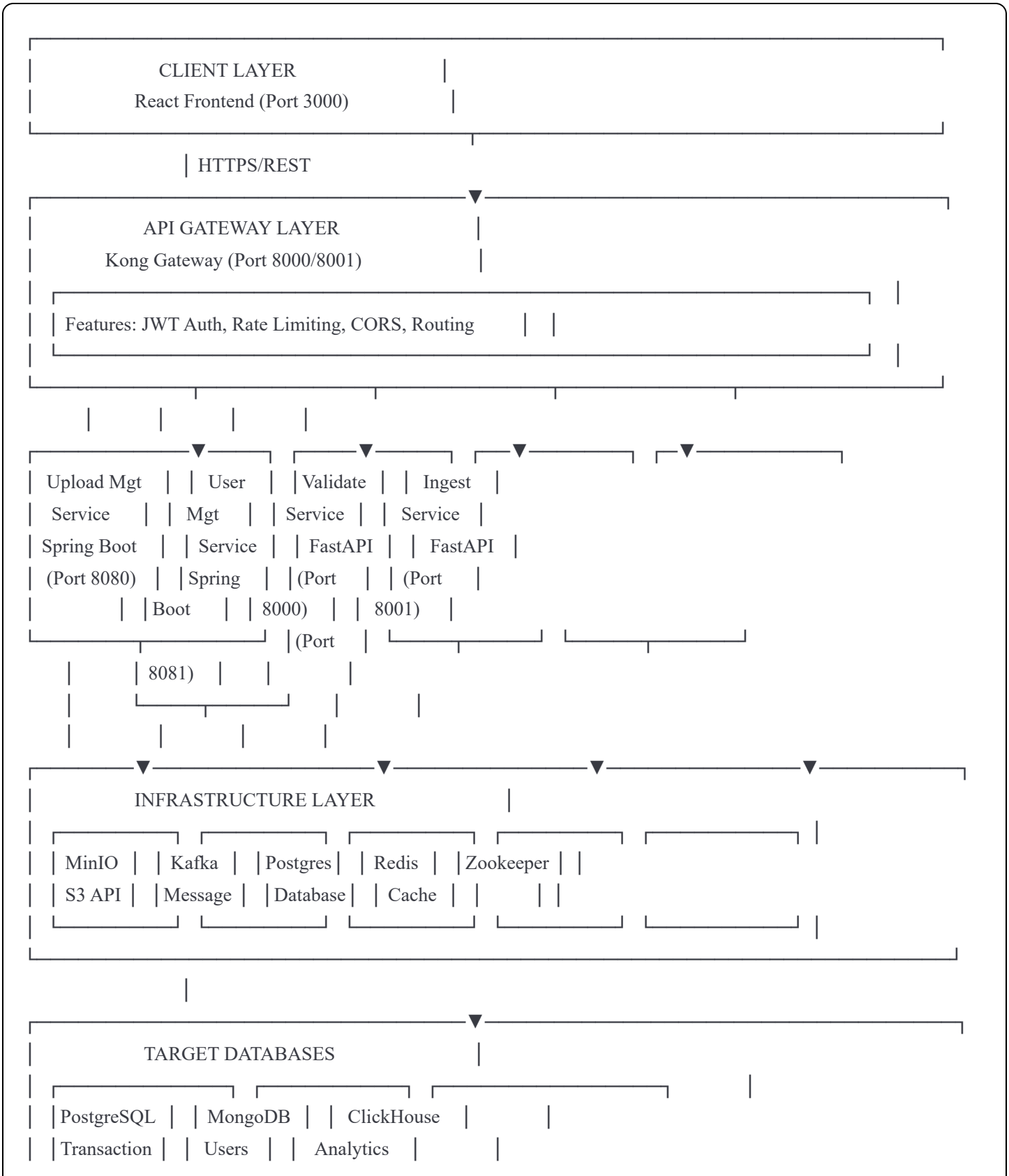
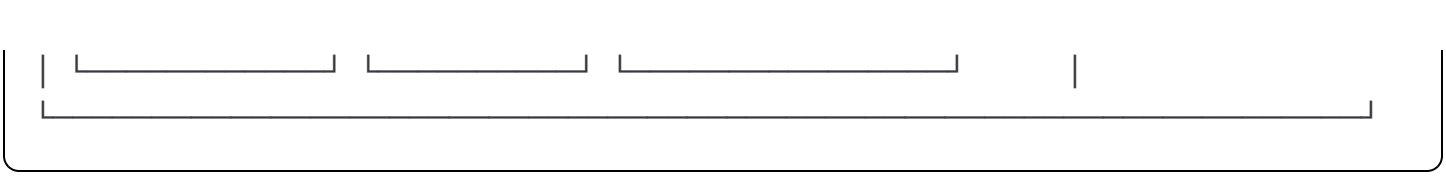
TÀI LIỆU KIẾN TRÚC HỆ THỐNG UPLOAD ENTERPRISE

