Tutorial 34: Google Cloud Pub/Sub + Event-Driven Agents

Difficulty: advanced
Reading Time: 1 hour
Tags: cloud, pubsub, event-driven, python, agents
Description: Build event-driven document processing pipelines with Google Cloud Pub/Sub and ADK agents for asynchronous processing.

This tutorial implements a real event-driven document processing system using Google Cloud Pub/Sub and ADK agents. It demonstrates a coordinator + specialist agents pattern with structured JSON output using Pydantic models.
Verified as of October 2025 with latest ADK and Gemini 2.5 Flash.

Estimated Reading Time: 50-60 minutes
Difficulty Level: Advanced
Prerequisites: Tutorial 01-03 (ADK Basics), Google Cloud project

---

🚀 Quick Start - Working Implementation

The easiest way to get started is with our **complete working implementation**:

```
cd tutorial_implementation/tutorial34
make setup      # Install dependencies
make test       # Run all tests
```

📁 View Full Implementation (../../tutorial_implementation/tutorial34)

What's included:

- ✔️ `root_agent`: Coordinator agent that routes documents to specialists
- ✔️ 4 Specialist agents: Financial, Technical, Sales, Marketing analyzers
- ✔️ Pydantic output schemas: Structured JSON results
- ✔️ 66 comprehensive tests (all passing)
- ✔️ Real-world example code ready to run

Table of Contents

Overview

What You'll Build

In this tutorial, you'll build an **event-driven document processing system** using:

- **Google Cloud Pub/Sub** (Event messaging)
- **Google ADK** (Multi-agent coordination)
- **Gemini 2.5 Flash** (Document analysis)
- **Pydantic Models** (Structured JSON output)

Architecture:

```
┌─────────────────────────────────────────────────────┐
│  Publisher: Sends documents to Pub/Sub              │
└────────────────────┬────────────────────────────────┘
                     │
         ┌───────────▼────────────┐
         │  Google Cloud Pub/Sub  │
         │  (document-uploads)    │
         └───────────┬────────────┘
                     │
         ┌───────────▼────────────┐
         │  root_agent (Coordinator)
         │  - Routes documents    │
         │  - Coordinates analysis│
         └───────────┬────────────┘
                     │
         ┌──┬──┬──┬──┘
         │  │  │  │
    ┌────▼┐ │ ┌┴────┬─────────┐
    │Fin. │ │ │Tech │ Sales  Marketing
    │Anal.│ │ │Anal.│Analyst Analyst
    └─────┘ │ └─────┴────────┘
```

Why Pub/Sub + ADK?

Feature	Benefit
Asynchronous	Non-blocking processing
Decoupled	Publishers and subscribers independent
Scalable	Auto-scales message volume
Structured	Pydantic models for JSON
Reliable	At-least-once delivery, retries

When to use Pub/Sub + ADK:

✔️ Asynchronous document processing
✔️ Multi-step workflows
✔️ Event-driven architectures
✔️ Systems with strict output schemas
✔️ Google Cloud deployments

❌ Real-time chat interfaces → Use Next.js/WebSocket
❌ Simple synchronous calls → Use direct API

Prerequisites & Setup

Local Testing (No GCP Required)

To get started without Google Cloud:

```
# Install dependencies
cd tutorial_implementation/tutorial34
make setup

# Run tests - verifies agent configuration
make test

# This works completely locally using in-memory processing
```

Google Cloud Setup (Optional - For Real Pub/Sub)

To deploy with real Google Cloud Pub/Sub:

1. Install gcloud CLI

```
# macOS
brew install --cask google-cloud-sdk

# Then initialize
gcloud init
```

2. Authenticate

```
# Login to Google Cloud
gcloud auth login

# Set default project
gcloud config set project your-project-id

# Verify authentication
gcloud auth list
```

3. Create Pub/Sub Resources

```
# Enable Pub/Sub API
gcloud services enable pubsub.googleapis.com

# Create topic
gcloud pubsub topics create document-uploads

# Create subscription
gcloud pubsub subscriptions create document-processor \
  --topic=document-uploads \
  --ack-deadline=600
```

4. Set Environment Variables

```
# Set your GCP project

# Set Gemini API key

# Set application credentials
gcloud auth application-default login
```

