**Repo layout** for your **AML Agentic Platform** with a Master agent + MCP servers. It’s designed to be minimal, modular, and extendable.

**📂 Repo Structure**

aml-agentic-platform/

│

├── master-agent/ # Orchestrator

│ ├── main.py # Entry point

│ ├── planner.py # Task decomposition

│ ├── executor.py # Calls agents

│ ├── verifier.py # Verification logic

│ ├── kb/ # Knowledge base (graph store)

│ │ ├── kb\_graph.json

│ │ └── kb\_utils.py

│ └── requirements.txt

│

├── mcp\_databricks/ # MCP server for Databricks

│ ├── server.py

│ ├── tools/

│ │ ├── dlt\_create\_update.py

│ │ ├── jobs\_run.py

│ │ └── uc\_grant.py

│ └── requirements.txt

│

├── mcp\_synapse\_sql/ # MCP server for Synapse Serverless

│ ├── server.py

│ ├── tools/

│ │ ├── exec\_query.py

│ │ ├── create\_view.py

│ │ └── cetas.py

│ └── requirements.txt

│

├── mcp\_mlflow/ # MCP server for ML lifecycle

│ ├── server.py

│ ├── tools/

│ │ ├── train.py

│ │ ├── register.py

│ │ ├── promote.py

│ │ └── batch\_score.py

│ └── requirements.txt

│

├── mcp\_sql\_case\_pbi/ # MCP server for Case Mgmt & BI

│ ├── server.py

│ ├── tools/

│ │ ├── sql\_exec.py

│ │ ├── schema\_ensure.py

│ │ ├── pbi\_refresh.py

│ │ └── pbi\_rbac\_set.py

│ └── requirements.txt

│

└── README.md

**🧠 Master Agent (Orchestrator)**

**master-agent/main.py**

import sys, json

from planner import plan\_tasks

from executor import execute\_plan

from kb.kb\_utils import record\_plan, record\_result

def main():

request = sys.argv[1] if len(sys.argv) > 1 else "Deploy baseline rules"

plan = plan\_tasks(request)

record\_plan(plan)

for task in plan:

result = execute\_plan(task)

record\_result(task, result)

print("✅ All tasks executed successfully")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**master-agent/planner.py** (toy decomposition logic)

def plan\_tasks(request: str):

if "baseline rules" in request.lower():

return [

{

"id": "T-1",

"agent": "databricks",

"tool": "dlt\_create\_update",

"args": {

"catalog": "aml",

"target\_schema": "silver",

"notebooks": [

"/Repos/aml/dlt/watchlists.py",

"/Repos/aml/dlt/rules\_baseline.py"

],

"continuous": True,

"storage": "abfss://logs@<sa>.dfs.core.windows.net/dlt/aml-core"

}

},

{

"id": "T-2",

"agent": "synapse\_sql",

"tool": "create\_view",

"args": {

"sql": "CREATE OR ALTER VIEW vw\_alerts AS SELECT \* FROM OPENROWSET(...)"

}

}

]

return []

**master-agent/executor.py**

import subprocess, json

def execute\_plan(task):

agent = task["agent"]

tool = task["tool"]

args = json.dumps(task["args"])

# Example: call MCP server via CLI

cmd = ["mcp", agent, tool, args] # placeholder, adapt to your MCP client

print(f"▶ Running {agent}.{tool} ...")

result = subprocess.run(cmd, capture\_output=True, text=True)

return {"task\_id": task["id"], "stdout": result.stdout, "stderr": result.stderr}

**master-agent/verifier.py**

def verify\_result(result):

if "error" in result["stderr"].lower():

return False

return True

**🛠 Example MCP server (Databricks)**

**mcp\_databricks/server.py**

from mcp.server.fastmcp import FastMCPServer

from tools.dlt\_create\_update import dlt\_create\_update

from tools.jobs\_run import jobs\_run

from tools.uc\_grant import uc\_grant

server = FastMCPServer("databricks")

server.register\_tool(dlt\_create\_update)

server.register\_tool(jobs\_run)

server.register\_tool(uc\_grant)

if \_\_name\_\_ == "\_\_main\_\_":

server.run()

**mcp\_databricks/tools/dlt\_create\_update.py**

from mcp.server.fastmcp import Tool

@Tool("dlt\_create\_update")

def dlt\_create\_update(catalog: str, target\_schema: str, notebooks: list, continuous: bool, storage: str):

"""

Create or update a DLT pipeline in Databricks.

"""

# TODO: use databricks-cli or REST API

return {"status":"ok", "pipeline\_id":"pipeline-123", "tables":[f"{catalog}.{target\_schema}.alerts"]}

**📖 README.md**

# AML Agentic Platform

This repo implements an \*\*Agentic AML Platform\*\* with:

- \*\*Master Agent\*\*: Plans, executes, verifies tasks.

- \*\*MCP Servers\*\*: Wrap Azure/Databricks/ML/SQL/PBI functions.

## Agents

- `mcp\_databricks`: Databricks (DLT, Jobs, UC).

- `mcp\_synapse\_sql`: Synapse Serverless (views, queries).

- `mcp\_mlflow`: ML lifecycle (train, register, score).

- `mcp\_sql\_case\_pbi`: Case mgmt (SQL, Power BI refresh).

## Running

1. Start MCP servers:

```bash

uv run mcp\_databricks/server.py

uv run mcp\_synapse\_sql/server.py

1. Run master agent:
2. uv run master-agent/main.py "Deploy baseline rules"

All tasks, results, and lineage are persisted in master-agent/kb/.

---

# 🔜 Next Steps

- Fill in \*\*real implementations\*\* in each tool (`az cli`, `databricks-cli`, `pyodbc`, `mlflow`).

- Wrap everything with your \*\*`syntropAIkit.mcp.BaseSession`\*\* (for retry, logging, safe builtins).

- Add \*\*gating/approval logic\*\* in `master-agent/executor.py` before prod runs.

- Connect to your \*\*knowledge graph\*\* so Master Agent reasons over current state.

---

👉 Do you want me to \*\*generate working implementations\*\* (e.g. Databricks API call for DLT