Part A — Design- Canonical event-driven commerce platform enabling independent billing and fulfillment migration to Azure with full auditability and replay

Commerce Transaction Platform (Monolith to Azure Migration)

Reference Architecture Diagram

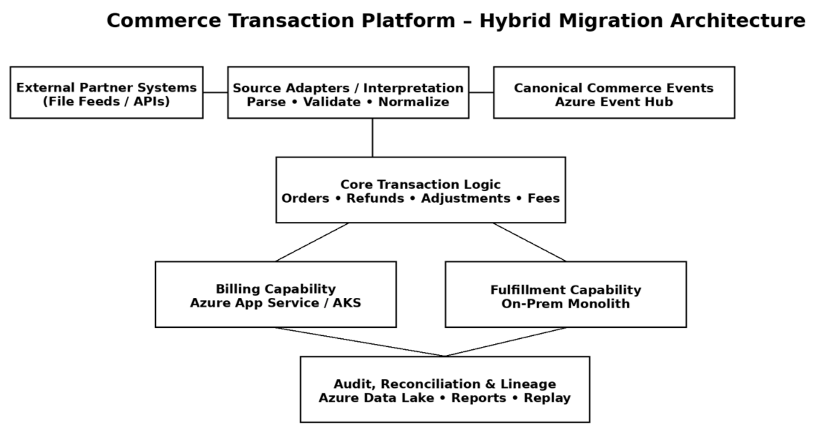

Commerce Transaction Platform – Hybrid Migration Architecture
External Partner Systems (File Feeds / APIs)
Source Adapters / Interpretation Parse • Validate • Normalize
Canonical Commerce Events Azure Event Hub
Core Transaction Logic Orders • Refunds • Adjustments • Fees
Billing Capability Azure App Service / AKS
Fulfillment Capability On-Prem Monolith
Audit, Reconciliation & Lineage Azure Data Lake • Reports • Replay

Figure 1 Reference Architecture

External Partner Sources → Source Adapters → Canonical Commerce Events → Core Transaction Logic → Hybrid Capability Migration → Audit & Reconciliation

---

Written Explanation

Platform Approach

This design treats the commerce system as a **platform**, not a set of point-to-point integrations. Architecture isolates **partner variability at the edges** while keeping **transaction semantics stable at the core**, enabling correctness, auditability, and incremental modernization.

Separation of Source Interpretation from Core Logic

Each external partner is handled through a **source-specific adapter** responsible for parsing files, validating structure, and interpreting partner-specific semantics. Adapters contain no billing or fulfillment logic and may evolve independently as partner schemas change.

Interpreted data is converted into a **canonical commerce event model** that represents stable business intent (orders, order items, refunds, fees, and adjustments). All downstream processing operates exclusively on this canonical representation, ensuring that partner-specific differences never leak into core transaction logic.

Canonical Representation

The canonical model serves as a **contract between ingestion and transaction processing**. It decouples business meaning from file formats, supports versioning, and enables replay, correction, and audit. Canonical events are immutable and enriched with full lineage metadata (source, file identifier, row identifier, timestamps), allowing deterministic reprocessing and reconciliation.

Azure Event Hub vs Event Grid

The authoritative transaction stream is implemented using **Azure Event Hub**. Event Hub is selected because the platform requires durable, high-throughput, ordered event streams with consumer-controlled replay to support billing correctness, reconciliation, and hybrid operation during migration.

Azure Event Grid is intentionally not used as the system of record for transactions. Event Grid is better suited for reactive notifications and orchestration (e.g., signaling batch completion or triggering downstream workflows) and may complement Event Hub, but it does not provide the replay and durability guarantees required for financial transaction processing.

Auditability and Reconciliation

Auditability is achieved through immutable canonical events, raw source retention, and deterministic transformations. Reconciliation is supported by replayable ingestion pipelines, explicit correction events, and comparison of source totals to canonical aggregates. This ensures operational confidence and regulatory readiness without manual intervention.

Monolith-to-Azure Migration (Billing First)

To migrate billing to Azure while fulfillment remains on-prem, the monolith is refactored to introduce **explicit capability boundaries**. Shared in-process calls are replaced with event-driven or service-level interfaces so billing no longer depends on monolithic state.

Fulfillment continues on-prem and publishes canonical events. Azure-based billing consumes the same events and evolves independently. Given intermittent connectivity, the system favors **availability over strict consistency**: fulfillment proceeds locally, and billing processes events asynchronously when connectivity resumes. Idempotent consumers prevent duplication.

Progressive cutover is achieved via dual publishing, shadow billing runs, and feature flags. Rollback is performed by disabling Azure consumers without impacting fulfillment, limiting blast radius and avoiding regressions.

Summary

By stabilizing the core around canonical events and absorbing change at the edges, this design enables rapid partner onboarding, safe hybrid migration, and long-term platform evolution without sacrificing correctness or auditability.

Part B — Order Normalization

This project normalizes heterogeneous order files from multiple external commerce systems into a canonical, item-level event format.

