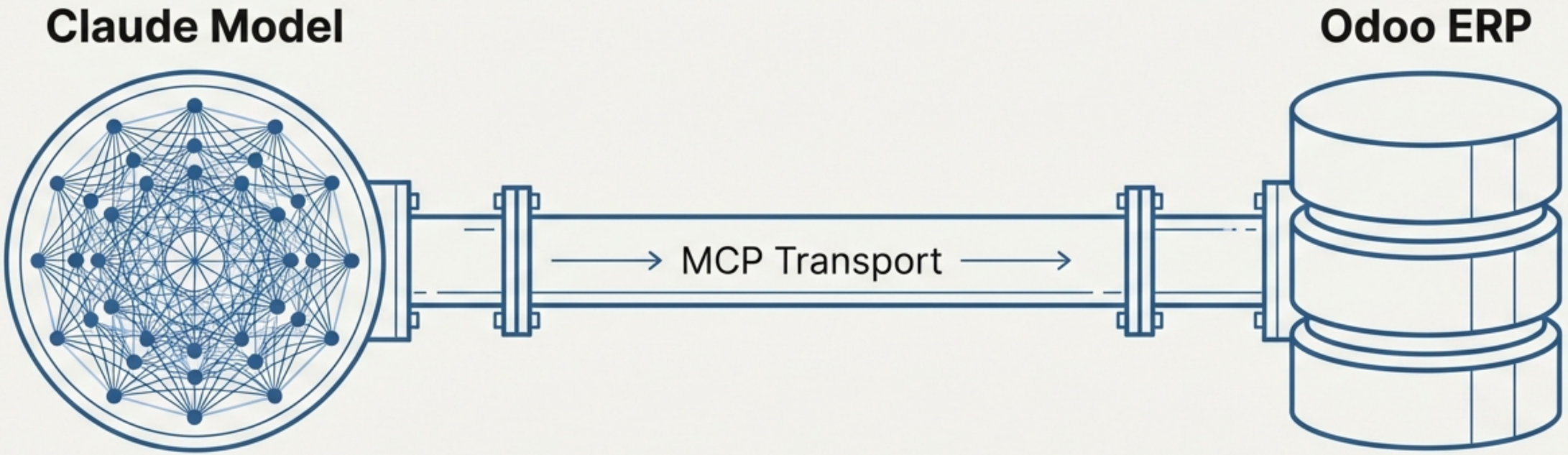
# MCP Python SDK + Odoo ERP

## A Guide to Production-Grade Automation

Implementing the Model Context Protocol for Business Outcome Optimisation.



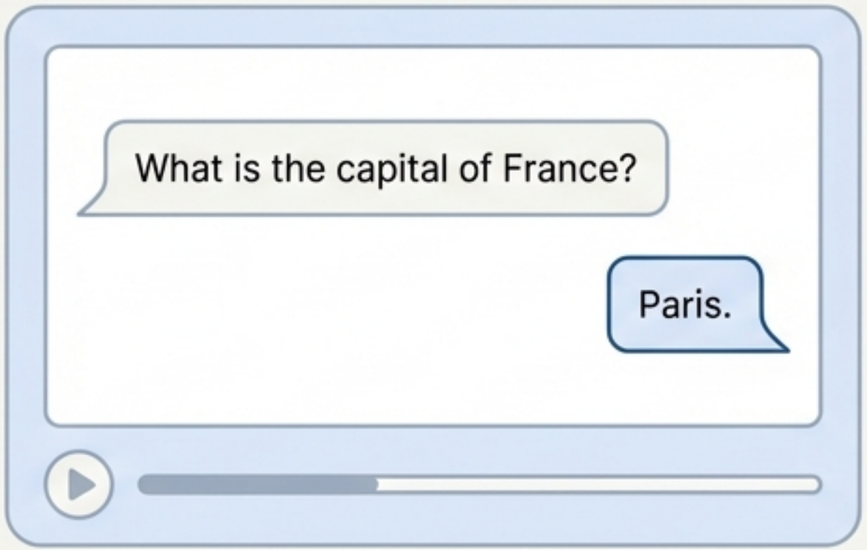

Tech Stack: `Python 3.x` | `FastMCP` | `Odoo XML-RPC` | `uv package manager`

NotebookLM

# Moving Beyond the “Demo Layer”

Most AI stops at the demo layer—generic Q&A **without system impact.** This blueprint is for **shipping code that connects to real operations:** reading live orders, creating purchase requisitions, and managing inventory.

