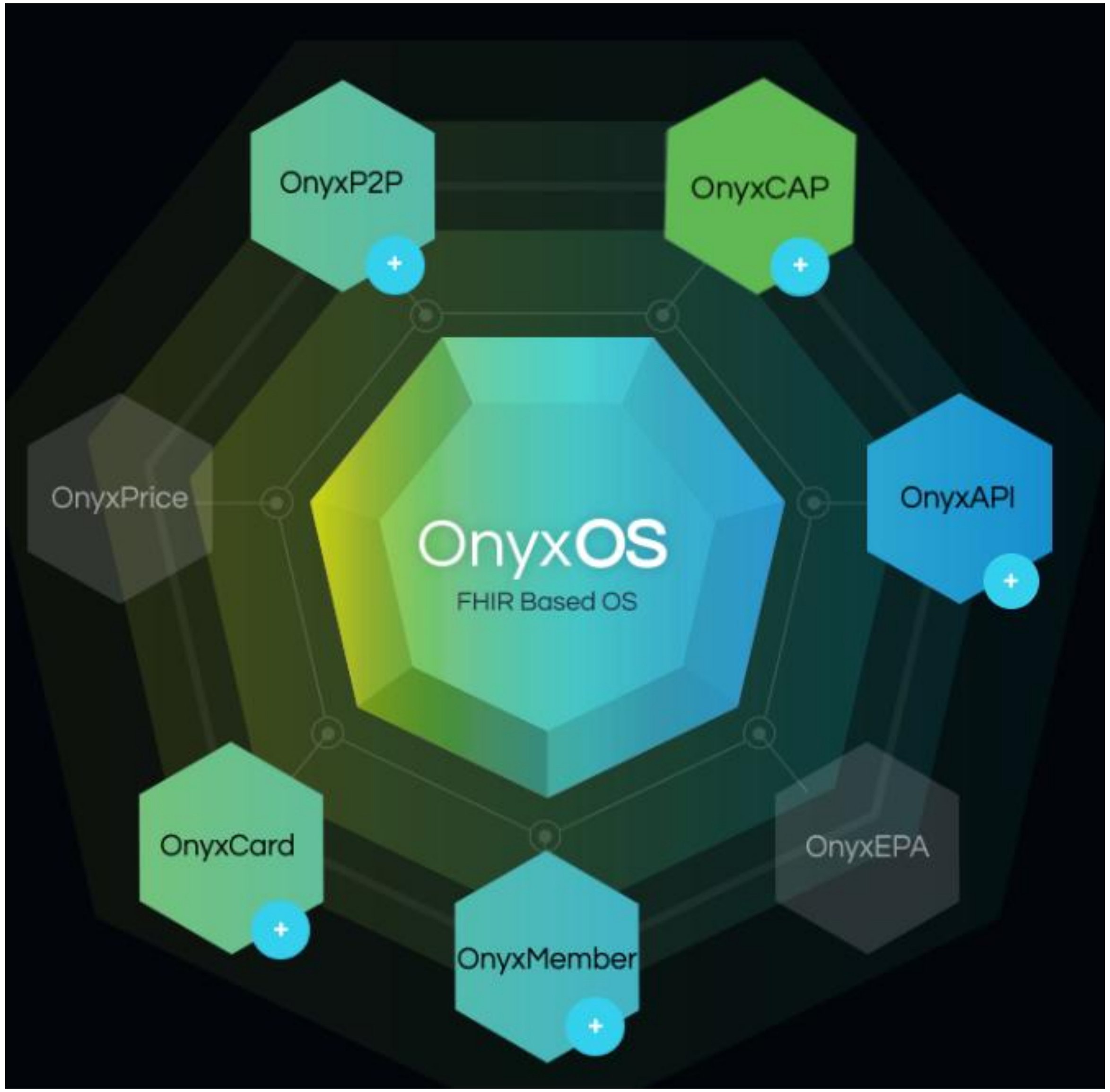


Onyx/Abacus-Data Ingestion

Converting Raw Data into FHIR API

IG – Implementation Guide

IG – NAME	IG- OFFICIAL URL	IG-VERSION	Resources	Purpose
Claims	https://hl7.org/fhir/us/carin-bb/STU2/index.html	2.0.0	Patient,Coverage,Oraganization,Practitioner, ExplanationOfBenefit	This implementation guide describes the CARIN for Blue Button® [*] Framework and Common Payer Consumer Data Set (CPCDS), providing a set of resources that payers can display to consumers via a FHIR API
Clinical	https://hl7.org/fhir/us/core/STU6.1/index.html	6.1.1	Observation, Encounter, Group, vitalsign,	Clinical information are here
PVD	https://hl7.org/fhir/us/davinci-pdex-plan-net/	1.1.0		
Formulary	https://hl7.org/fhir/us/davinci-drug-formulary/	2.0.1		

ABSTRACT

Claims 2.0 represents the latest implementation of the CARIN Blue Button Implementation Guide (CARIN BB IG) STU2.0 designed to facilitate the ingestion of claims related data into the Azure Health Data Service. This documentation provides a comprehensive guide to the entire data ingestion process, which involves converting raw data into FHIR (Fast Healthcare Interoperability Resources) resources that are accessible via REST APIs.