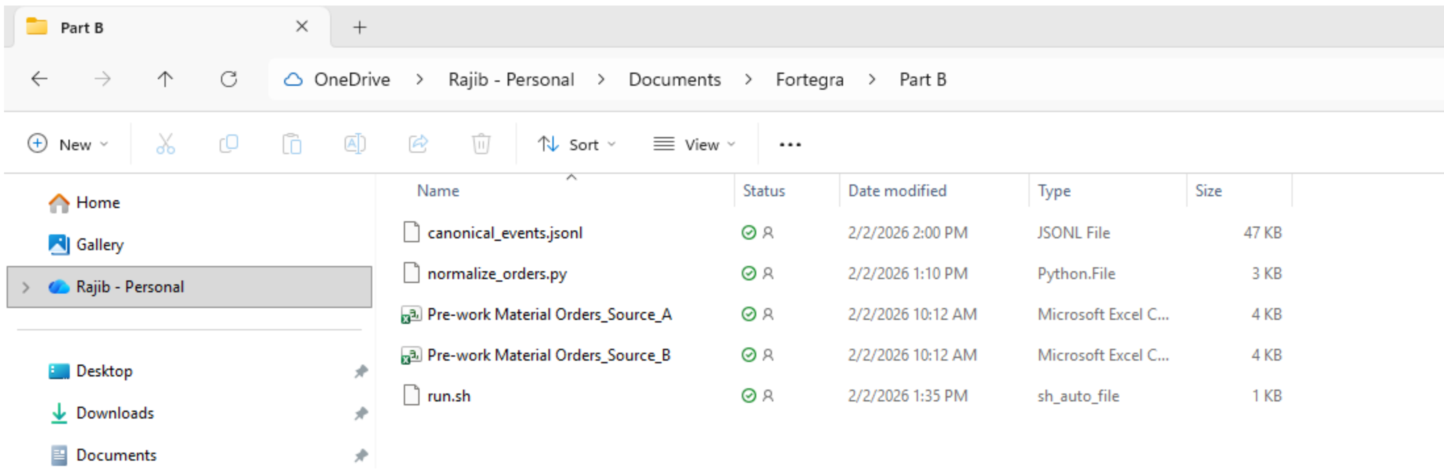
System A provides one row per order item, while System B embeds multiple SKUs within a single order row.

Source-specific adapters transform each input into a unified “OrderItemCreated” event model.

The solution is CLI-driven to mirror real-world batch ingestion pipelines.

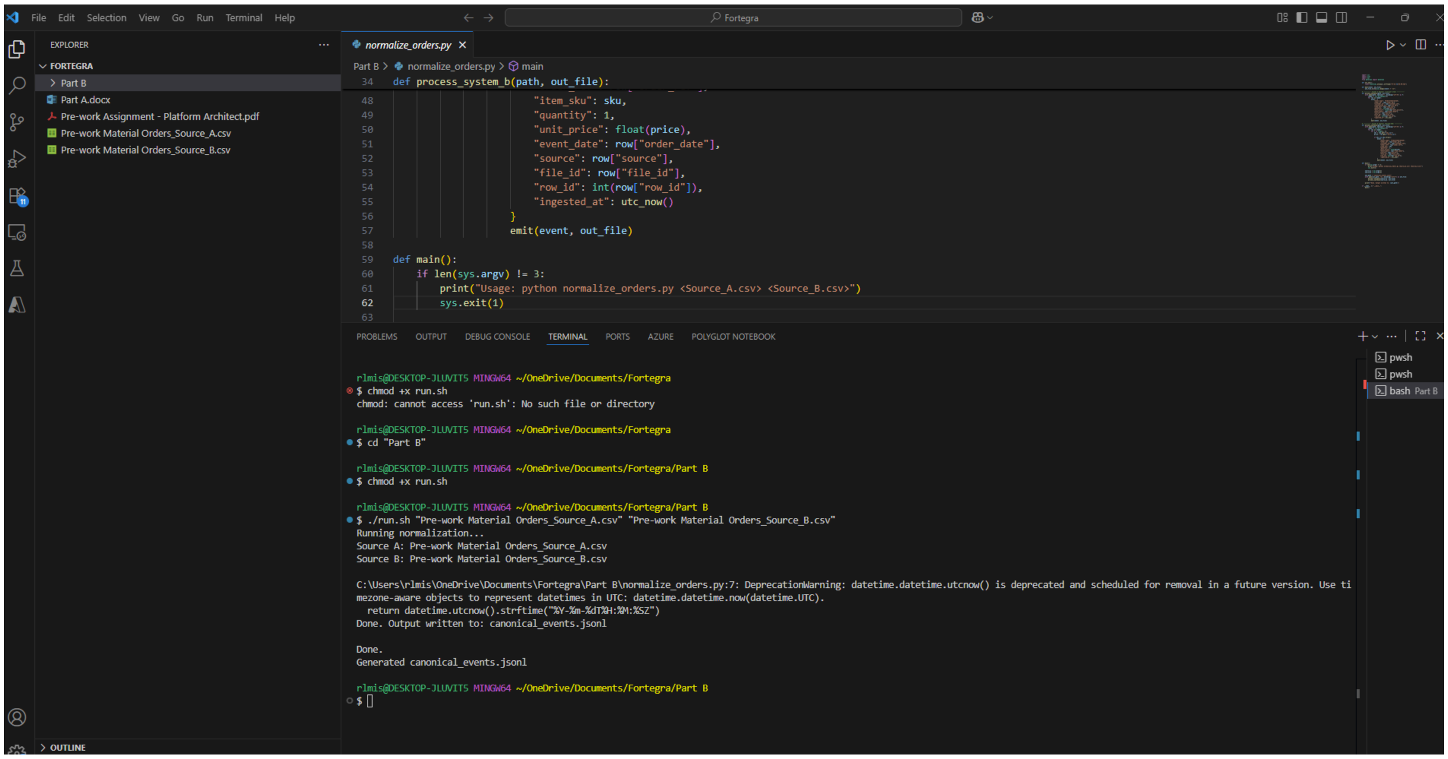
Run: “./run.sh line_item_orders.csv packed_orders.csv”

