plan.md you have an architecture diagram. That diagram was based in i
```

Code Snippet 8: *Claude Code Updating Architecture Diagram*

After having done that, I asked Claude Code to generate the diagram in Excalidraw format, which I then exported as a PNG and included below:

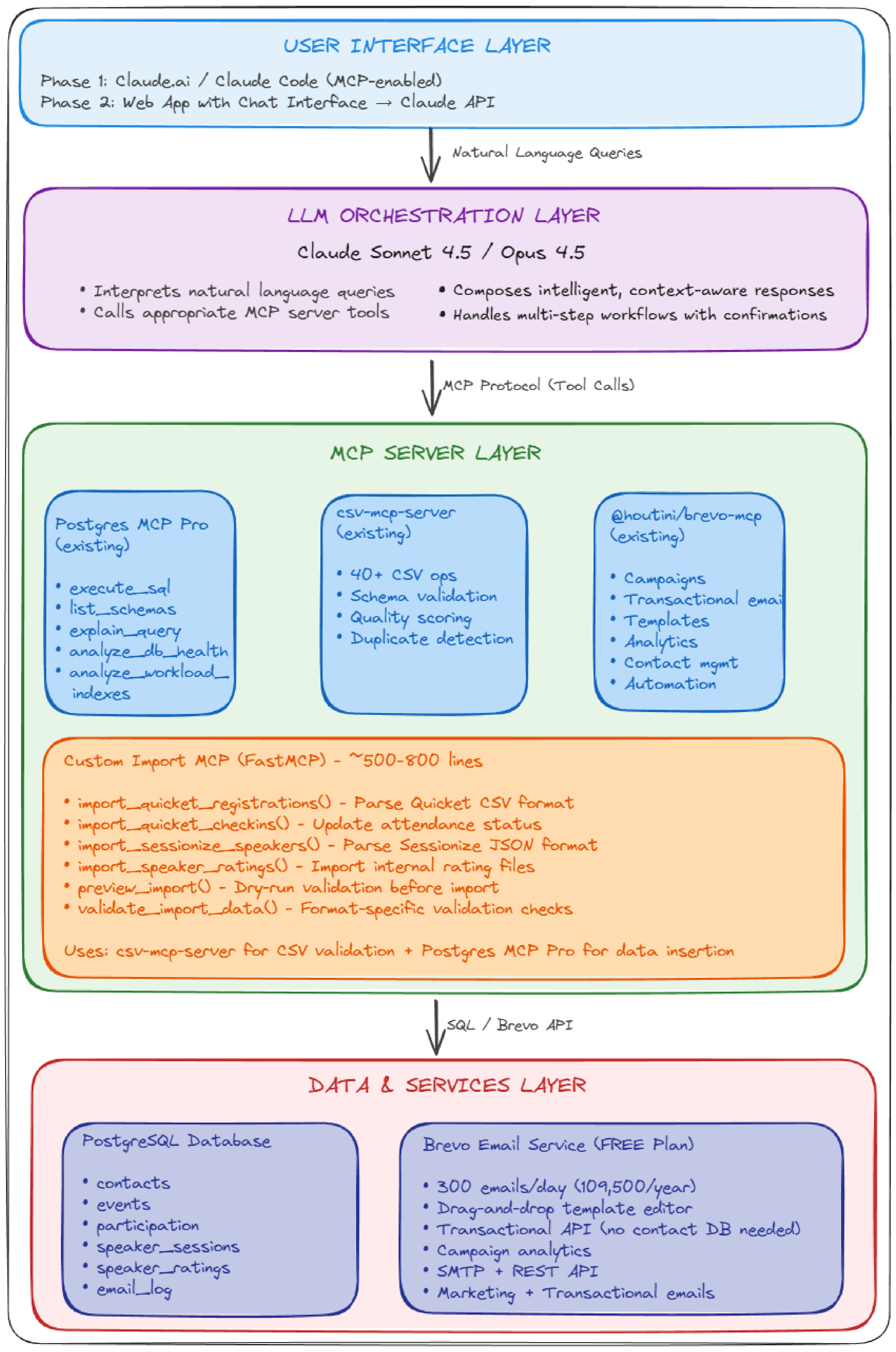

USER INTERFACE LAYER
Phase 1: Claude.ai / Claude Code (MCP-enabled)
Phase 2: Web App with Chat Interface → Claude API
Natural Language Queries
LLM ORCHESTRATION LAYER
Claude Sonnet 4.5 / Opus 4.5
• Interprets natural language queries
• Calls appropriate MCP server tools
• Composes intelligent, context-aware responses
• Handles multi-step workflows with confirmations
MCP Protocol (Tool Calls)
MCP SERVER LAYER
Postgres MCP Pro (existing)
• execute_sql
• list_schemas
• explain_query
• analyze_db_health
• analyze_workload_indexes
csv-mcp-server (existing)
• 40+ CSV ops
• Schema validation
• Quality scoring
• Duplicate detection
@houtini/brevo-mcp (existing)
• Campaigns
• Transactional emai
• Templates
• Analytics
• Contact mgmt
• Automation
Custom Import MCP (FastMCP) - ~500-800 lines
• import_quicket_registrations() - Parse Quicket CSV format
• import_quicket_checkins() - Update attendance status
• import_sessionize_speakers() - Parse Sessionize JSON format
• import_speaker_ratings() - Import internal rating files
• preview_import() - Dry-run validation before import
• validate_import_data() - Format-specific validation checks
Uses: csv-mcp-server for CSV validation + Postgres MCP Pro for data insertion
SQL / Brevo API
DATA & SERVICES LAYER
PostgreSQL Database
• contacts
• events
• participation
• speaker_sessions
• speaker_ratings
• email_log
Brevo Email Service (FREE Plan)
• 300 emails/day (109,500/year)
• Drag-and-drop template editor
• Transactional API (no contact DB needed)
• Campaign analytics
• SMTP + REST API
• Marketing + Transactional emails

Figure 3: *Event Contact Management System Architecture*

> ***NOTE:*** *I had to slightly modify the diagram in Excalidraw to get the layout right, but the content is exactly what Claude Code generated.*

MCP Server Strategy

Based on our research, here's the practical strategy:

1. PostgreSQL MCP Server (Use Existing: Postgres MCP Pro)

- Handles all database queries
- Natural language → SQL translation
- Analytics and reporting
- Data insertion for imports
- Potentially pre-created queries for common reports

2. Import MCP Server (Use Existing: csv-mcp-server + Build Custom)

- Specific to Quicket and Sessionize export formats
- Data validation and transformation
- Duplicate detection
- Batch upload orchestration