## The Hype: Passive Retrieval




## The Goal: Active Operations

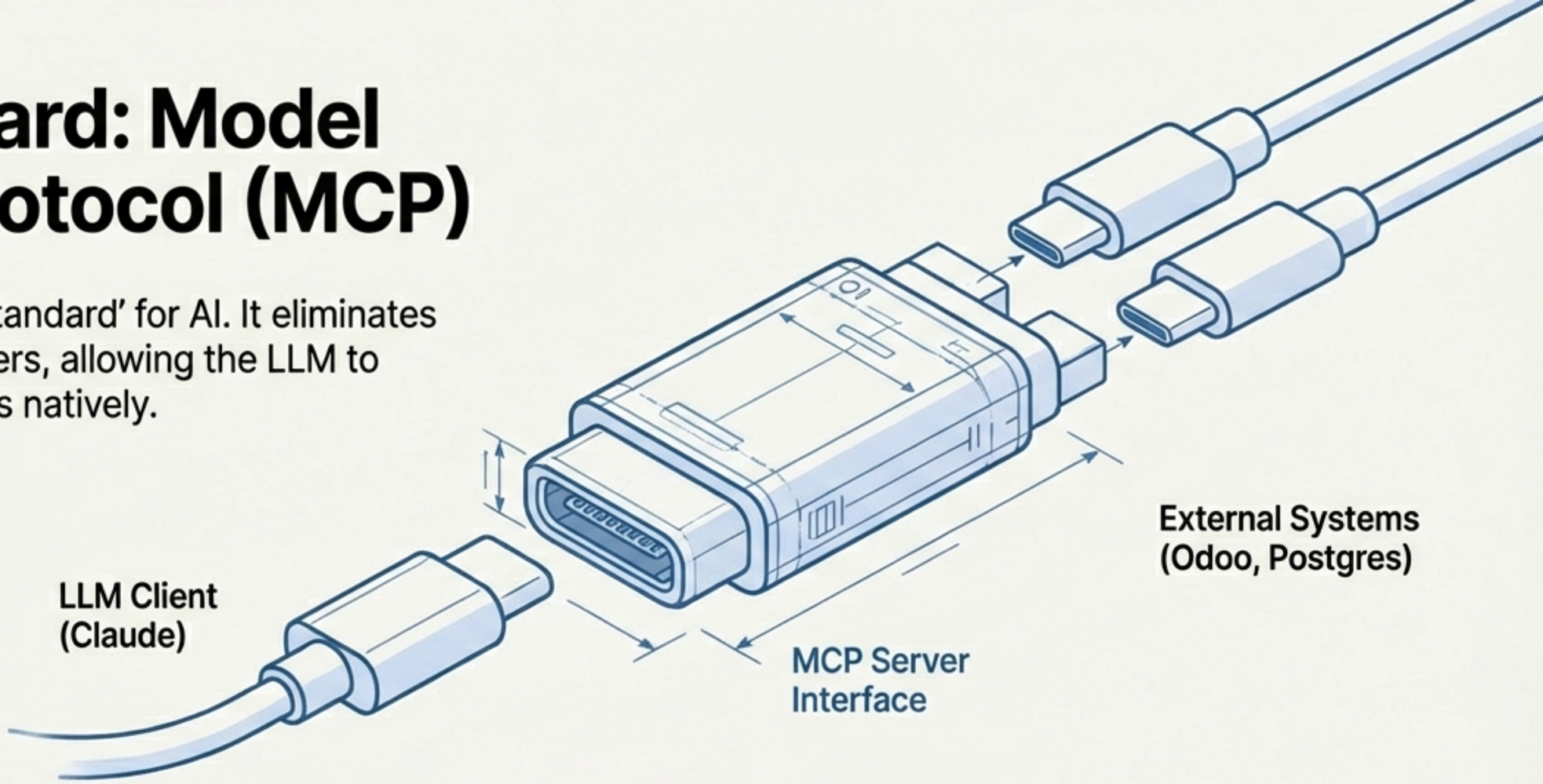
```
[INFO] Connection Established: Odoo XML-RPC
[INFO] Tool Call: check_stock(product='MacBook Pro')
[INFO] Inventory Check: 4 Units Available
[SUCCESS] Tool Call: create_requisition(qty=10)
[SUCCESS] PO #4459 Confirmed
[SUCCESS] PO #4459 Confirmed
```

*“This is about actually shipping something... in a way that you would not be embarrassed to put in front of a client.”*

NotebookLM

# The Standard: Model Context Protocol (MCP)

MCP acts as the 'USB standard' for AI. It eliminates bespoke integration layers, allowing the LLM to discover and utilise tools natively.




### Resources

Data the model reads (GET endpoints).
e.g., Sales Orders, Customer Lists.