ONYX is fully prepared for the latest upgrades to the Azure Health Data Service and the newest STU2.0 Implementation Guide. Throughout the ingestion process, various logging mechanisms, including audit, error, and control logs, are employed to ensure transparency, error tracking, and process control. This documentation serves as a detailed guide for developers, data engineers, and system administrators, providing the necessary steps, configurations, and best practices to efficiently manage the data ingestion workflow and utilize the resulting FHIR resources.

HIGH LEVEL UNDERSTANDING

RAW DATA

FROM CLIENT SYSTEM(SFTP, SHIR(VM),blob strorage)

DATA related to profiles - Patient, Organization, Practitioner, Coverage, ExplanationOfBenefit

File format - .CSV/.TXT/.JSON/.XML

INPUT 2 ONYX/Abacus SYSTEM

CONVERT

PROCESS (ONYX SYSTEM)

GETTING THE SOURCE FILE

PREPROCESSING

- DUPLICATE CHECK,
- MANDATORY FIELDS
- DataType

FHIR TRANSFORM()

UPLOAD – post/put

UPSERT

FHIR REST API

INTEROPARABILITY

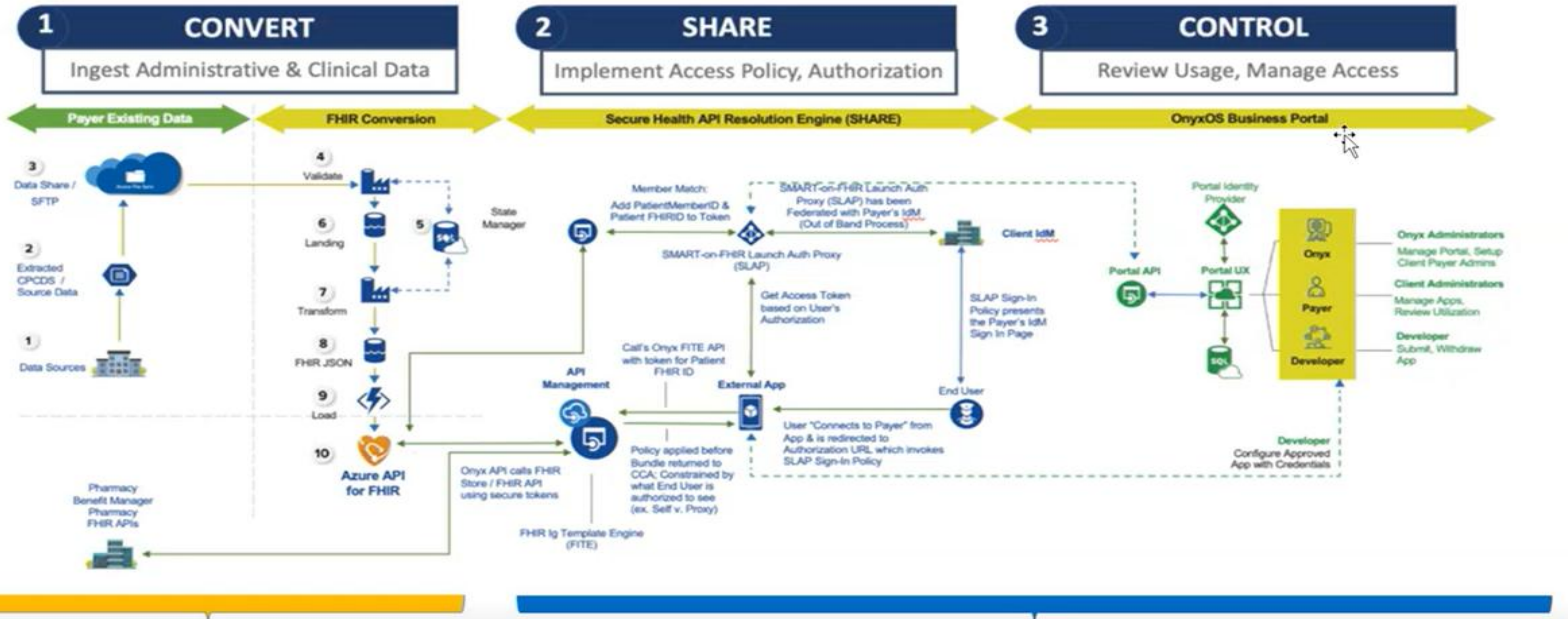
FHIR RESOURCE – Patient, Organization, Practitioner, Coverage, ExplanationOfBenefit