Understanding the Architecture

The Coordinator + Specialist Pattern

This implementation uses a **coordinator agent** that intelligently routes documents to specialized analyzers:

```
┌──────────────────────────────────────────────────────┐
│  root_agent (Coordinator)                            │
│  - Analyzes document type                            │
│  - Routes to appropriate analyzer                    │
│  - Coordinates specialized agents                    │
└───────┬──────────────────────────────────────────────┘
        │
    ┌───┴───┬─────────────┬──────────────┐
    │       │             │              │
┌───▼──┐ ┌──▼───┐ ┌───────▼───┐   ┌──────▼──┐
│Finan.│ │Tech. │ │Sales      │   │Marketing
│Anal. │ │Anal. │ │Analyst    │   │Analyst
└──────┘ └──────┘ └───────────┘   └─────────┘
    │        │             │              │
    └────────┴─────────────┴──────────────┘
              │
     Structured JSON Output
     (Pydantic Models)
```

Key Components

1. **root_agent** (`pubsub_agent/agent.py`):
2. Coordinator that routes documents to specialists
3. Analyzes document type and content
4. Calls appropriate sub-agent tool
5. Returns structured analysis
6. **Sub-Agents** (financial, technical, sales, marketing):
7. Specialized analyzers for document types
8. Enforce structured JSON via Pydantic output_schema
9. Extract type-specific metrics and insights
10. **Pydantic Output Schemas**:
11. `FinancialAnalysisOutput` : Revenue, profit, metrics
12. `TechnicalAnalysisOutput` : Technologies, components
13. `SalesAnalysisOutput` : Deals, pipeline value
14. `MarketingAnalysisOutput` : Campaigns, engagement metrics

Pub/Sub Guarantees

Feature	Description
At-least-once	Messages delivered ≥1 time
Asynchronous	Non-blocking processing
Scalable	Auto-scales message volume
Durable	Messages stored in topics
Reliable	Automatic retries on failure

Core Components

Agent Configuration

View the agent at `pubsub_agent/agent.py`:

```python
# Coordinator agent
root_agent = LlmAgent(
    name="pubsub_processor",
    model="gemini-2.5-flash",
    description="Event-driven document processing coordinator",
    instruction="Routes documents to specialized analyzers",
    tools=[financial_tool, technical_tool, sales_tool, marketing_tool],
)

# Sub-agents (financial, technical, sales, marketing)
# Each configured with output_schema for structured JSON
```

Output Schemas

All sub-agents return structured Pydantic models:

```python
# Financial documents return:
FinancialAnalysisOutput(
    summary: DocumentSummary,
    entities: EntityExtraction,
    financial_metrics: FinancialMetrics,
    fiscal_periods: list[str],
    recommendations: list[str]
)

# Technical documents return:
TechnicalAnalysisOutput(
    summary: DocumentSummary,
    entities: EntityExtraction,
    technologies: list[str],
    components: list[str],
    recommendations: list[str]
)

# Similar for Sales and Marketing analyzers
```

Example Usage

Locally without GCP:

```
cd tutorial_implementation/tutorial34
make test
```

Test the agent in code:

```
import asyncio
from google.adk import Runner
from google.adk.sessions import InMemorySessionService
from google.genai import types
from pubsub_agent.agent import root_agent

async def test_document_analysis():
    session_service = InMemorySessionService()
    runner = Runner(
        app_name="document_analyzer",
        agent=root_agent,
        session_service=session_service
    )

    session = await session_service.create_session(
        app_name="document_analyzer",
        user_id="test_user"
    )

    prompt = types.Content(
        role="user",
        parts=[types.Part(
            text="Analyze: Revenue $1.2M, Profit 33%, Q4 2024"
        )]
    )

    async for event in runner.run_async(
        user_id="test_user",
        session_id=session.id,
        new_message=prompt
    ):
        print("Response:", event)

asyncio.run(test_document_analysis())
```

Using ADK Web Interface:

```
adk web
```

Then visit `http://localhost:8000` and select `pubsub_processor` from the agent dropdown.

Running Locally

Without Pub/Sub (Local Testing)

```
cd tutorial_implementation/tutorial34

# Run all tests
make test

# See test coverage
make test-cov
```

Tests validate:
- Agent configuration
- Sub-agent setup
- Pydantic output schemas
- Agent imports and structure

With Pub/Sub (Google Cloud)

After setting up GCP (see Prerequisites), run publisher and subscriber:

Terminal 1 - Start subscriber:

```
python subscriber.py
```

Terminal 2 - Publish documents:

```
python publisher.py
```

The subscriber will process each document with the coordinator agent.

Google Cloud Deployment

Step 1: Set Up Pub/Sub Resources

```
gcloud pubsub topics create document-uploads
gcloud pubsub subscriptions create document-processor \
  --topic=document-uploads \
  --ack-deadline=600
```

Step 2: Run Subscriber

```
python subscriber.py
```

Step 3: Publish Documents

```
python publisher.py
```

The subscriber will automatically process each Pub/Sub message using the coordinator agent.