### Tools

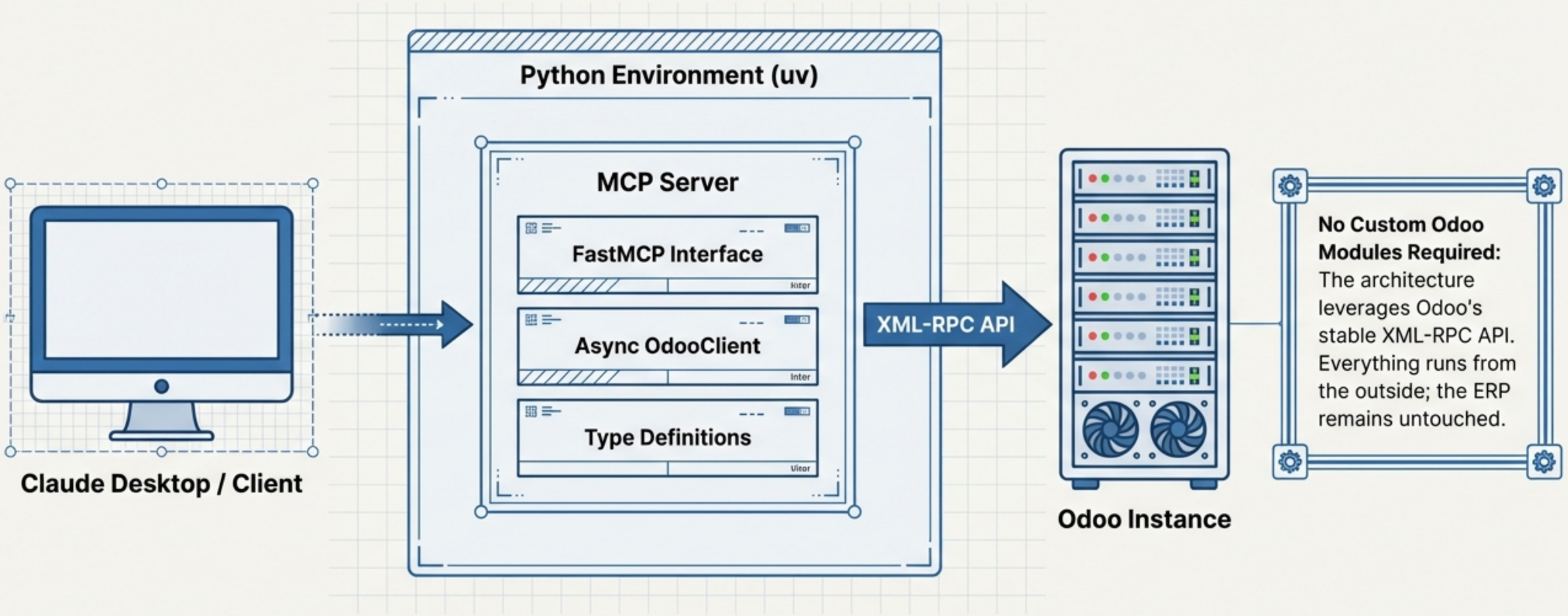
Functions the model calls (POST endpoints).
e.g., Create Invoice, Confirm Delivery.

### Prompts

Reusable interaction templates.