1. Extract all the files attached in the email to a folder similar to shown below.



Name	Status	Date modified	Type	Size
canonical_events.jsonl	✓	2/2/2026 2:00 PM	JSONL File	47 KB
normalize_orders.py	✓	2/2/2026 1:10 PM	Python.File	3 KB
Pre-work Material Orders_Source_A	✓	2/2/2026 10:12 AM	Microsoft Excel C...	4 KB
Pre-work Material Orders_Source_B	✓	2/2/2026 10:12 AM	Microsoft Excel C...	4 KB
run.sh	✓	2/2/2026 1:35 PM	sh_auto_file	1 KB

2. chmod +x run.sh…as per the compress file attached in the email after extracting to a folder that is open in VSCode in Windows with python library installed.
3. In VSCode open the folder ➔ Terminal Menu ➔Select Bash ➔ Run “chmod +x run.sh”
4. Run “./run.sh "Pre-work Material Orders_Source_A.csv" "Pre-work Material Orders_Source_B.csv"
5. You should see the following output as I have tested.



File Edit Selection View Go Run Terminal Help
Fortegra
EXPLORER
FORTEGRA
Part B
Part A.docx
Pre-work Assignment - Platform Architect.pdf
Pre-work Material Orders_Source_A.csv
Pre-work Material Orders_Source_B.csv
normalize_orders.py
Part B > normalize_orders.py > main
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE POLYGLOT NOTEBOOK
pwsh
pwsh
bash Part B
OUTLINE

```python
def process_system_b(path, out_file):
                    "item_sku": sku,
                    "quantity": 1,
                    "unit_price": float(price),
                    "event_date": row["order_date"],
                    "source": row["source"],
                    "file_id": row["file_id"],
                    "row_id": int(row["row_id"]),
                    "ingested_at": utc_now()
                }
                emit(event, out_file)

def main():
    if len(sys.argv) != 3:
        print("Usage: python normalize_orders.py <Source_A.csv> <Source_B.csv>")
        sys.exit(1)
```

```
rlmis@DESKTOP-JLUVIT5 MINGW64 ~/OneDrive/Documents/Fortegra
$ chmod +x run.sh
chmod: cannot access 'run.sh': No such file or directory

rlmis@DESKTOP-JLUVIT5 MINGW64 ~/OneDrive/Documents/Fortegra
$ cd "Part B"

rlmis@DESKTOP-JLUVIT5 MINGW64 ~/OneDrive/Documents/Fortegra/Part B
$ chmod +x run.sh

rlmis@DESKTOP-JLUVIT5 MINGW64 ~/OneDrive/Documents/Fortegra/Part B
$ ./run.sh "Pre-work Material Orders_Source_A.csv" "Pre-work Material Orders_Source_B.csv"
Running normalization...
Source A: Pre-work Material Orders_Source_A.csv
Source B: Pre-work Material Orders_Source_B.csv

C:\Users\rlmis\OneDrive\Documents\Fortegra\Part B\normalize_orders.py:7: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use ti
mezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  return datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%SZ")
Done. Output written to: canonical_events.jsonl

Done.
Generated canonical_events.jsonl

rlmis@DESKTOP-JLUVIT5 MINGW64 ~/OneDrive/Documents/Fortegra/Part B
$ 
```

Clarifying questions – Risk Reduction

The solution demonstrates how heterogeneous partner inputs can be normalized into a stable canonical event model, enabling Fortegra to scale onboarding while retaining governance, auditability, and decision authority.

Source & Data Semantics

- Are SKUs unique **per partner, per program**, or **globally**?
- Can a single order contain mixed product semantics?
- How should retired or renamed SKUs be handled?

Volume & Latency

- Expected peak order and item throughput per source?
- Batch vs near-real-time expectations?
- Replay windows and retention requirements?

Governance & Compliance

- Regulatory retention and audit requirements?
- Source-level SLAs and data quality expectations?
- How are disputes and corrections handled?

Platform Evolution

- Expected cadence of new partner onboarding?
- Schema evolution tolerance (backward/forward compatibility)?
- AI advisory vs decision authority boundaries?