Troubleshooting

Common Issues

Issue 1: gcloud command not found

Cause: Google Cloud CLI not installed

Solution:

```
# macOS
brew install --cask google-cloud-sdk

# After installation, verify
gcloud --version
```

Issue 2: Agent not found when running locally

Cause: Agent module not properly installed

Solution:

```
cd tutorial_implementation/tutorial34

# Install in development mode
pip install -e .

# Verify agent imports
python -c "from pubsub_agent.agent import root_agent; print(root_agent.name)"
```

Issue 3: Tests fail with import errors

Cause: Dependencies not installed

Solution:

```
cd tutorial_implementation/tutorial34

# Install dependencies
make setup

# Or manually
pip install -r requirements.txt

# Run tests
make test
```

Issue 4: Messages Not Delivered on Pub/Sub

Cause: Subscription not receiving published messages

Solution:

```
# Verify subscription exists
gcloud pubsub subscriptions list

# Check subscription details
gcloud pubsub subscriptions describe document-processor

# Manually pull a message to test
gcloud pubsub subscriptions pull document-processor --limit=1

# Check IAM permissions
gcloud pubsub subscriptions get-iam-policy document-processor
```

Issue 5: Pub/Sub Authentication Error

Error: `DefaultCredentialsError: Could not automatically determine credentials`

Solution:

```
# Set up application default credentials
gcloud auth application-default login

# Or set explicit credentials

# Verify setup
gcloud auth list
```

Issue 6: Tests fail with "GOOGLE_API_KEY not set"

Cause: Gemini API key not configured

Solution:

```
# Set your Gemini API key

# Verify it's set
echo $GOOGLE_API_KEY

# Run tests again
make test
```

Issue 7: Agent processes documents but returns empty results

Cause: Model not returning expected output format

Solution:

- Verify GOOGLE_API_KEY is set and valid
- Check that the document content is clear and valid
- Review agent instructions in `pubsub_agent/agent.py`
- Test with a simple document first

```python
# Test the agent directly
import asyncio
from google.adk import Runner
from google.adk.sessions import InMemorySessionService
from google.genai import types
from pubsub_agent.agent import root_agent

async def test():
    session_service = InMemorySessionService()
    runner = Runner(
        app_name="test",
        agent=root_agent,
        session_service=session_service
    )
    session = await session_service.create_session(
        app_name="test",
        user_id="test"
    )
    message = types.Content(
        role="user",
        parts=[types.Part(text="Revenue $1M, Profit 30%")]
    )
    async for event in runner.run_async(
        user_id="test",
        session_id=session.id,
        new_message=message
    ):
        print(event)

asyncio.run(test())
```

Next Steps

You've Mastered Event-Driven Agents with Pub/Sub! 🎉

You now know how to:

✓ Build multi-agent coordinator systems
✓ Use Pydantic for structured JSON output
✓ Implement async agent processing
✓ Route documents to specialized analyzers
✓ Use Google Cloud Pub/Sub for event-driven processing
✓ Test agents locally without GCP
✓ Deploy to production with Pub/Sub integration

Key Patterns Learned

- **Coordinator + Specialist**: One agent routes to many specialized agents
- **Structured Output**: Pydantic models enforce JSON schemas
- **Async Processing**: Non-blocking document analysis
- **Event-Driven**: Pub/Sub handles message buffering and retries
- **Tool Composition**: Sub-agents as tools within coordinator

Continue Learning

Tutorial 29: UI Integration Overview
Compare all integration approaches (Next.js, Vite, Streamlit, etc.)

Tutorial 30: Next.js + CopilotKit Integration
Build real-time chat interfaces with React

Tutorial 35+: Advanced Patterns
Master deployment, scaling, and production optimization

Additional Resources

- Google Cloud Pub/Sub Documentation (https://cloud.google.com/pubsub/docs)
- ADK Documentation (https://google.github.io/adk-docs/)
- Pydantic Documentation (https://docs.pydantic.dev/)
- Gemini API Reference (https://ai.google.dev/docs)

🎉 **Tutorial 34 Complete!**

You've successfully built an event-driven document processing system with a multi-agent coordinator architecture. This pattern scales to millions of documents while maintaining structured, validated output.

Questions or feedback? Open an issue on the ADK Training Repository (https://github.com/raphaelmansuy/adk_training).

Generated on 2025-10-19 17:57:28 from 34_pubsub_adk_integration.md

Source: Google ADK Training Hub