NotebookLM

# System Architecture & Data Flow

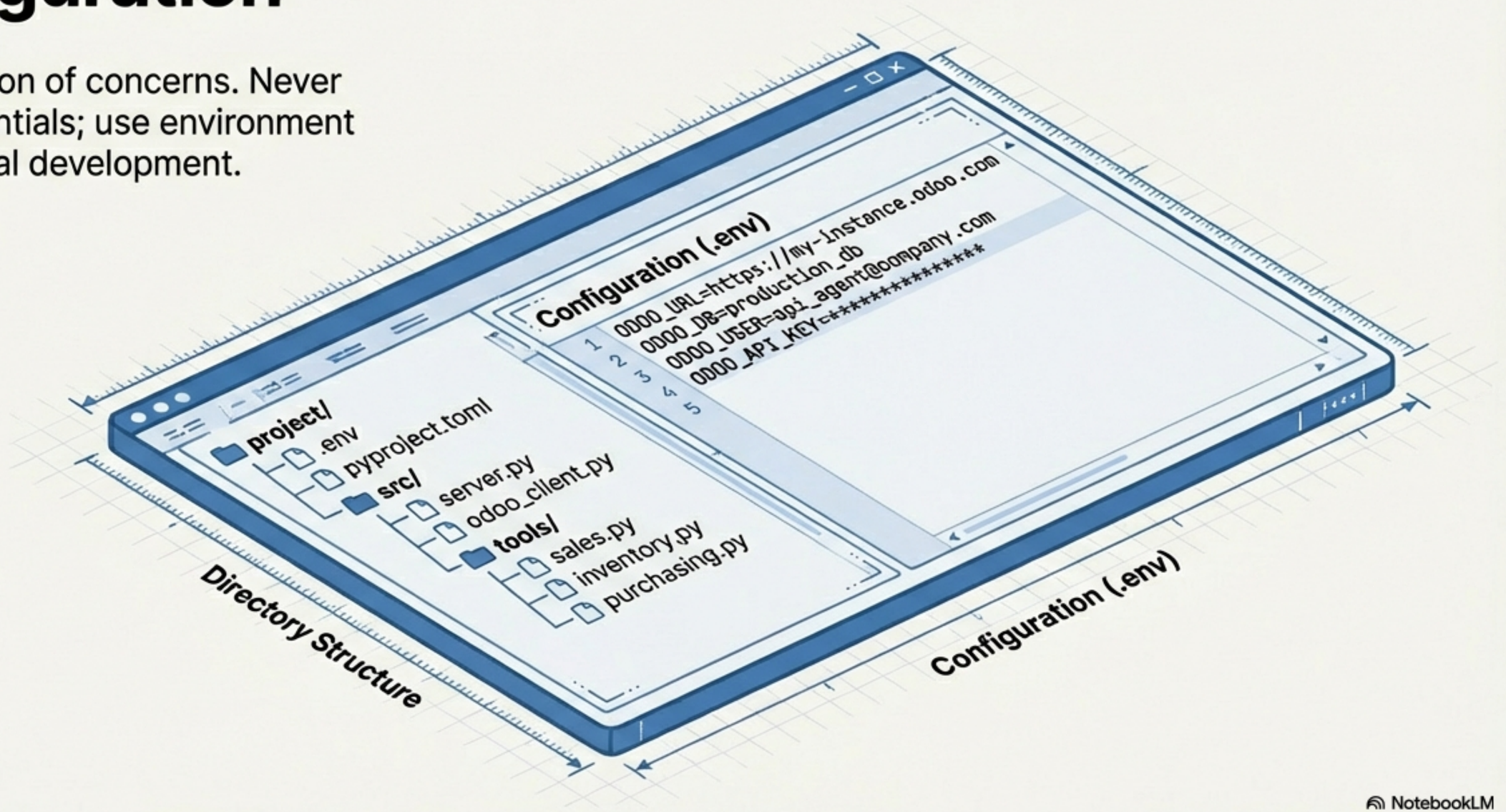



Tech Note: Uses official modelcontextprotocol Python SDK (v1.26.0+).

NotebookLM

# The Foundation: Project Setup & Configuration

A clean separation of concerns. Never hardcode credentials; use environment variables for local development.




NotebookLM

Engineering Editorial

# The Connector: Async Odoo XML-RPC Client

The client wraps the Odoo API to handle authentication, connection pooling, and transport errors centrally.

src/odoo_client.py

```python
class OdooClient:
    def __init__(self, url, db, username, api_key):
        self.common = xmlrpc.client.ServerProxy(f'{url}/xmlrpc/2/common')
        self.object = xmlrpc.client.ServerProxy(f'{url}/xmlrpc/2/object')
        self.uid = self.authenticate()

    async def execute_kw(self, model, method, args, kwargs=None):
        # Wrapper for auth & error handling
        loop = asyncio.get_running_loop()
        return await loop.run_in_executor(
            None, self.object.execute_kw,
            self.db, self.uid, self.password,
            model, method, args, kwargs or {}
        )
```

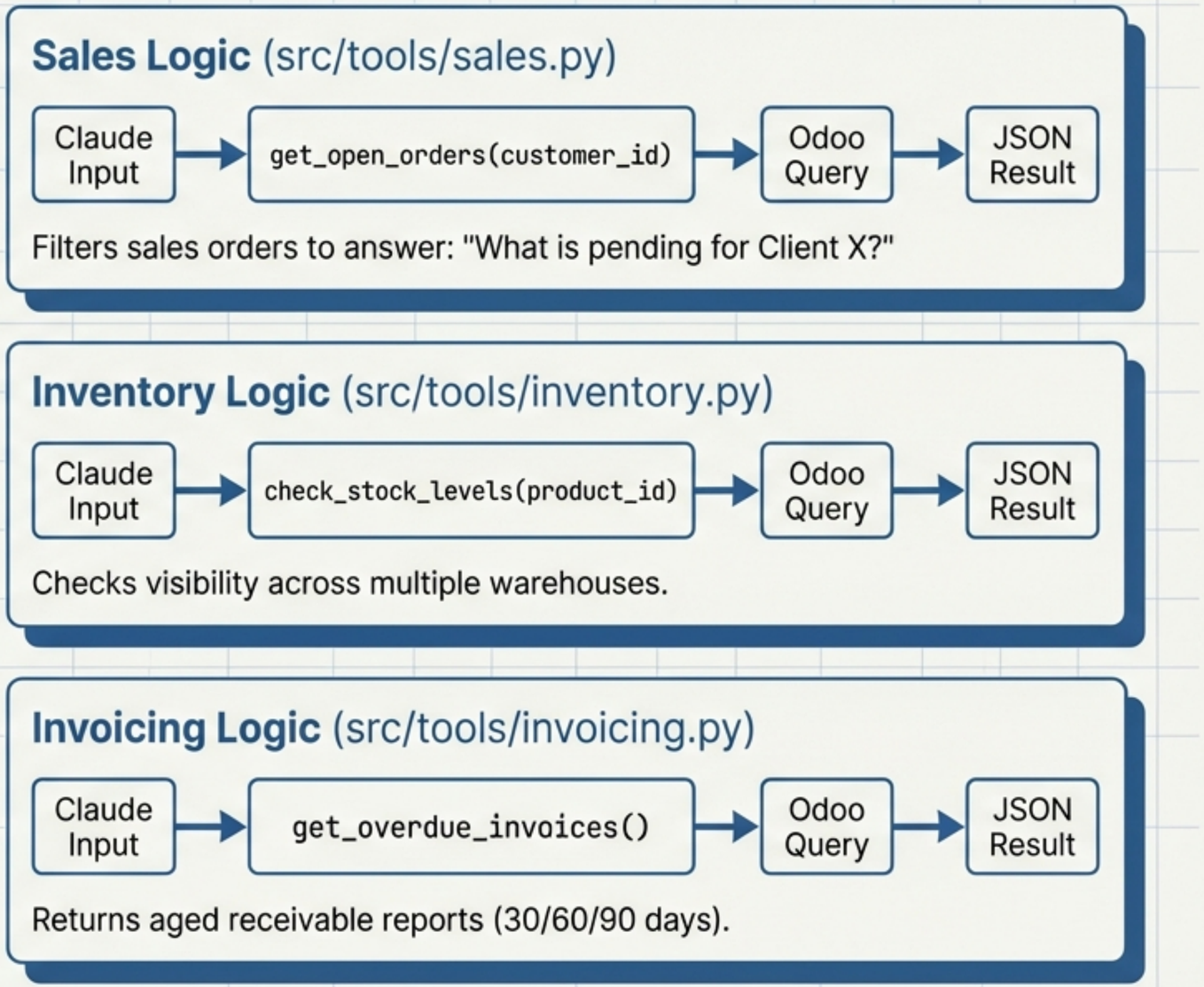
Two-Endpoint Logic (Auth vs Ops)

