RosterSource Architecture

RosterSource

Getting physician rosters from the source

Table of Contents

Contact Us:

Gunjan Siroya
Gunjan.siroya@Netspective.com
+1 (949) 236 7436

Lori Serafin
lserafin@humana.com
+1 (502) 580 2554

RosterSource Vision - Business Objectives

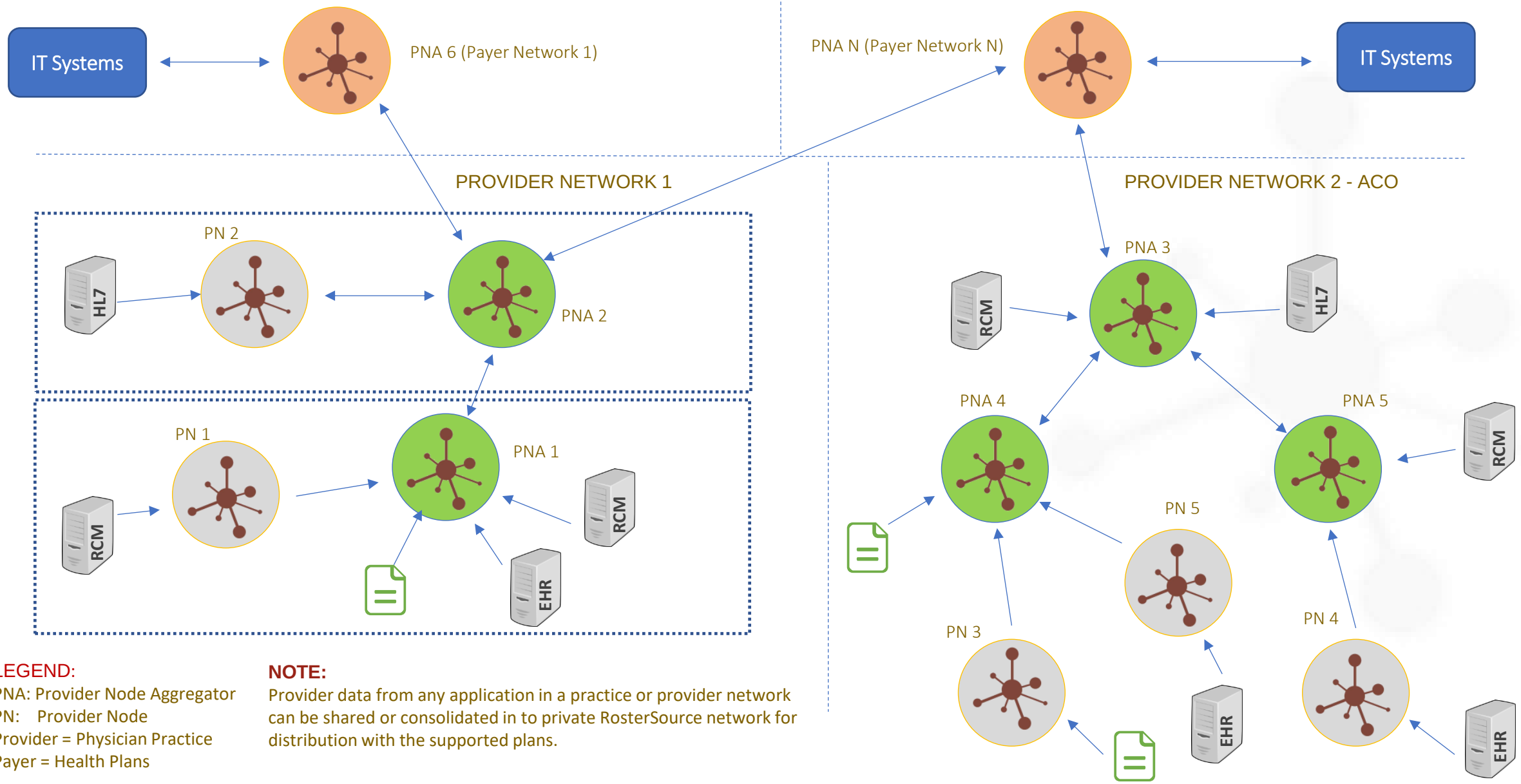
Vision: Consort with the brightest minds to collaboratively build an awe-inspiring solution that addresses the healthcare industry's directory data quality deficiencies.

- Make it easier for physician practices to communicate roster changes in a trusted and secure manner, without adding to the practice's administrative burden.
- Integrate with physician practices to obtain source-of-truth data, no matter where the data resides
- Create operational efficiencies and savings for both data producer and data consumer.
- The resulting solution is scalable, inspires enthusiastic adoption and is a springboard serving other business cases for value-adding services and products.

Objective: Develop an open source solution that will enable data consumers to receive physician demographic updates near real-time via an online federated roster and attain

- **Highest accuracy** – source of truth for public, directory-related information
- **Least disruption** – no interruption in Provider workflow, annoyance avoidance
- **Regulatory compliance** – fulfill CMS objectives for contact, effective process and consumer access