REST API ACESSESABLE – GET,PUT,POST,DELETE

THIS IS A STANDARD FOR DATA EXCHANGE DEFINED BY HL7

OnyxOS Platform – An API First Platform | Designed to support Multiple IGs

OnyxOS is an interoperability platform that supports multiple implementation guides for all business cases


1 CONVERT
Ingest Administrative & Clinical Data
2 SHARE
Implement Access Policy, Authorization
3 CONTROL
Review Usage, Manage Access
Payer Existing Data
FHIR Conversion
Secure Health API Resolution Engine (SHARE)
OnyxOS Business Portal
3 Data Share / SFTP
2 Extracted CPCDS / Source Data
1 Data Sources
4 Validate
6 Landing
5
State Manager
7 Transform
8 FHIR JSON
9 Load
10 Azure API for FHIR
Pharmacy Benefit Manager Pharmacy FHIR APIs
Onyx API calls FHIR Store / FHIR API using secure tokens
FHIR Ig Template Engine (FITE)
API Management
Member Match: Add PatientMemberID & Patient FHIRID to Token
SMART-on-FHIR Launch Auth Proxy (SLAP)
SMART-on-FHIR Launch Auth Proxy (SLAP) has been Federated with Payer's IdM (Out of Band Process)
Client IdM
Get Access Token based on User's Authorization
SLAP Sign-In Policy presents the Payer's IdM Sign In Page
Call's Onyx FITE API with token for Patient FHIR ID
External App
End User
Policy applied before Bundle returned to CCA; Constrained by what End User is authorized to see (ex. Self v. Proxy)
User "Connects to Payer" from App & is redirected to Authorization URL which invokes SLAP Sign-In Policy
Portal Identity Provider
Portal API
Portal UX
Onyx
Payer
Developer
Onyx Administrators
Manage Portal, Setup Client Payer Admins
Client Administrators
Manage Apps, Review Utilization
Developer
Submit, Withdraw App
Developer
Configure Approved App with Credentials

