Protocol Team – Technical Stack and Workflow Overview

# 1. Technologies and Their Performance

This section outlines the technologies selected by the Protocol Team to support scalable and high-throughput communication between buyer and seller apps, inspired by the ONDC model.

|  |  |  |
| --- | --- | --- |
| Technology | Purpose | Performance/Scalability |
| Node.js + Express | To develop asynchronous REST APIs for the protocol gateway. | Handles 1,000+ requests/sec with clustering and load balancing. |
| Redis | In-memory cache for session and message state management. | Supports ~100,000 operations/sec, very low latency. |
| MongoDB | NoSQL database for catalog, orders, and session data. | Handles 1,000+ transactions/sec with sharding and indexing. |
| HMAC/RSA Security | Ensures message authenticity, integrity, and trust between apps | HMAC is fast for internal use; RSA offers high-trust security for public, open networks with scalable public key infrastructure. |
| PostgreSQL | Relational database for logging and persistent order records. | Efficient with ACID compliance; supports 1000+ concurrent queries. |
| NATS / Kafka | Message queue for async and real-time communication. | Kafka handles 1M+ messages/sec; NATS is lightweight and fast for microservices. |
| Axios | HTTP client to call external APIs between services. | Lightweight and promises-based; suitable for scalable communication. |
| Beckn Protocol JSON Format | Standard protocol for interoperability. | Standardized JSON schema ensures clear, cross-platform messaging. |

# 2. Workflow Architecture

The protocol team acts as the bridge for communication between Buyer and Seller apps. Below is a step-by-step of the transaction lifecycle.

1. Buyer App initiates a /search request to the Protocol Layer.  
2. Protocol forwards request to registered Seller Apps.  
3. Seller Apps respond via /on\_search with catalog data.  
4. Protocol returns catalog options to Buyer App.  
5. Buyer sends /select and /order to place an order.  
6. Protocol forwards order to Seller App and waits for /on\_order confirmation.  
7. Status and updates are handled via /status and /on\_status.

# 3. Integration Requirements from Buyer and Seller Teams

From Buyer App Team:  
- APIs: /search, /select, /order, /status  
- Format: JSON messages aligned with Beckn Protocol  
- Metadata: User ID, session info, location  
  
From Seller App Team:  
- APIs: /on\_search, /on\_select, /on\_order, /on\_confirm  
- Real-time inventory and availability  
- Ability to confirm or reject order

# 4. Free vs Paid – Features & Power Comparison

|  |  |  |
| --- | --- | --- |
| **Technology** | **Free Version (Power & Features)** | **Paid Version (Power & Features)** |
| Node.js / Express | Full feature set; 10K+ concurrent connections; Open Source. | N/A – Hosting on cloud (e.g. AWS, Heroku) may incur cost. |
| Redis | Core Redis: 1M+ ops/sec, single node, no persistence. | Redis Enterprise/Cloud: HA, backups, scaling, multi-region support. |
| MongoDB | Community Edition: 1K+ writes/sec; manual sharding. | MongoDB Atlas/Enterprise: auto-scaling, monitoring, backups, global clusters. |
| PostgreSQL | Full DB engine, 5000+ TPS, self-hosted. | Managed cloud DBs: backups, scaling, replication (RDS, Azure, etc.). |
| Kafka | Apache Kafka OSS: 100K+ messages/sec, manual management. | Confluent Cloud: schema registry, auto-scaling, security, support. |
| NATS | Sub-ms latency; core pub-sub for microservices. | JetStream/Cloud: persistence, RBAC, monitoring dashboard. |
| Axios | Completely free and open-source. | N/A |
| Beckn Protocol | Free and open JSON schema. | N/A |

# 5. System Architecture Diagram

The following diagram illustrates the communication flow between the Buyer App, Protocol Layer, and Seller App.