Async Wrapper for Blocking Calls

NotebookLM

# Toolset A: Intelligence & Analysis (Read Ops)

Read capabilities allow the AI to acquire context from the ERP before making decisions.




NotebookLM

Engineering Editorial

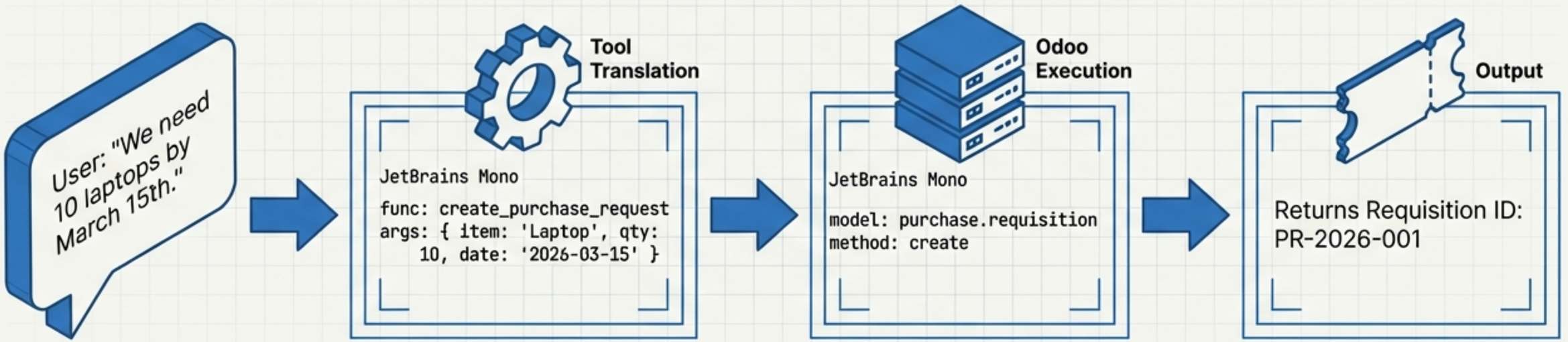
# Toolset B: Action & Execution (Write Ops)

The transition to agentic behavior: performing actions, not just retrieving info.

### Other Write Capabilities:

- **confirm_sales_order()**: Converts draft quotes to confirmed orders.
- **flag_vendor_bill()**: Marks anomalies in recent bills.

## Purchase Request Workflow




NotebookLM

# The Assembly: FastMCP Server & Lifespan

Efficient resource management using the "Lifespan" pattern to reuse connections.

src/server.py

```python
from mcp.server.fastmcp import FastMCP
from contextlib import asynccontextmanager

@asynccontextmanager
async def lifespan(server: FastMCP):
    # Initialize Odoo Client once on startup
    odoo = OdooClient.from_env()
    yield {'odoo': odoo}
    # Cleanup on shutdown (if needed)

mcp = FastMCP('Odoo Agent', lifespan=lifespan)

@mcp.tool()
async def check_stock(product: str, ctx: Context) -> str:
    odoo = ctx.request_context.lifespan_context['odoo']
    return await odoo.execute_kw(..., [product])
```