Out of Box Data Converters

Out of Box Policy & Consent Management

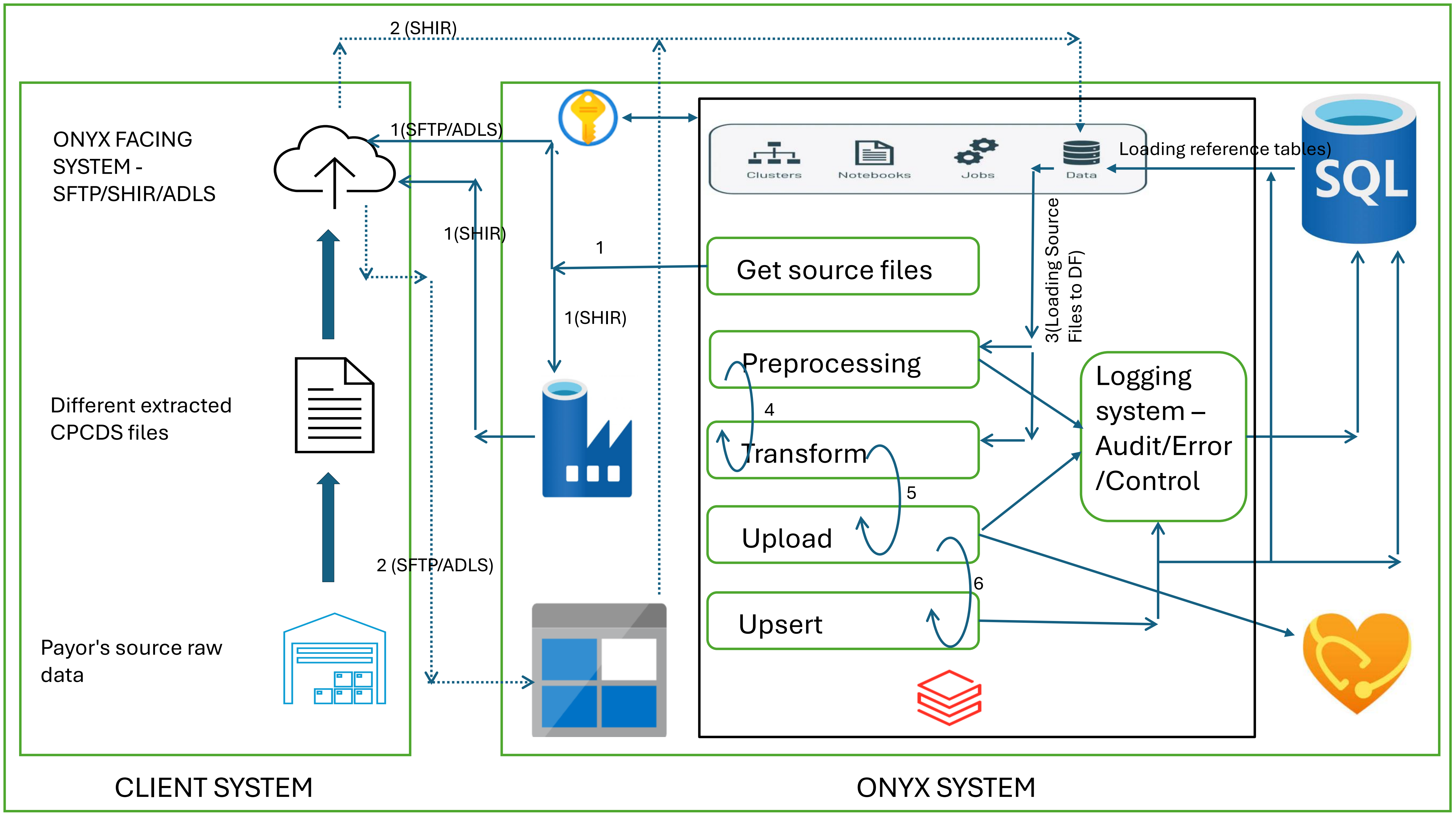
Tech Stack for Azure Data Ingestion

ONYX OS utilizes a robust Azure/AWS based tech stack to support its functionality, including-

- **Azure Databricks**: For data processing and transformation.
- **Azure MS SQL Server**: For database management.
- **SFTP**: For secure file transfers.
- **ADF**: used for copy activity to copy files from SHIR
- **ADLS**: For scalable data storage this is used in combination with ADF for copy activity.
- **Azure Health Data Service**: For managing and accessing FHIR resources. (earlier **Azure API for FHIR**)
- **Azure Key Vault**: For managing sensitive information and secrets.

Azure vs AWS

SN	AZURE	AWS	DEV EFFORT	PURPOSE
1	Application Insights	CLOUD WATCH	CODE CHANGES REQUIRED	To log the API hits
2	APIM	NA		Ingress is primarily used within a Kubernetes cluster to route external traffic to specific services within the cluster
3	AZURE KEY VAULT	CONFIG MAP AND SECRETS	CODE CHANGES REQUIRED	this is where we have kept all the secrets and env variable details for Azure implementation
4	APP CONFIG	CONFIG MAP AND SECRETS	CODE CHANGES REQUIRED	
5	APP SERVICE ENV VARAIBLE	ENV VARIABLE UNDER CONTAINERS	CODE CHANGES REQUIRED	
6	MS SQL SERVER	Delta Table - DataBrick - need to check with Srikanth/Shouray	CODE CHANGES REQUIRED	Audit, Error, Control Logs
7	MS SQL SERVER	Postgres	CODE CHANGES REQUIRED	User Management
8	MONGO DB	MONGO DB we have a separate dedicated DB for AWS	Ques - can we use the same mongo db for internal development ?	Centrailzed DB where we keep the data from different sources through Extraction API and this become data fuel for retrieval APIs
9	AZURE API FOR FHIR /Azure Health Data Service	FIRELY / HealthLake	Code Changes required	centralized data store for FHIR resources
10	SLAP V3 - AZURE VERSION	SLAP V3 - AWS VERSION	WIP - SLAP TEAM	WITH ENDPOINT TO FETCH THE MEMBER LOGIN REPORT
11	SLAP V2 - AZURE VERSION	SLAP V2 - AWS VERSION		
12	APP SERVICE- OnyxInsightsAPI	APP SERVICE- OnyxInsightsAPI		
13	APP SERVICE - OnyxInsightsUI	APP SERVICE - OnyxInsightsUI		


2 (SHIR)
ONYX FACING SYSTEM - SFTP/SHIR/ADLS
1(SFTP/ADLS)
1(SHIR)
1
1(SHIR)
Clusters
Notebooks
Jobs
Data
Loading reference tables)
SQL
3(Loading Source Files to DF)
Get source files
Preprocessing
4
Transform
5
Upload
6
Upsert
Logging system – Audit/Error /Control
Different extracted CPCDS files
Payor's source raw data
2 (SFTP/ADLS)
CLIENT SYSTEM
ONYX SYSTEM

ER not required because FHIR gives the internal data structure and the references with other profiles

Different entities and their relationship – defined by FHIR itself

For all the healthcare entities we do not need to put together the ER diagram as it is already defined by the FHIR

Patient	Organization	Practitioner	Coverage	Eob

Getting Source Files

Objective : This step involves transferring raw data files from the client's location to the Azure Databricks File System (ADB DBFS). Supported file formats and transfer methods are configured during this phase to ensure compatibility and security.