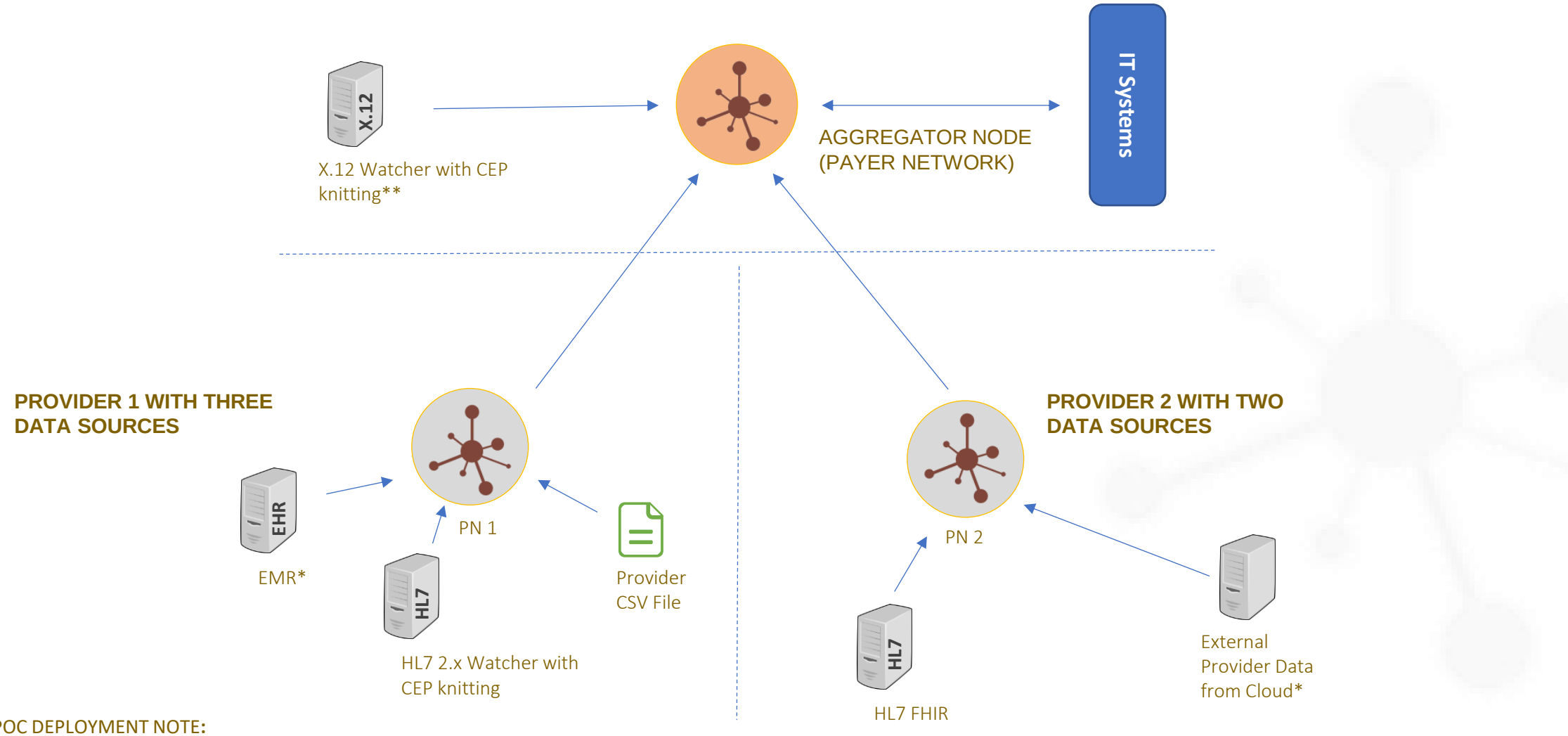
RosterSource Vision - Network Layout



LEGEND:
PNA: Provider Node Aggregator
PN: Provider Node
Provider = Physician Practice
Payer = Health Plans

NOTE:
Provider data from any application in a practice or provider network can be shared or consolidated in to private RosterSource network for distribution with the supported plans.

