Feasibility Assessment: HL7 v2 → FHIR Conversion in Python

1. Objective

The objective is to identify a sustainable and maintainable approach for converting HL7 v2 messages into FHIR R4 resources within a Python-based ecosystem. Several options exist, ranging from custom mapping implementations to leveraging Microsoft's HL7 FHIR Converter or enterprise integration engines such as Rhapsody.

2. Approaches Considered

2.1 Manual Parsing and Mapping (Custom Python + HL7 Libraries)

Description: Use Python libraries (hl7apy, fhir.resources) to parse HL7 messages and manually map them into FHIR resources. This requires implementing mapping logic for segments such as PID → Patient, OBR/OBX → Observation, etc.

Pros: Full control, lightweight dependencies, no external systems required.

Cons: High implementation and maintenance burden, requires in-house HL7/FHIR expertise, error-prone, hard to scale.

Feasibility: Only suitable for small proof-of-concepts or narrow-scope projects.

2.2 Azure FHIR $convert-data Endpoint (Managed Service)

Description: Leverage Azure Health Data Services' $convert-data endpoint to POST HL7 v2 messages and receive a FHIR Bundle. Microsoft manages templates and infrastructure. Integration is done via REST with Azure AD authentication.

Pros: Low setup, production-ready, maintained by Microsoft, good for Azure-centric enterprises.

Cons: Vendor lock-in, PHI residency tied to Azure data centers, subscription costs, limited portability outside Azure.

Feasibility: Best for organizations heavily invested in Azure ecosystem.

2.3 Self-Hosted Microsoft HL7 FHIR Converter (Containerized REST Service)

Description: Deploy Microsoft's open-source HL7 FHIR Converter as a Docker container or Kubernetes service. Python applications interact with it via REST: sending HL7 messages and receiving FHIR JSON Bundles. Mapping is encapsulated in Liquid templates which can be customized without modifying application code.

Pros: Encapsulation of HL7 parsing and FHIR mapping outside of Python app, modular and extensible architecture, deployment flexibility (on-prem, cloud, hybrid), avoids vendor lock-in, compliant for PHI handling.

Cons: Requires DevOps resources for deployment, scaling, monitoring, and patching.

Feasibility: Highly recommended as the cleanest and most strategic choice for production environments.

2.4 Rhapsody Integration Engine (Commercial HL7/FHIR Translator)

Description: Use Rhapsody from Lyniate to parse HL7 messages and transform them into FHIR resources. Rhapsody provides a GUI-driven integration engine, message routing, auditing, and monitoring, with built-in FHIR support.

Pros: Mature, widely adopted in healthcare, strong enterprise features (monitoring, auditing, error handling), support for multiple integration patterns and data sources.

Cons: High licensing cost, requires specialized integration team, overhead is heavy if used solely for HL7→FHIR conversion.

Feasibility: Strong for enterprises already invested in Rhapsody, less ideal if introduced only for HL7→FHIR translation.

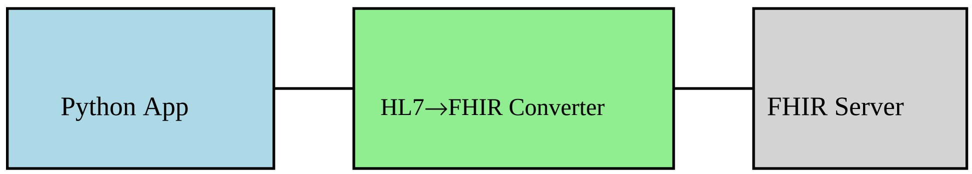
3. Comparative Summary

Approach	Effort	Maintenance	Encapsulation	Vendor Dependency	Cost	Compliance	Suitability
anual Python Mapping	High	High	None	None	Low	High	POC/small scope
Azure $convert-data	Low	Low	High	High	Medium	Limited	Azure-native
Self-Hosted Converter	Medium	Medium	High	Low	Low	High	Recommended
Rhapsody Engine	Medium	Med/High	High	High	High	High	If invested in Rhapsody

4. Recommendation

While all four approaches are technically feasible, the Self-Hosted Microsoft HL7 FHIR Converter (Option 2.3) is the cleanest and most strategic choice. It encapsulates HL7 parsing and FHIR mapping in a modular REST service, keeping the Python ecosystem lightweight and free of domain-specific complexity. This approach provides flexibility for deployment across environments and ensures compliance with PHI handling requirements. Rhapsody is only advisable if the organization is already invested in its ecosystem.

5. High-Level Architecture


Python App
HL7→FHIR Converter
FHIR Server