1. **Supported file formats:** .csv(comma or pipeline separated) (preferred one) , .txt(comma or pipeline separated), .json, .xml
2. **Supported Transfer methods**:
   1. SFTP (preferred one)
   2. ADLS based SFTP
   3. SHIR (Self Hosted Integration Runtime)
3. **Source file Naming convention**:
   1. CPCDS_Claims
   2. CPCDS_Members
   3. CPCDS_Coverages
   4. Organizations
   5. Practitioners

Preprocessing

Obejective : Raw data is cleaned, validated, and prepared for transformation. This step includes below logic

- mandatory check --- with respect to FHIR IG page
- duplicate check --- with respect to identifier value
- Audit log ---
  - counts how many records got processed before PreProc & after Preproc
  - This is a common table for all the profiles and igs
- Error log --- this will capture all the entries which got failed during the preprocessing ---- for each profile we will have individual error log table
  - Patient – PatientErrorlogTable
  - --
  - ------------------------------------
- Control log ---- to give the status with failure/success

- **Notebooks** – 8 preprocessing notebooks got created as Patient, Practitioner, Organization, Coverage, EOB-Inpatient, EOB-Outpatient, EOB-Professional, EOB-Pharmacy which runs in parallel and produce preprocessed .csv files

FHIR PREP

1. Preproc-df left join on bid(business identifier) with the cross-walk table

Patient.identifier value	Patient_name	Patient_address	Gender	Age	Telecome	Birthdate

1. POST - when FHIRID is not there then that request as a POST and we will create a new resource in the fhir server
2. PUT – and if we get the FHIR id th

The request will as PUT only.

Businees ID	FHIR ID	Organization Name

To build the references also

- For building references litteral reference we use the fhir prep step as well

- For example in the coverage.beneficiary reference we need to make a reference to patient profile

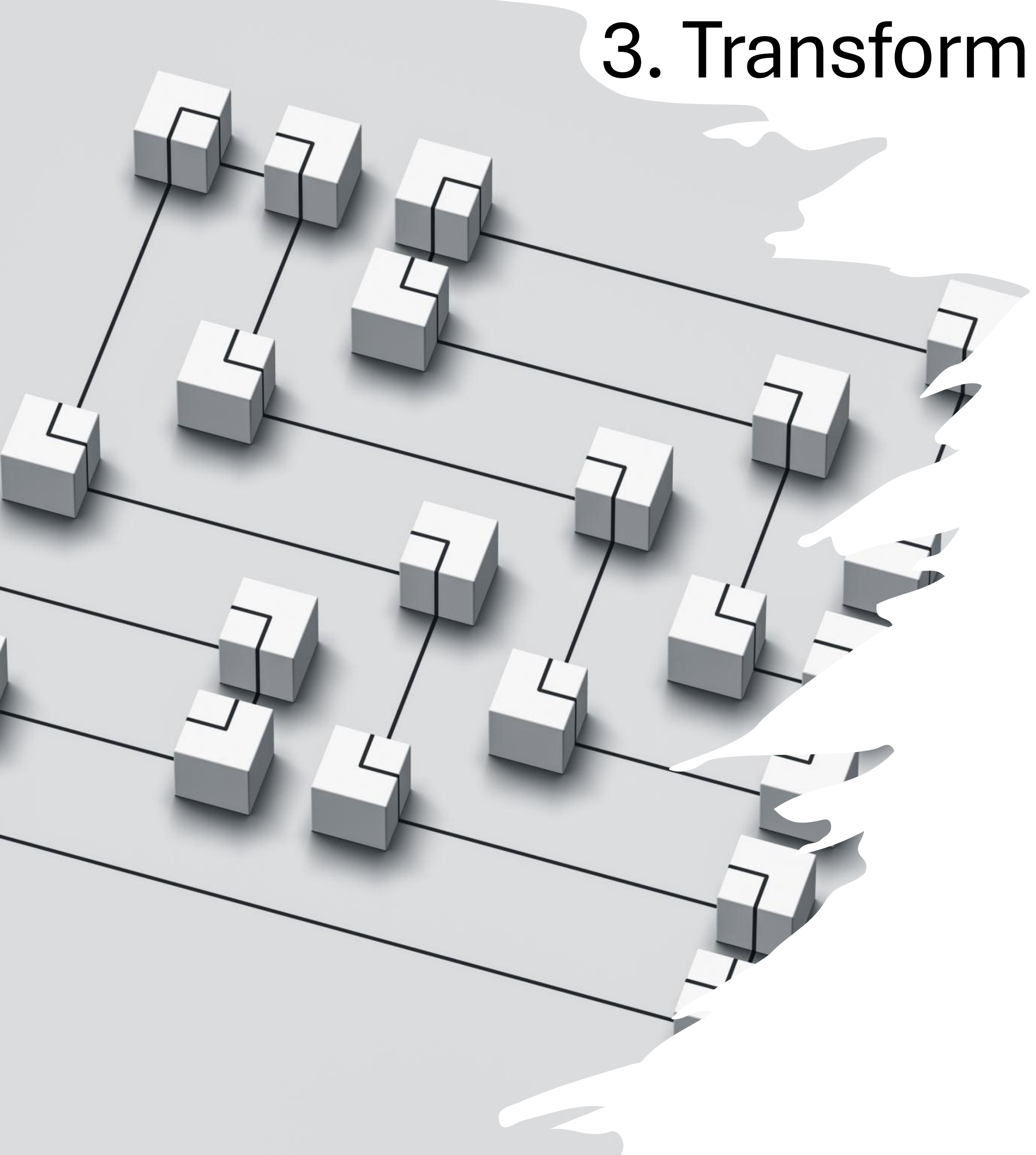
Data Ingestion Types

1. Historical Load

Manual Ingestion – Onshore Team – Srikanth and team

2. Daily Ingestion –

Auto Trigger (JWF scheduler) – where we ingest the data daily



3. Transform

Objective : In the FHIR Transform step, pre-processed data is converted into FHIR resources using predefined mapping rules. This step ensures that the data adheres to FHIR standards and the CARIN BB IG STU2.0 profiles.