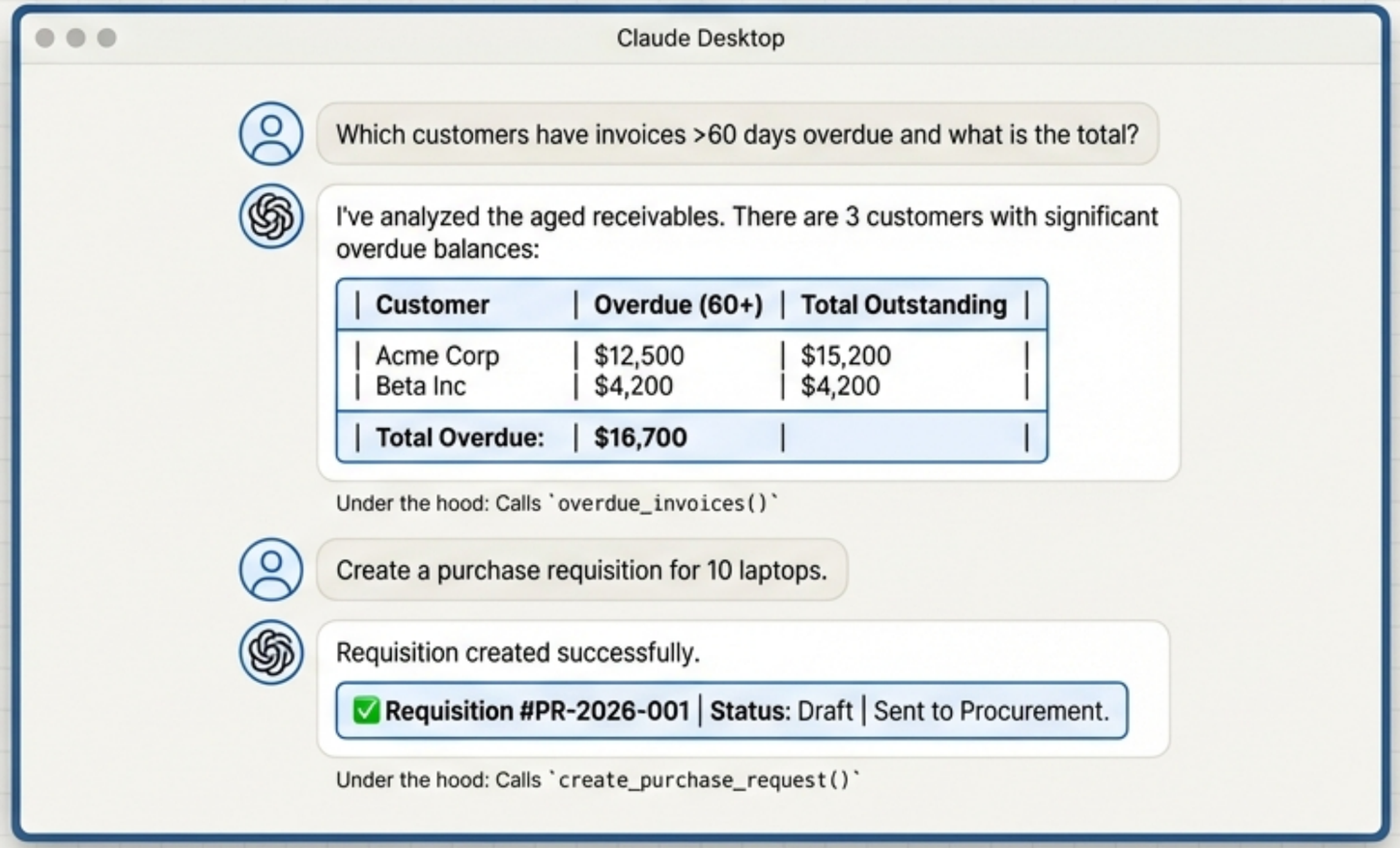
**Lifespan Context**: Initializes connection once, injects into every tool call.

NotebookLM

Engineering Editorial

# **In Practice:** From Natural Language to **ERP Action**

Complex ERP navigation is replaced by pure intent-to-action.




