Excel Processor Setup Documentation

1. Requirements Document

Business Requirement
- Automate updating a master Excel portfolio workbook with values from a Fidelity CSV export.
- Each ticker sheet in the Excel template has account headers in row 1 (B1..F1).
- The CSV “Account Name” field doesn’t match exactly; we must normalize & map.
- For each Symbol × Account Name, aggregate:
  • Quantity → Excel row 24 (“Total Buy Qty”).
  • Cost Basis Total → Excel row 39 (“Buy Shunks”).
- Must support adding new accounts in future without re-coding (via DynamoDB mapping).
- Must scale with minimal cost (serverless, pay-per-use).

Architecture Goals
- Store inputs/outputs in Amazon S3.
- Trigger an AWS Lambda on CSV upload.
- Lambda loads the Excel template, updates rows, writes back an updated workbook + run report.
- Provide RunReport in JSON + Excel sheet for validation.
- Expose an MCP WebSocket API so AI agents can orchestrate flows.
- Optional Bedrock integration to infer mappings with embeddings/LLMs.

2. Implementation Document

AWS Components
- S3 bucket (versioned, KMS-encrypted).
- KMS CMK for encryption.
- DynamoDB table “MappingOverrides” (account name → Excel header mapping).
- Lambda “ExcelProcessorFn” (Python 3.11, openpyxl, boto3).
- API Gateway WebSocket API (“McpWsApi”) for MCP tool calls.
- DynamoDB table “WsConnections” to track WebSocket clients.

CDK Repo Layout
aws-nasmatchstockprofile-mcp/
■■■ bin/app.ts
■■■ lib/base-infra.ts
■■■ lambda/
■ ■■■ processor/handler.py
■ ■■■ wsmcp/
■ ■■■ on_connect.py
■ ■■■ on_disconnect.py
■ ■■■ on_message.py
■■■ docs/ExcelProcessorSetup.pdf

Lambda Logic
- Parse CSV by header (Account Name, Symbol, Quantity, Cost Basis Total).
- Normalize account names (_norm_header).
- Resolve mapping (DDB > Env).
- Aggregate by symbol → account header.

- Write into Excel ticker sheets (row 24 = Qty, row 39 = Cost).
- Save run report JSON + RunReport sheet.

Example Mapping
BrokerageLink → 401K
BrokerageLink Roth → 401 ROTH
Health Savings Account → HSA
ROTH IRA (after-tax Mega BackDoor Roth) → ROTH IRA1
INDIVIDUAL-Margin → Brokerage

Deployment
npm install
npm run build
cdk deploy BaseInfra

Invocation & Validation
aws lambda invoke --function-name BaseInfra-ExcelProcessorFn ...
--payload '{ "source_key":"source/Portfolio_Positions.csv",
"target_key":"source/nasmatch-portfolio.xlsx", "output_key":"output/nasmatch-portfolio-updated.xlsx" }'

Then fetch report from S3 and verify in Excel.

Cost Profile
- S3 storage + requests = pennies.
- Lambda execution (ms-scale).
- DynamoDB PAY_PER_REQUEST (cheap).
- API Gateway WebSocket (per-message fee).