3. Brevo MCP Server (Use Existing: @houtini/brevo-mcp)

- Email campaign management
- Contact list operations
- Email templates
- Send to filtered contact groups

The beauty of this approach: I only need to build a thin, custom Import MCP server that the Import MCP server calls. This thin layer handles Quicket/Sessionize-specific format parsing. Everything else already exists.

Technology Stack Summary

We are nearing the end of this post. Let's consolidate our decisions:

Database: PostgreSQL (most likely the latest version)

- Robust, mature, excellent for relational data
- Strong community support
- Perfect for our contact and participation tracking needs

MCP Servers:

1. **Postgres MCP Pro** - Database access and querying
2. **csv-mcp-server + custom import MCP Server** - Quicket/Sessionize imports
3. **@houtini/brevo-mcp** - Email operations

Conversational Interface: Claude.ai (initially)

- Later can expand to Claude Code for more integrated workflows
- Potential future: Custom chatbot interface

Data Import Format: CSV/Excel exports from Quicket and Sessionize

- Batch upload via custom MCP server
- Data validation and duplicate detection
- Transform to match our schema

Email Delivery: Brevo

- Contacts pulled from our PostgreSQL database
- Templates and campaigns managed via @houtini/brevo-mcp MCP server
- Delivery and analytics through Brevo API

The Development Workflow

Here's how development will actually work with this architecture:

Phase 1: Database Setup (Part 4)

1. Set up a PostgreSQL database
2. Create schema (contacts, events, event_participation, communications)
3. Install and configure Postgres MCP Pro
4. Test natural language queries with Claude

Phase 2: Data Import (Part 5)

1. Build custom Import MCP server
2. Handle Quicket CSV format (registrations, check-ins)
3. Handle Sessionize CSV format (speakers, sessions)
4. Test batch imports with real exported data

Phase 3: Email Integration (Part 6)

1. Install and configure the Brevo MCP server
2. Create email templates
3. Implement filtered contact list
4. Test email workflows (speaker invitations, attendee communications)

CATALOG

Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
  Conversation
Discovering Existing MC...
Choosing an Email Service
  Alternative Email Services
System Architecture Des...
  MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

Phase 4: Real-World Usage (Part 7)

1. Import actual Data & AI Community Day data
2. Document common query patterns
3. Create email campaigns
4. Generate participation reports and analytics

What Makes This AI-Native?