RosterSource Vision - Network Layout Use Case

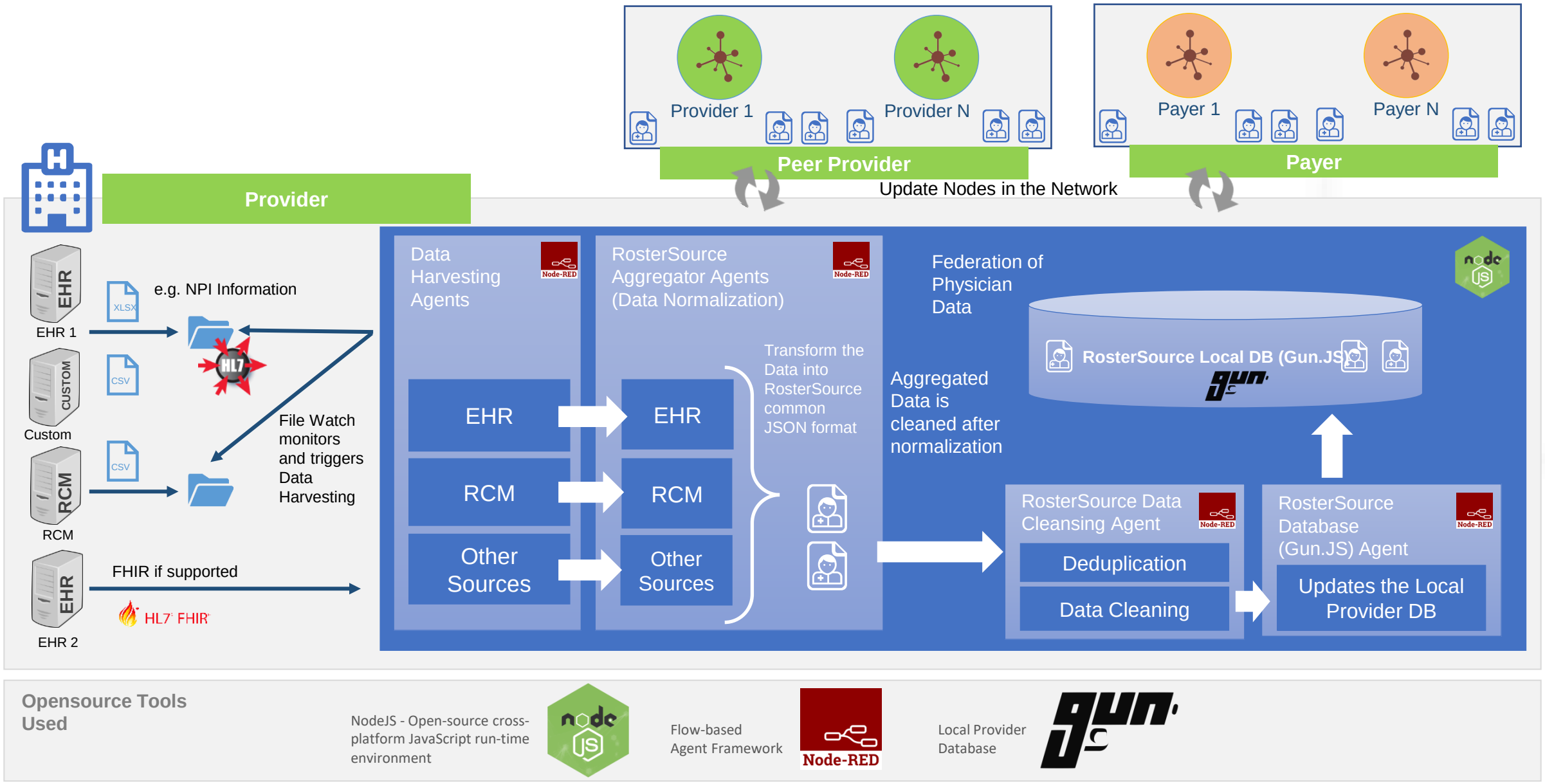


POC DEPLOYMENT NOTE:
Node, Aggregator Node are copies of the same RosterSource software but configured differently depending on the role they need to play and policies of the organizations

*PoC TEST DATA NOTE:
Initial PoC will leverage OpenEMR product suite and NPPES test data.

**Optional

RosterSource Vision - Architecture



Opensource Tools Used

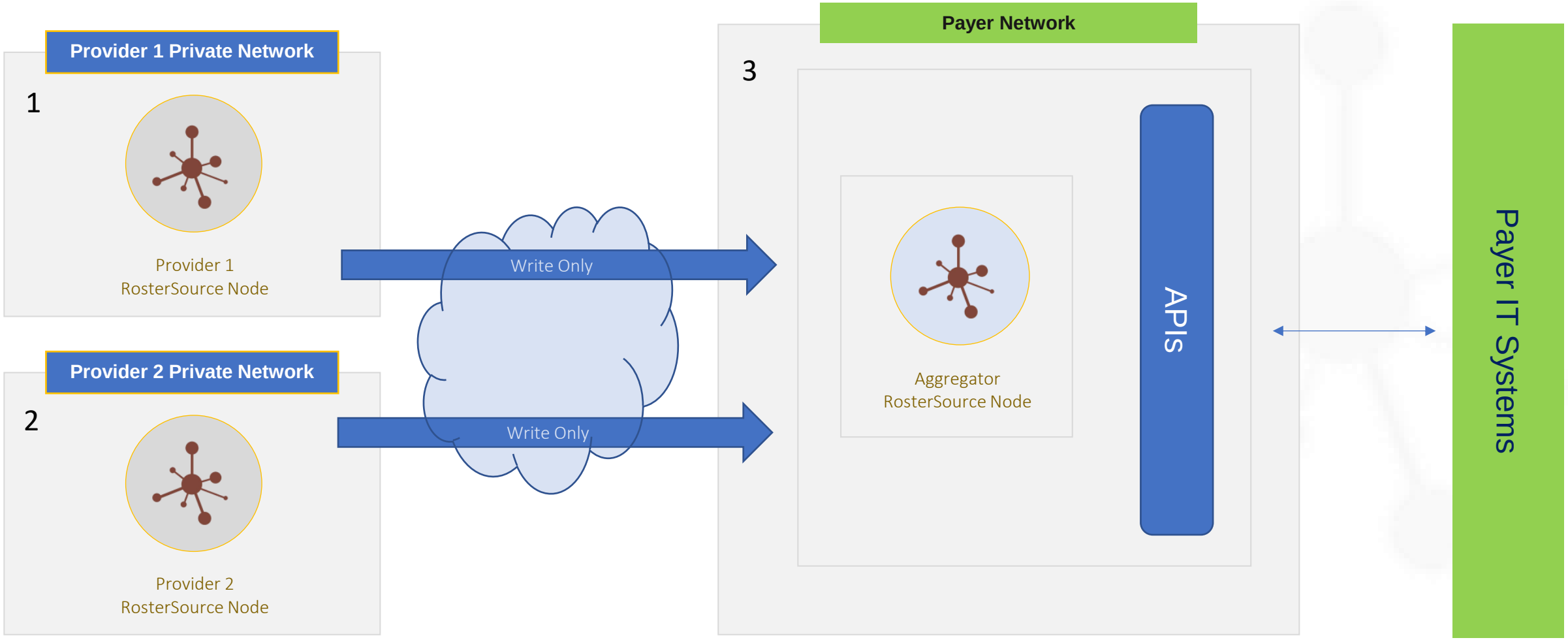
NodeJS - Open-source cross-platform JavaScript run-time environment