NotebookLM

Engineering Editorial

# **Production Hardening:** Security & Access Control

## Least Privilege

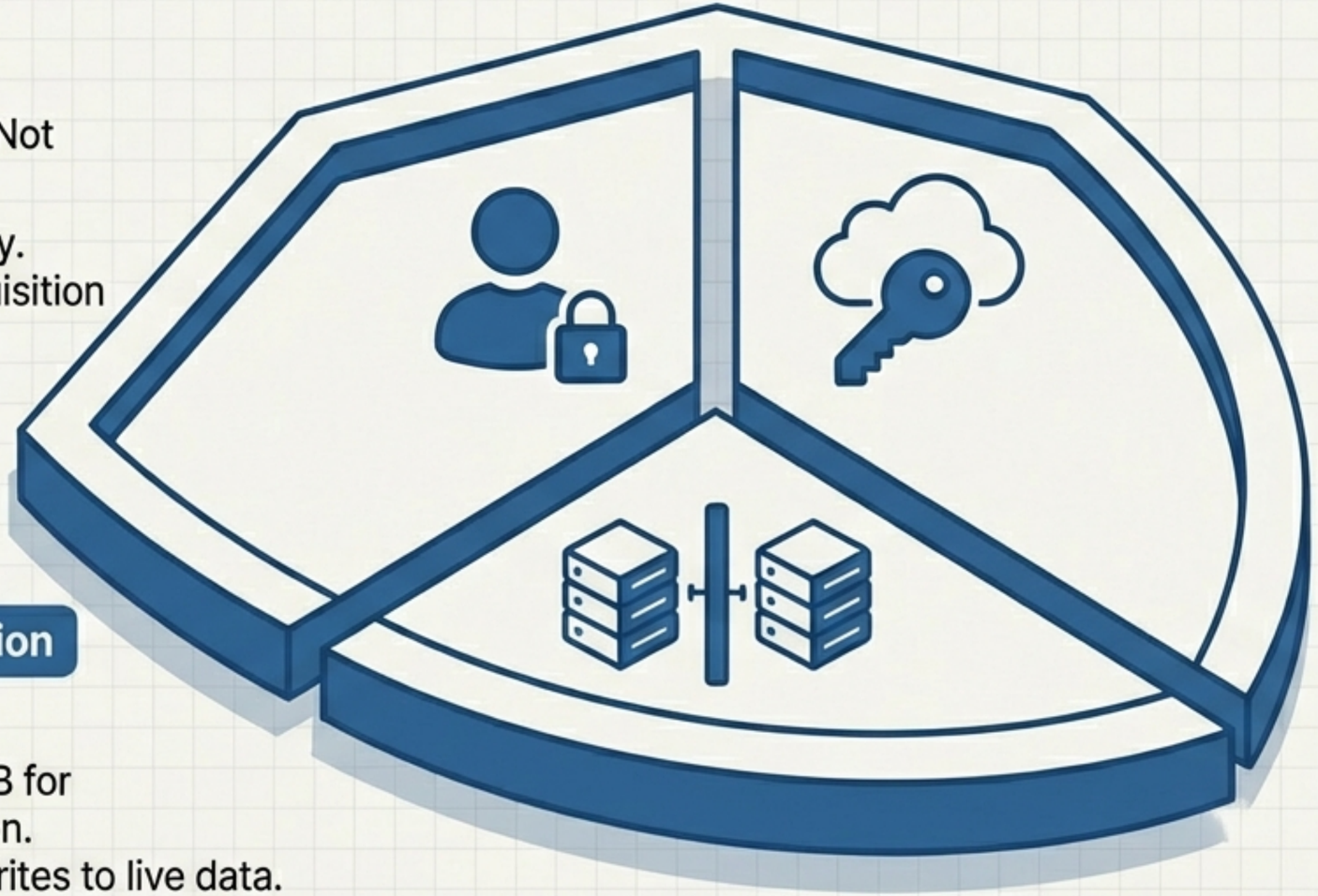
Dedicated API User (Not Admin).

Read: Sales, Inventory.
Write: Purchase Requisition ONLY.

## Secrets Management

Infrastructure Level.

Replace .env files.
Inject via AWS Secrets Manager or HashiCorp Vault.



## Environment Isolation

Config Separation.

Distinct Odoo URL/DB for Staging vs. Production.
Prevent accidental writes to live data.

## Cernentinn! Isolation

Unvenuale Level.

Prevent conventilnilities.