Let me be explicit about what makes this system different from a traditional web application:

Traditional Approach:

- Build admin dashboard with forms
- Create pages for contacts, events, reports
- Write custom queries for each report type
- Build email composer interface
- Manual data import screens

AI-Native Approach:

- Ask questions in natural language
- Claude translates to SQL via MCP server
- Email operations through conversational commands
- Data import described in conversation, not forms
- Reports generated on-demand by describing what you want
- New functionality can be implemented in the MCP layer. Not touching the client.

The system has no “user interface” in the traditional sense; the interface IS the conversation. This is only possible because of MCP servers, bridging the gap between natural language and system operations.

Setting Up for Part 4

In the next post, we’ll get hands-on. We’ll:

1. Set up a PostgreSQL database (I’ll show both local and cloud options)
2. Create our schema with proper indexes
3. Install and configure Postgres MCP Pro
4. Have our first conversations with the database
5. Test complex queries like “Show me speaker retention across years”

We’ll start entering data into the system and see how natural-language database access works in practice. No more theory; it’s time to build.

Conclusion: Why This Matters

Look, I could have built a traditional CRUD application with a web interface. It would work fine. But here’s what I’ve learned at Derivco as we’ve gone all-in on AI-assisted development: **the best use of AI isn’t just writing code faster; it’s rethinking how systems should work in an AI-first world.**

If I’m going to spend my time building this system, I want it to be genuinely useful, not just another admin dashboard. I want to be able to pull up Claude.ai on my phone at a conference and ask, “How many people checked in so far?” without logging in. I want to say “Email all speakers their session times” while I’m having coffee, not fill out forms.

That’s the promise of AI-native systems, and that’s what we’re building here. Not because it’s trendy, but because it’s actually better for how I work.

And here’s the thing about Claude Code that this post has hopefully demonstrated: it’s not just a faster way to write code; it’s a thinking partner for architecture and planning. Throughout this entire post, Claude Code helped me research existing MCP servers, compare email service options, validate architectural decisions, and even push back when I wasn’t being specific enough in my requirements.

At Derivco, we’ve discovered that the real productivity gain from AI-assisted coding isn’t the speed of writing code; it’s the quality of decisions made before any code is written. Claude Code forced me to articulate my requirements clearly, suggested patterns I hadn’t considered,

- CATALOG
Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
Conversation
Discovering Existing MC...
Choosing an Email Service
Alternative Email Services
System Architecture Des...
MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

and helped me discover existing solutions rather than reinventing the wheel. The plan we developed together isn't perfect. Still, it's dramatically better than what I would have created alone, and we did it in hours instead of weeks. This collaborative planning approach is why I'm confident we're building the right system, not just building a system right.

In Part 4, we'll make this real. We'll install PostgreSQL, set up the database, configure our first MCP server, and start having conversations with our data. If you've been following this series from the beginning, you'll see how far we've come, from installing Claude Code to architecting a genuinely innovative system.

The irony of being too busy to write about event management while managing events? Still there. But now I'm building something that might help with that.

See you in Part 4!

~ Finally

That's all for now. I hope you find this information valuable. Please share your thoughts and ideas on this post, or ping (mailto:niels.it.berglund@gmail.com) me if you have any suggestions. Your input is highly valued and can help shape the direction of our discussions. If you found this post helpful, please share it with your network. Follow me on LinkedIn (https://www.linkedin.com/in/nielsberglund/) for more updates on this project and other AI-related topics.

← **PREVIOUS POST (/DRAFTS/BUILDING-AN-EVENT-MANAGEMENT-SYSTEM-WITH-CLAUDE-CODE-PART-3---SYSTEM-PLANNING-AND-ARCHITECTURE-DESIGN/)**

Disqus comments not available by default when the website is previewed locally.
comments powered by Disqus (https://disqus.com)

- CATALOG

Plot Twist
Understanding What We ...
The AI-Native Architectur...
Using Claude Code for R...
  Conversation
Discovering Existing MC...
Choosing an Email Service
  Alternative Email Services
System Architecture Des...
  MCP Server Strategy
Technology Stack Summ...
The Development Workfl...
What Makes This AI-Nati...
Setting Up for Part 4
Conclusion: Why This M...
~ Finally

● (mailto:niels.it.berglund@gmail.com)
● (https://twitter.com/nielsberglund)
● (https://github.com/nielsberglund)
● (https://www.linkedin.com/in/niels-berglund-0122593)
● (https://medium.com/@nielsberglund)
● (https://stackoverflow.com/users/7656880/niels-berglund)
● (https://fosstodon.org/@nielsb)
● (https://nielsberglund.com/index.xml)

Copyright © Niels Berglund 2025
CleanWhite Hugo Theme (https://themes.gohugo.io/hugo-theme-cleanwhite) by Huabing (https://zhaohuabing.com) | Star 802