Flow-based Agent Framework

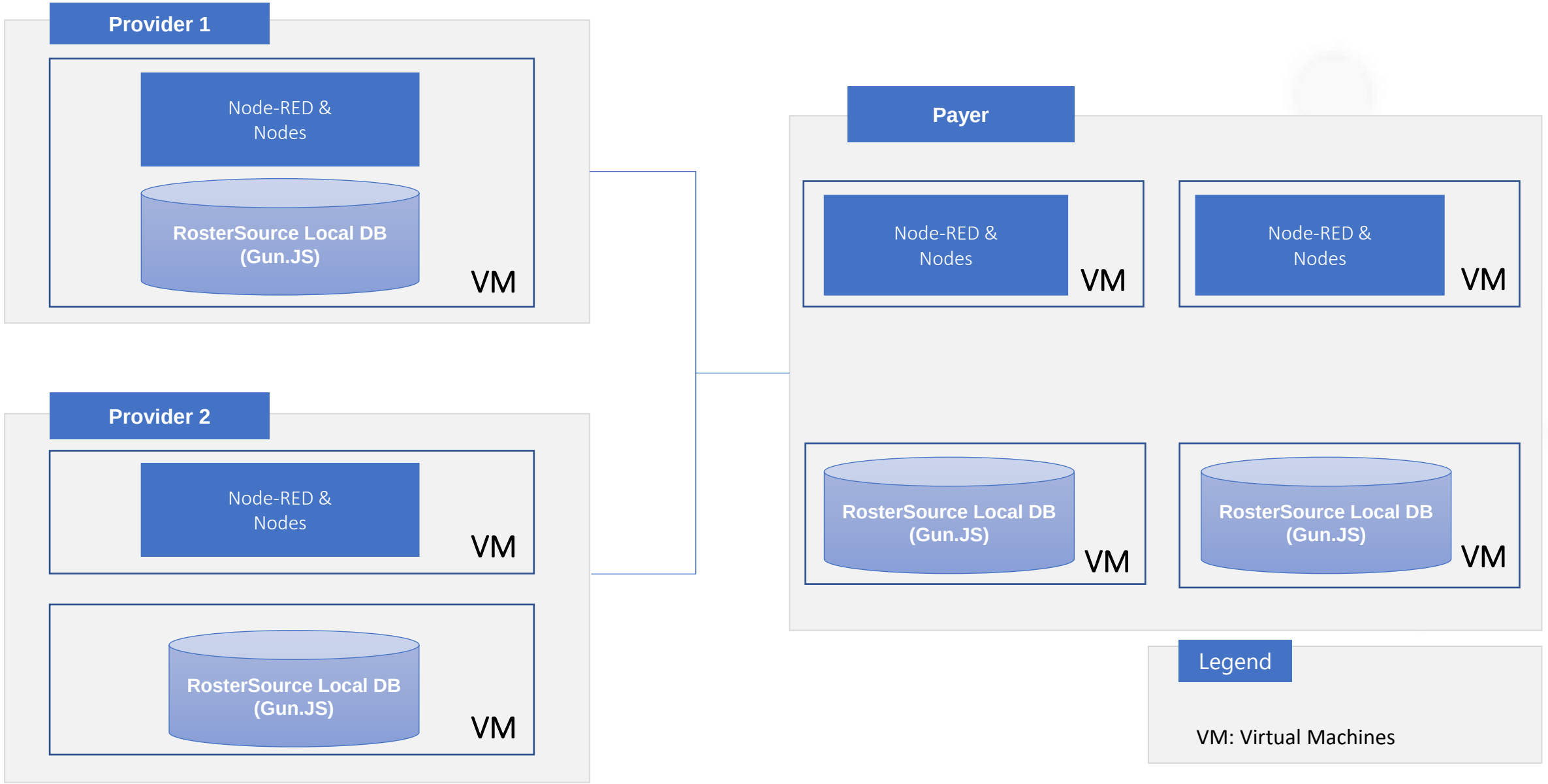
Local Provider Database

RosterSource Vision - Deployment Architecture

Gun.JS can be configured to accept data or only push the data. This will allow the network to be configured to only send data out but not receive, or send and receive with only configuration updates and no code changes. For POC it will be only Push from Providers.



