**Feasibility Assessment: HL7 v2 → FHIR Conversion**

**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 mappings to leveraging mature healthcare integration platforms.

**2. Approaches Considered**

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

**Description:**

* Use Python libraries (hl7apy, fhir.resources) to parse HL7 and construct FHIR resources.
* Build and maintain custom mapping logic for each message type.

**Pros:**

* No external dependencies beyond Python.
* Maximum flexibility in defining mappings.
* Full control of pipeline.

**Cons:**

* High development and maintenance burden.
* Requires deep HL7 and FHIR expertise in-house.
* Error-prone and hard to standardize across message types.

**Feasibility:**

* Works for small, narrow POCs.
* **Not recommended** for production where scale, compliance, and evolving standards matter.

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

**Description:**

* Use Azure Health Data Services $convert-data endpoint.
* Microsoft-managed templates translate HL7 → FHIR.
* Python integrates via REST calls with Azure authentication.

**Pros:**

* Minimal setup; Microsoft manages conversion logic.
* Production-ready with SLAs.
* Easy to plug into Azure-native infrastructure.

**Cons:**

* Vendor lock-in (Azure dependency).
* Ongoing subscription costs.
* Limited customization of runtime environment.
* PHI data residency tied to Azure data centers.

**Feasibility:**

* Excellent for Azure-centric enterprises.
* May not fit organizations preferring on-premise/hybrid models.

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

**Description:**

* Deploy Microsoft’s [HL7 FHIR Converter](https://github.com/microsoft/hl7-fhir-converter) as a containerized REST API.
* Python apps send HL7 → receive FHIR JSON Bundles.
* Mapping logic encapsulated in Liquid templates, which can be customized.

**Pros:**

* **Encapsulates HL7 parsing & FHIR mapping** outside of Python app.
* Clear separation of concerns: Python focuses on orchestration.
* Extensible via Liquid templates without code rewrites.
* Portable (runs on-prem, cloud, Kubernetes).
* No vendor lock-in; can run in regulated environments.

**Cons:**

* Requires DevOps for hosting and maintenance.
* Responsibility for upgrades and monitoring lies with the team.

**Feasibility:**

* **Highly recommended** for production-grade architecture.
* Balances maintainability, compliance, and modularity.

**2.4 Rhapsody Integration Engine (Commercial HL7/FHIR Translator)**

**Description:**

* Use **Rhapsody** (from Lyniate) to parse HL7 and map to FHIR.
* Rhapsody provides a GUI-driven integration engine, message routing, and built-in FHIR support.
* Python apps would consume the converted FHIR output from Rhapsody via REST or file drop.

**Pros:**

* Mature, widely adopted in healthcare interoperability.
* Visual tools for defining HL7 → FHIR mappings.
* Enterprise features: monitoring, routing, auditing, error handling.
* Built-in connectors for multiple healthcare data sources.

**Cons:**

* **Commercial licensing cost** (significant vs open source).
* Requires dedicated integration team for configuration and ongoing support.
* Overhead may be heavy if only HL7→FHIR conversion is needed.
* Not as lightweight to embed in Python-native workflows.

**Feasibility:**

* Strong option for enterprises already using Rhapsody.
* Adds complexity and cost if introduced solely for HL7→FHIR conversion.

**3. Comparative Summary**

| **Approach** | **Effort** | **Maintenance** | **Encapsulation of FHIR Domain** | **Vendor Dependency** | **Cost** | **Compliance Flexibility** | **Suitability** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Manual Python Mapping | High | High | None | None | Low | High | POC/small scope only |
| Azure $convert-data | Low | Low | High (MSFT managed) | High (Azure-only) | Pay-as-you-go | Limited | Best for Azure-native |
| Self-Hosted Converter | Medium | Medium | High (clean encapsulation) | Low | Open-source | High | **Recommended** |
| Rhapsody Engine | Medium | Medium/High | High (engine-managed) | High (commercial vendor) | High (license) | High | Best if already invested in Rhapsody |

**4. Recommendation**

While Rhapsody is a strong option for organizations already invested in enterprise integration platforms, it introduces **significant cost and operational overhead** if adopted solely for HL7→FHIR conversion.

Between the Microsoft options:

* **Azure $convert-data** is compelling for Azure-native shops but introduces vendor lock-in and compliance considerations.
* **Self-Hosted Microsoft HL7 FHIR Converter** offers the **cleanest architecture** by encapsulating HL7 parsing and FHIR mapping in a modular REST service. This keeps the Python ecosystem lightweight and free of domain-specific complexity, while preserving control over compliance and deployment environments.