Detailed Process

1. **Application of Mapping Rules**
   - Pre-processed records are transformed into FHIR resources using predefined mapping rules specified in the ONYX Data Guide (SAFHIR_Claims_STU2_Data_Guide.xlsx).
   - The mapping rules are implemented using Jinja templates, which facilitate the conversion of raw data into the structured format required by FHIR.
2. **Transformation into FHIR JSON**
   - The Jinja templates generate FHIR-compliant JSON records from the pre-processed data.
   - These JSON records represent various FHIR resource types such as Patient, Organization, Practitioner, Coverage, and ExplanationOfBenefit.
3. **Creation of FHIR Bundles**
   - The individual FHIR JSON records are then bundled into a collection known as a FHIR Bundle.
   - A FHIR Bundle groups multiple FHIR resources into a single entity, making it easier to manage and upload the data.

Jinja Template

Upload

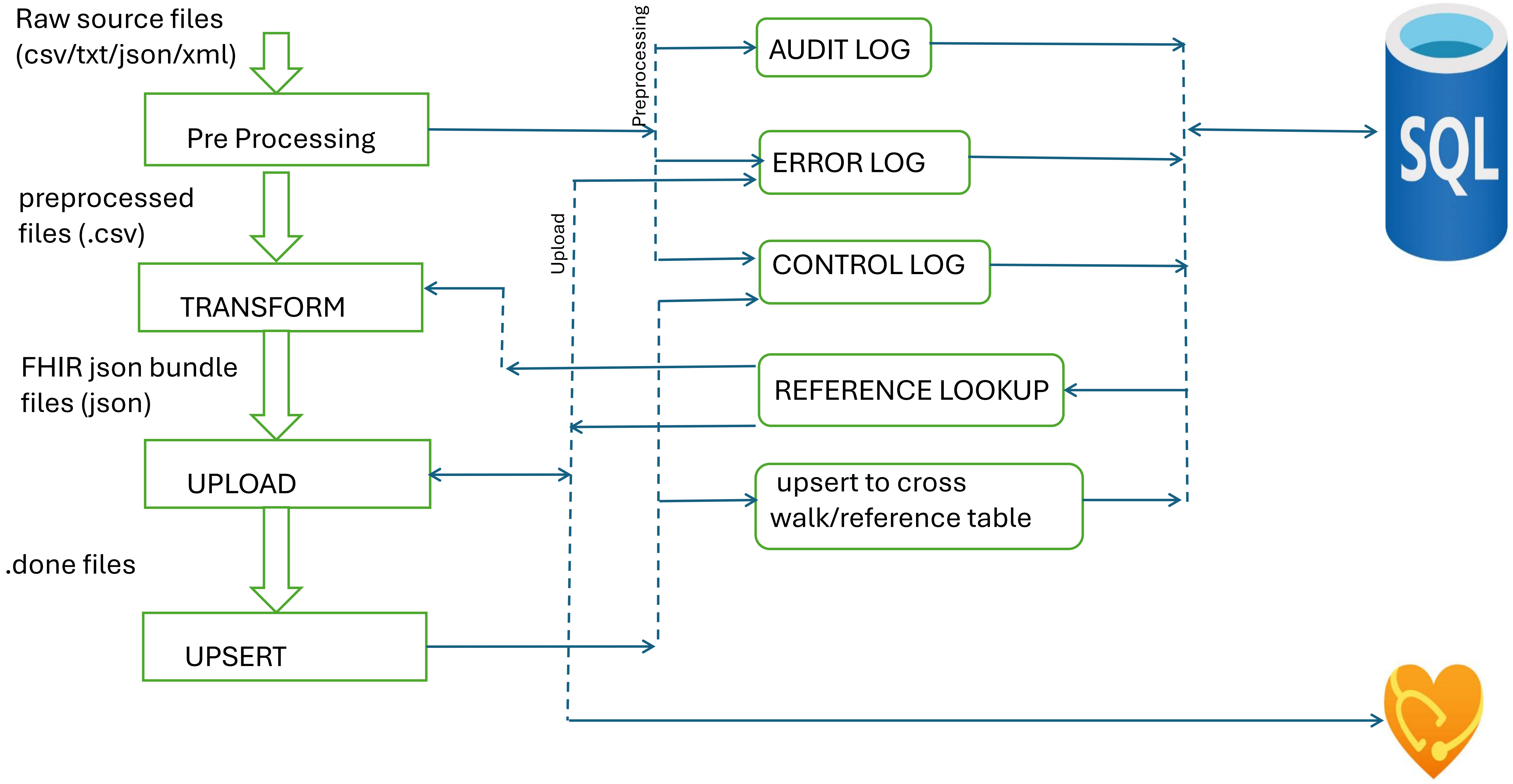
The transformed FHIR resources are uploaded to the Azure Health Data Service. This step involves handling large datasets efficiently and ensuring data integrity during the upload process.