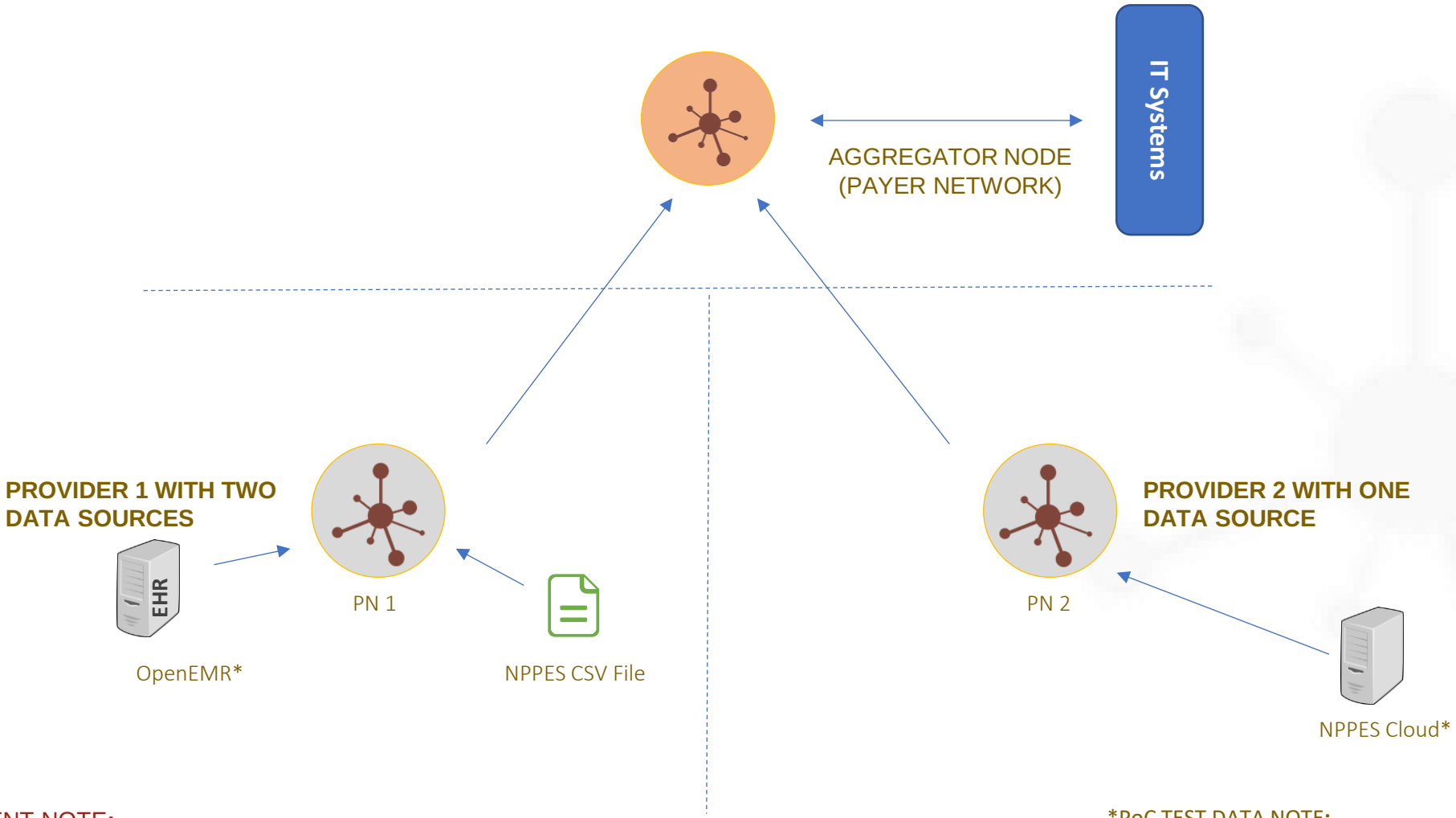
RosterSource Vision - Deployment Architecture



RosterSource

Proof of Concept (PoC)

Proof of Concept (PoC) - Network Diagram

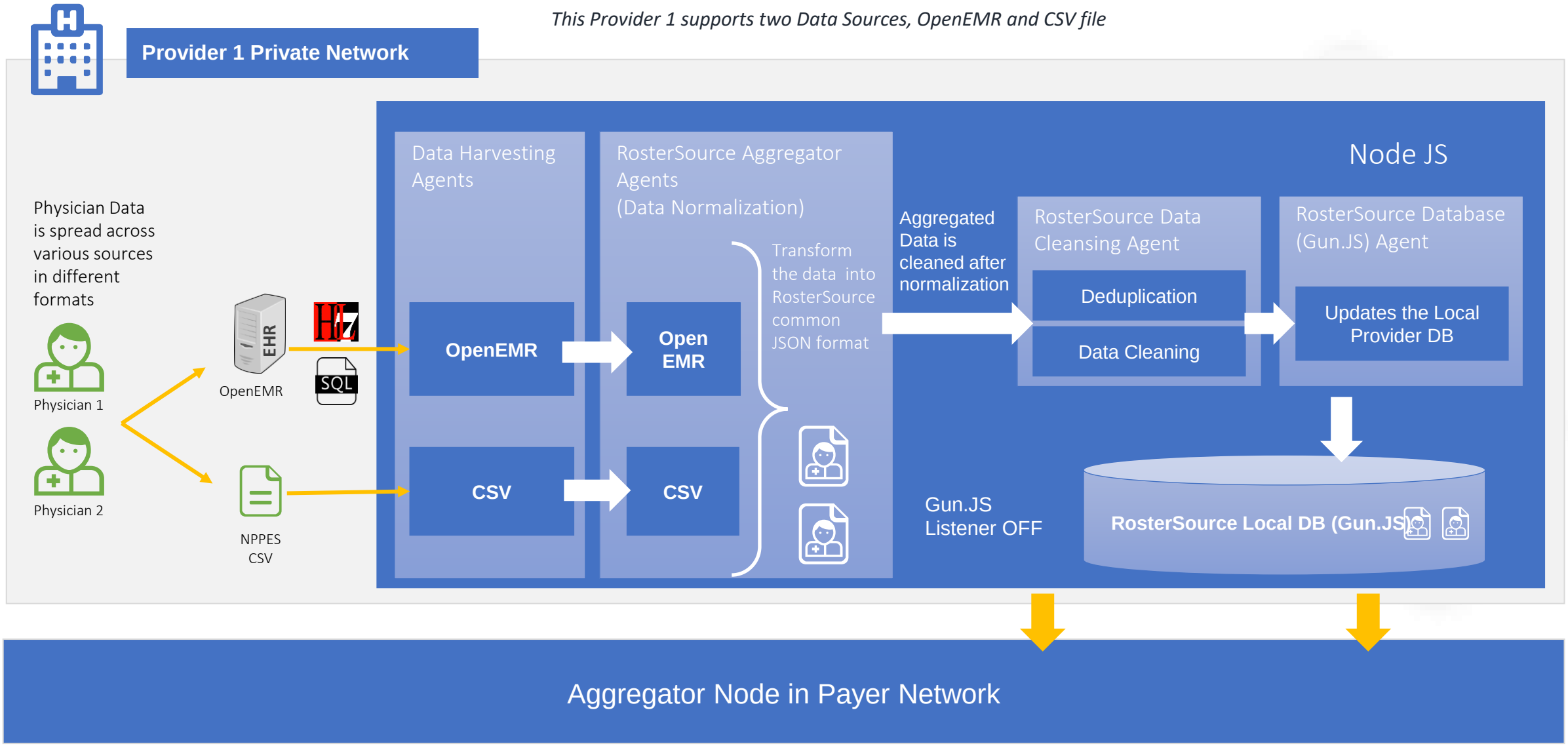


POC DEPLOYMENT NOTE:
Node, Aggregator Node are copies of the same RosterSource software but configured differently depending on the role they need to play and policies of the organizations

*PoC TEST DATA NOTE:
Initial PoC will leverage OpenEMR product suite and NPPES test data.