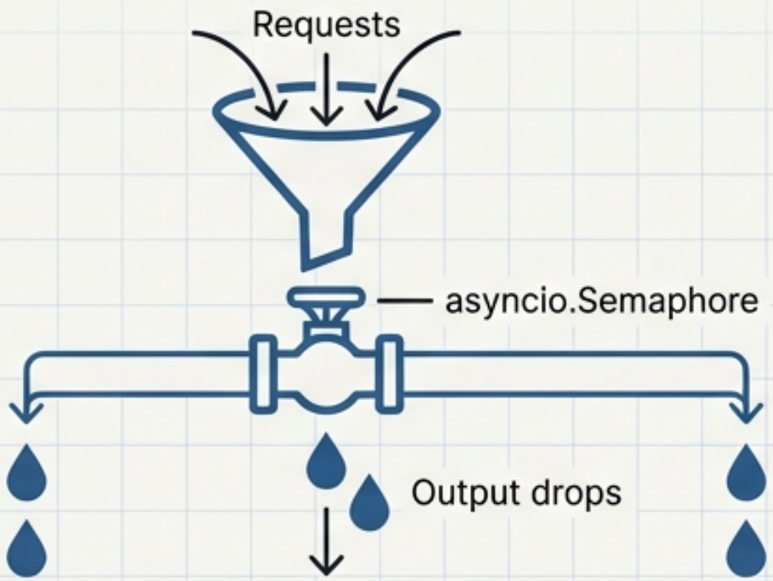
NotebookLM

Engineering Editorial

# Reliability: Rate Limiting & Health Checks

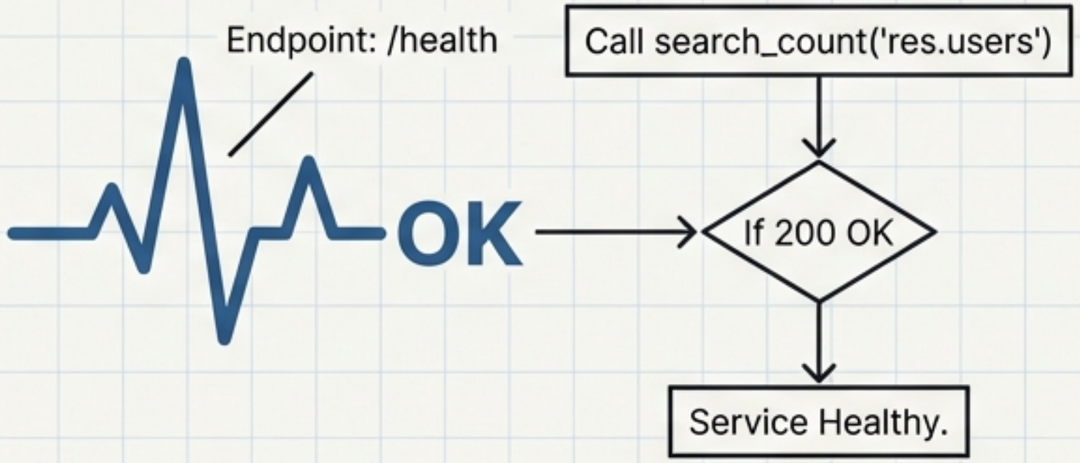
Essential for deploying the MCP server as a long-running HTTP service.

### Rate Limiting (Token Bucket)




Prevents flooding the Odoo XML-RPC interface in multi-user environments.

### Health Check




Verifies Odoo reachability for load balancers.

NotebookLM

Engineering Editorial

# Observability: Logging & Caching Strategy

## Structured Logging

```
[INFO] [Req-ID: abc-123] Tool 'check_stock' started
[INFO] [Req-ID: abc-123] Odoo Query: 45ms
[INFO] [Req-ID: abc-123] Tool 'check_stock' completed successfully
```

→ Traceability via Request IDs.

## Caching Strategy

Library: cachetools (TTL Cache)

- Cache Static Data: Product Lists, Customer IDs
- **Method:** search_read
- **Benefit:** Reduces DB load for frequently accessed read-only data.

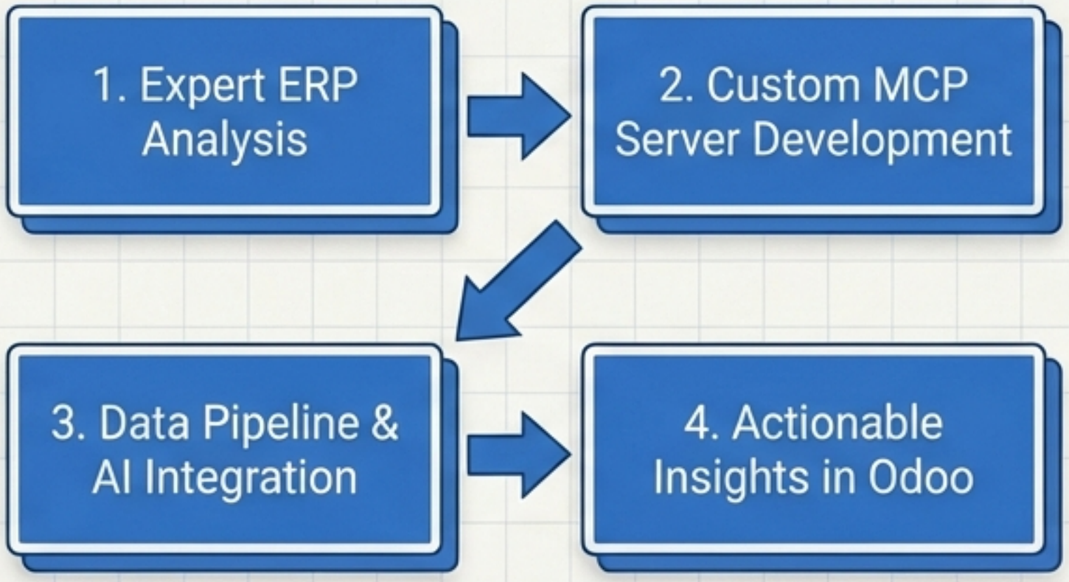
NotebookLM

Engineering Editorial

# How Bithost Helps Organizations: Implementing Data-Driven, Decision-Making Integration in Odoo

Empowering your business with intelligent, scalable ERP integrations.

## Bithost's Approach




## Key Benefits for Decision Making

- **Unified Data View:** Consolidate data across systems.
- **Real-time Decision Support:** AI-driven analysis within Odoo.
- **Automated Workflows:** Trigger actions based on data insights.
- **Scalable & Secure Architecture:** Built for growth and compliance.

NotebookLM