Upsert

New records are inserted, and existing ones are updated in the Azure Health Data Service to maintain data consistency. This step includes conflict resolution and data consistency checks to ensure the accuracy and reliability of the data

- Post upload it will also maintain one cross walk table(reference table) with respect to each profile which will give 1 to 1 mapping for bid and fhirid
- Build the references

Business Identifier	FHIR id
Pat-100001	bjdofdosfjo304u304830280
Pat-100002	dfdjfooeroeuo34309-35-935-3


Raw source files (csv/txt/json/xml)
Pre Processing
preprocessed files (.csv)
TRANSFORM
FHIR json bundle files (json)
UPLOAD
.done files
UPSERT
Preprocessing
Upload
AUDIT LOG
ERROR LOG
CONTROL LOG
REFERENCE LOOKUP
upsert to cross walk/reference table
SQL

RAW DATA

FROM CLIENT SYSTEM -SFTP

DATA related to profiles - Patient, Organization, Practitioner, Coverage, ExplanationOfBenefit

File format - .CSV

INPUT 2 ONYX SYSTEM

CONVERT

PROCESS (ONYX SYSTEM) – AZURE DATA BRICK

GETTING THE SOURCE FILE

PROCESSING & TRANSFORMING to FHIR GROUP BUNDLE

UPLOAD TO FHIR STORE

FHIR REST API

INTEROPARABILITY

FHIR RESOURCE – Group

REST API ACESSESABLE – GET,PUT,POST,DELETE

STANDARD FOR DATA EXCHANGE DEFINED BY HL7

Infra Setup – DevOps Manual

Resource Group

Resource groups provide a logical container to manage and organize Azure resources, simplifying administration and enabling efficient resource management.

Azure Data Brick

Azure SQL Server

Azure Key Vault

Managed Resource Group in Azure?

When you create certain Azure services (like **Azure Databricks**, **AKS (Kubernetes)**, etc.), Azure automatically creates a **Managed Resource Group (MRG)** for you.

- This group is **managed by Azure**, not you.
- It contains **infrastructure resources** that your service needs (like load balancers, VMs, network interfaces, public IPs, etc.).
- You **should not modify or delete** anything inside it, because Azure controls it.
- Example: When you create an **Azure Databricks workspace**, it will automatically create a managed resource group like:

Vnet – Virtual Network

A **VNet** in Azure stands for **Virtual Network** — it's basically your own **private network** inside Azure's cloud.

Think of it as the Azure equivalent of a local area network (LAN) in a physical office:

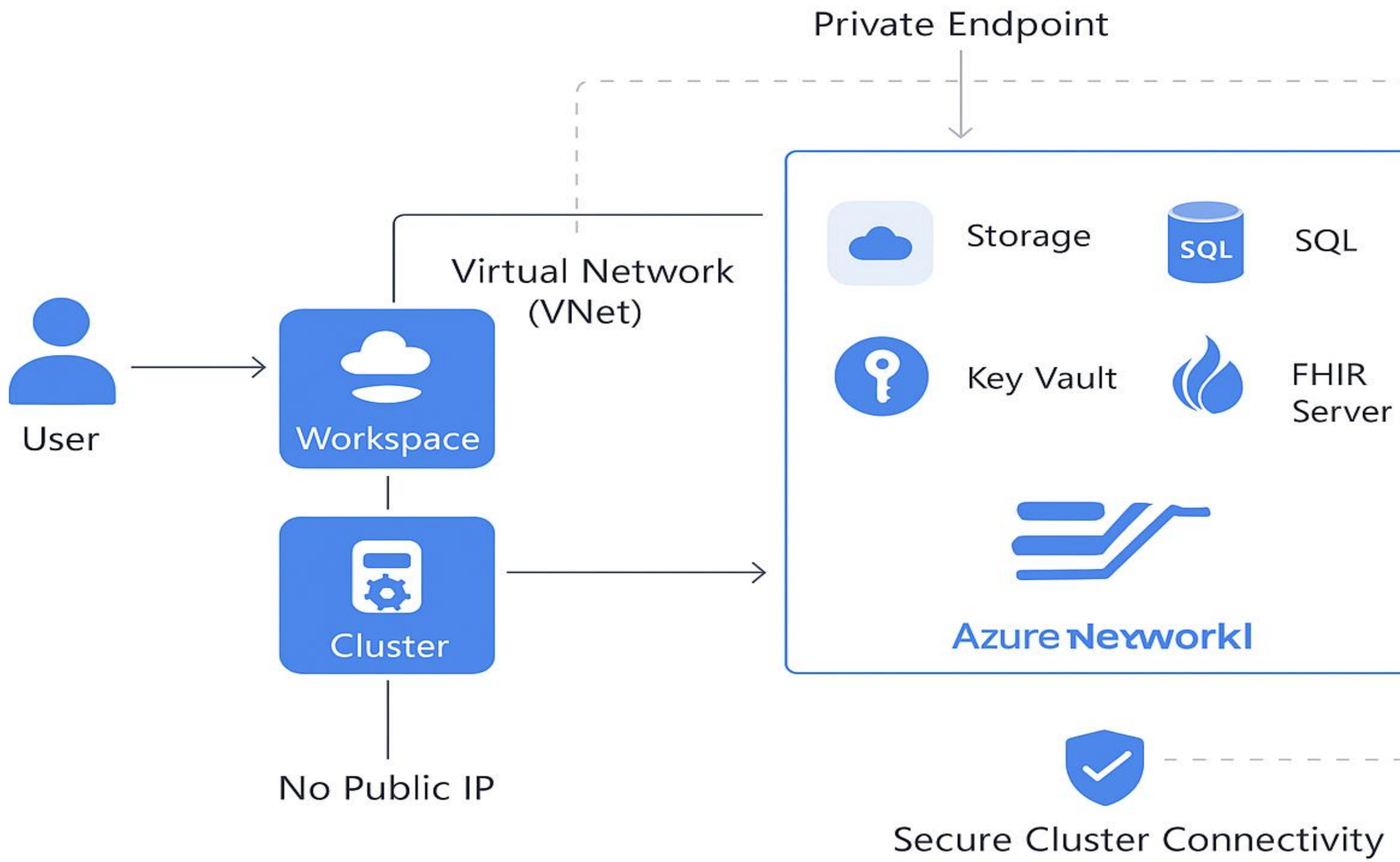
You can place different Azure resources (VMs, databases, Databricks clusters, etc.) inside it.

You control how they connect with each other and to the internet.

You can make it **isolated** from the public internet or allow **secure, restricted access**.

- **Key points about VNet:**
  - **Segregation:** Keeps your resources safe and organized.
  - **Subnets:** You can break a VNet into smaller sections for better traffic control.
  - **Security:** You can use **Network Security Groups (NSGs)** and firewalls to allow/block specific traffic.
  - **Connectivity:** VNets can connect to other VNets, on-premises networks (via VPN or ExpressRoute), or the public internet.
- **Example for Databricks ETL pipeline:** If you have:
  - Azure SQL Server,
  - Azure Key Vault,
  - Azure FHIR Server,
  - Azure Databricks

  you can put them all inside the **same VNet** so they can talk to each other privately without sending data over the public internet.


Private Endpoint
Storage
SQL
Virtual Network
(VNet)
User
Workspace
Key Vault
FHIR
Server
Cluster
Azure Neworkl
No Public IP
Secure Cluster Connectivity

When we enable **No Public IP:**

Cluster nodes have **only private IPs** within a VNet subnet.

Communication between:

User → Workspace
Workspace → Cluster
Cluster → Azure services (like Blob Storage, Key Vault, SQL Server, FHIR Server)

happens **through private endpoints** or Azure Private Link.

The **Databricks control plane** (managed by Microsoft) talks to the **data plane** (your cluster) via a **relay** service—no internet traffic to your cluster.

Wheel package

•**Reusability** –
Instead of copying all the python scripts .py files everywhere, you just build once and install the wheel wherever needed.

•**Version Control** –
You can tag versions (my_package-0.1.0-py3-none-any.whl) and easily upgrade/downgrade.

•**Cluster-Wide Installation** –
In Databricks, once uploaded, the wheel can be installed at cluster level
all notebooks can import without path hacks.

•**Cleaner Code Imports** –
Instead of sys.path.append() tricks, you just do:
from commonlibrary import preprocfunctions as pf
naturally, as we import the generic packages because the package is available at cluster level.

•**Portability** –
The same wheel can be installed locally, in Dev, UAT, and Prod without code duplication.

•**Faster Deployment** –
Just rebuild the wheel when your library changes and re-install on the cluster.

Steps to follow

https://medium.com/@jonathan.hoffman91/a-step-by-step-guide-to-building-python-wheels-4ed8160809a2

Validation- QA

Objective : Here we validate the counts for the ingestion against fhir pass

Job summary details count and checking the FHIR pass count manually if counts are in sink then we are good.

1. Counts – 7 Profiles – search query to check how

Many patient resources got ingested with todays

Date

2. Data check – here we try to check if all the data from the source files got mapped to fhir json resource or not.

Q & A