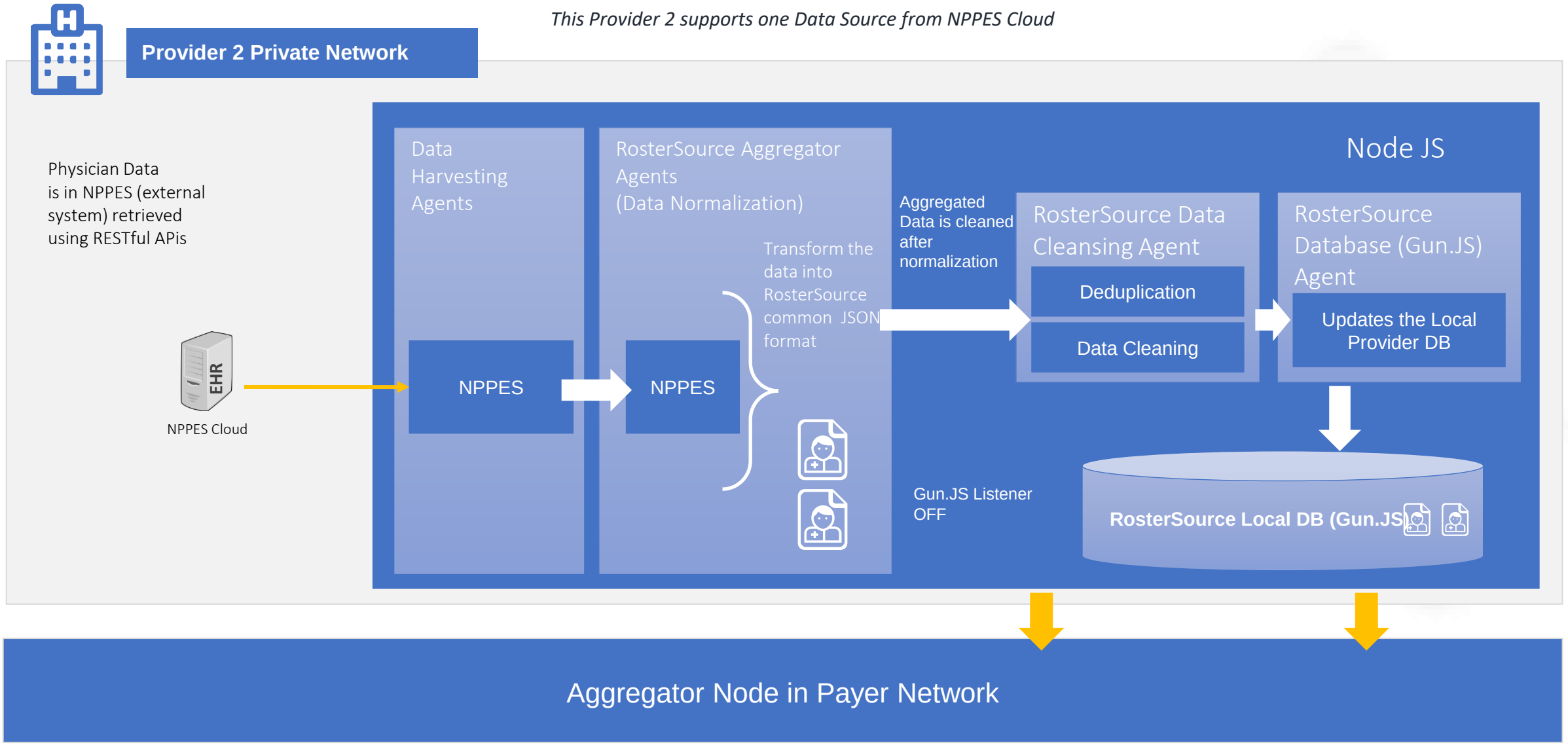
PoC - Provider 1 Private Network Architecture

This Provider 1 supports two Data Sources, OpenEMR and CSV file



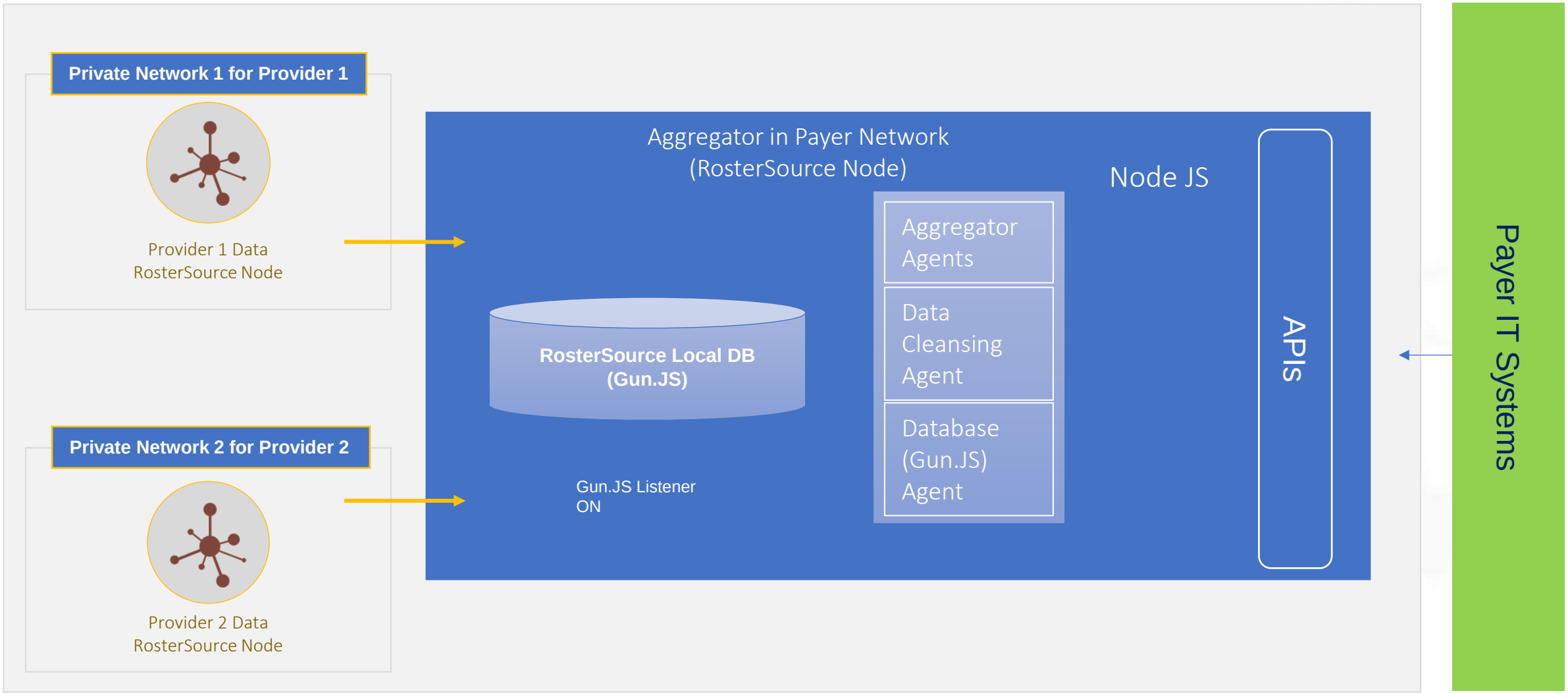
PoC - Provider 2 Private Network Architecture

This Provider 2 supports one Data Source from NPPES Cloud



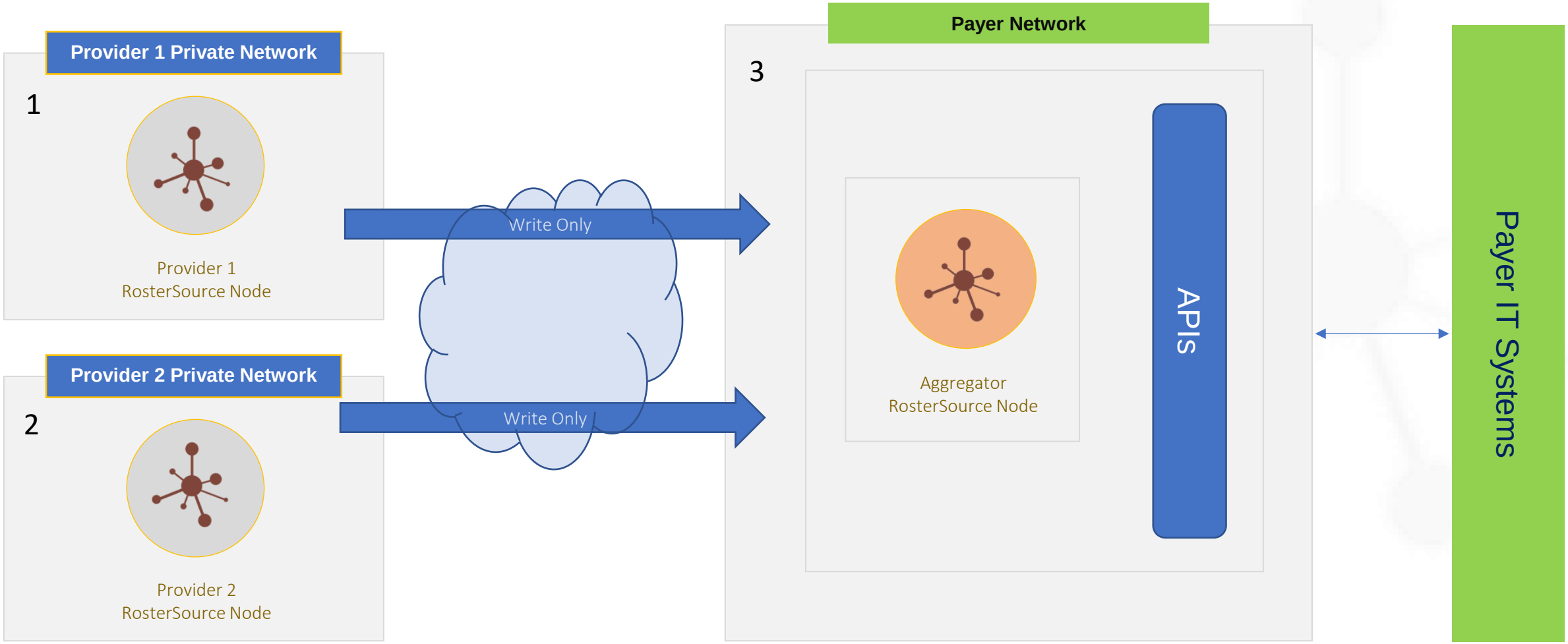
PoC – Aggregator in Payer Network Architecture

Gun.JS's Listener is configured to access the data from Providers. Since there is no peer relationship established with Providers, no data is pushed between Providers



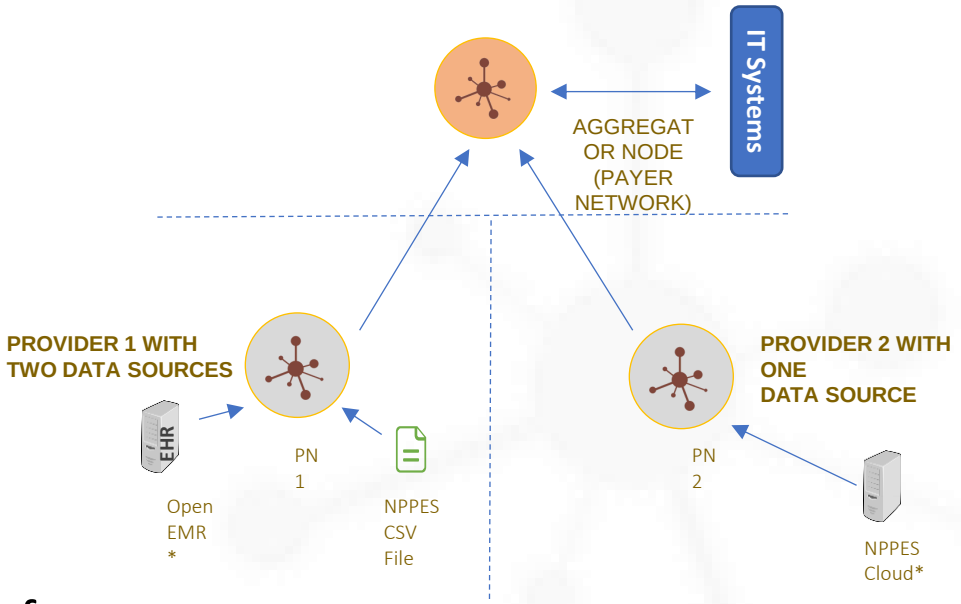
PoC – RosterSource Deployment Architecture

Gun.JS can be configured to accept data or only push the data. This will allow the network to be configured to only send data out but not receive, or send and receive with only configuration updates and no code changes. For POC it will be only Push from Providers.



PoC – RosterSource Deployment Approach

- 3 different deployment environments with common software components (Node)
  - 2 for Providers
  - 1 for Payer
- Every deployment environment will be configured differently depending on the location of the node in the hierarchy and policy – Node or Aggregator
  - Node 1: In Node mode for Provider 1 Network
  - Node 2: In Node mode for Provider 2 Network
  - Node 3: In Aggregator mode in Payer Network
- Monitoring and configuration software will ease the burden of deployments by mapping data with their sources and their status



Technical Environment

List of tools/ software needed for RosterSource

- The RosterSource Application is based on NodeJS framework.
- RosterSource internally uses Node-Red for creating various agents (e.g. Data Harvesting Agents, Aggregator Agents etc.)
- RosterSource internally uses GunJS as its local data storage.
- Both Node-Red and GunJS will be installed when the RosterSource application is installed.
- Pre-requisites for installing RosterSource - NodeJS (https://nodejs.org/en/, https://github.com/nodejs/node)
- Libraries used by RosterSource
  - Gun JS (http://gun.js.org/, https://github.com/amark/gun)
  - Node-Red (https://nodered.org/, https://github.com/node-red/node-red)
- Github is used for version control.

Glossary

Data Harvesting Agents Definition

- Specific to each Data Source format, there will be a Data Harvesting Agent which pulls data from the source and provides it to the respective RosterSource Aggregator Agent.
- For example, there will be an **FHIR Data Harvesting Agent** which is capable of pulling data from a FHIR interface.
- Creating multiple Data Harvesting Agents (e.g. FHIR Data Harvesting Agent for OpenEMR, SQL Data Harvesting Agent for OpenEMR etc. for the initial PoC) will be required for each of the Data and Data Source formats.
- The output format of Data Harvesting Agents will be JSON. RosterSource Aggregator Agents process this JSON further.

RosterSource Aggregator Agents Definition

- RosterSource Aggregator Agents receive the data from the Data Source as JSON
- RosterSource Aggregator Agents transform incoming JSON data in custom JSON format to RosterSource Common JSON format.
- Specific to each Data Source format, there will be a RosterSource Aggregator Agent to transform the raw data that comes in from Data Harvesting Agents to RosterSource Common JSON format.
- For example, a **RosterSource Aggregator** can filter out unwanted fields from the input and transform the data to RosterSource Common JSON format by renaming the fields as required.

RosterSource Cleansing Agent Definition

- Once the RosterSource Aggregator Agents transform the data received to the Common JSON Format, it requires cleaning.
- The output of RosterSource Aggregator Agents is sent to RosterSource Cleansing Agent for performing cleaning operations.
- The cleaning process is common for all data sources.
- One example of the functionality of the **RosterSource Data Cleansing Agent** is the deduplication of Data in RosterSource Common JSON format.

RosterSource Database (Gun.JS) Agent Definition

- Once the data is cleansed, it needs to be stored in the local provider database (GunJS).
- RosterSource Database Agent pushes the data received from RosterSource Cleansing Agent to the local provider DB in RosterSource Common JSON format from where it gets federated.