1. TỔNG QUAN KIẾN TRÚC

1.1 Kiến trúc Microservices





1.2 Các Thành Phần Chính

Frontend Layer

- **React Application**: SPA với hooks và context
- **API Service**: Axios client với interceptors
- **State Management**: React useState/useReducer
- **Authentication**: JWT token storage

API Gateway Layer

- **Kong Gateway**:
  - Load balancing
  - Rate limiting (100 req/min upload, 50 req/min user)
  - JWT authentication
  - CORS handling
  - Request/Response transformation
  - Logging và metrics

Backend Services

1. Upload Management Service (Java Spring Boot)

- Quản lý metadata upload
- Generate presigned URLs từ MinIO
- Xác nhận upload completion
- Publish events đến Kafka
- PostgreSQL để lưu upload records

2. User Management Service (Java Spring Boot)

- Authentication (Login/Register)
- JWT token generation

- User và Role management
- PostgreSQL để lưu user data

3. Validation Service (Python FastAPI)

- Consume Kafka events (upload.completed)
- Download file từ MinIO
- Validate schema, data quality
- Support CSV, JSON, Parquet
- Store validation results
- Publish validation events

4. Ingest Service (Python FastAPI)

- Consume Kafka events (validation.completed)
- Download validated files
- Transform data format
- Multi-database connectors:
  - PostgreSQL (transactions)
  - MongoDB (user profiles)
  - ClickHouse (analytics)
- Celery tasks cho async processing

Infrastructure Layer

1. MinIO (Object Storage)

- S3-compatible API
- Presigned URLs cho direct upload
- Bucket: enterprise-uploads
- Ports: 9000 (API), 9001 (Console)

2. Kafka (Message Broker)

- Event-driven architecture
- Topics:

- upload.completed
- validation.completed
- ingestion.completed

- Zookeeper cho coordination

3. PostgreSQL

- upload_db: Upload metadata
- user_db: User data
- Multiple instances cho target databases

4. Redis

- Cache layer
- Celery backend
- Session storage

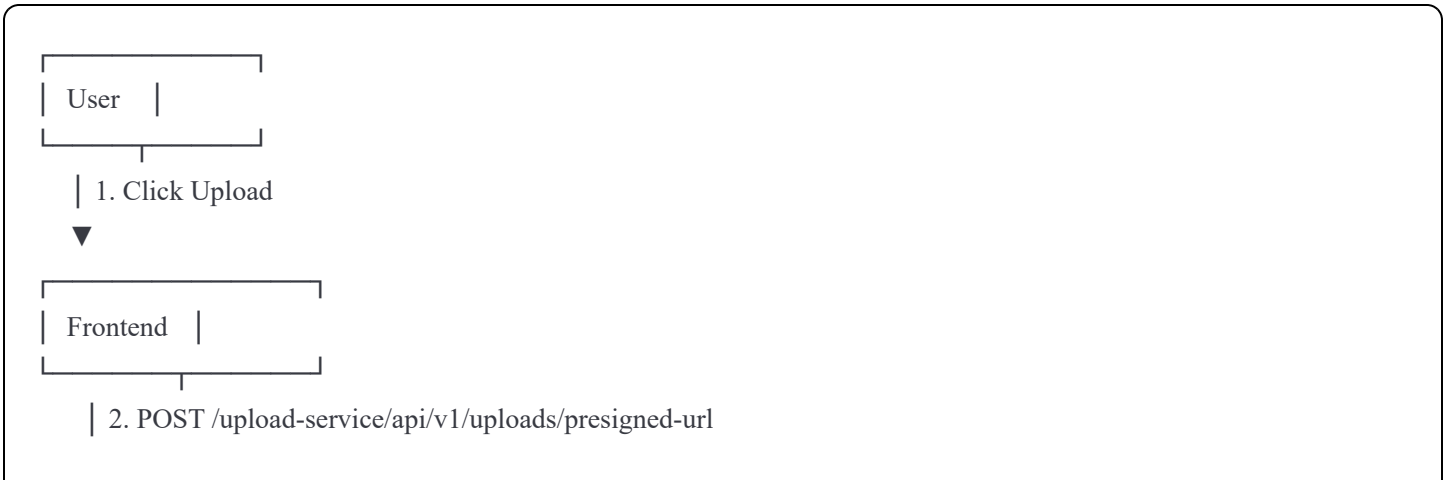
Target Databases

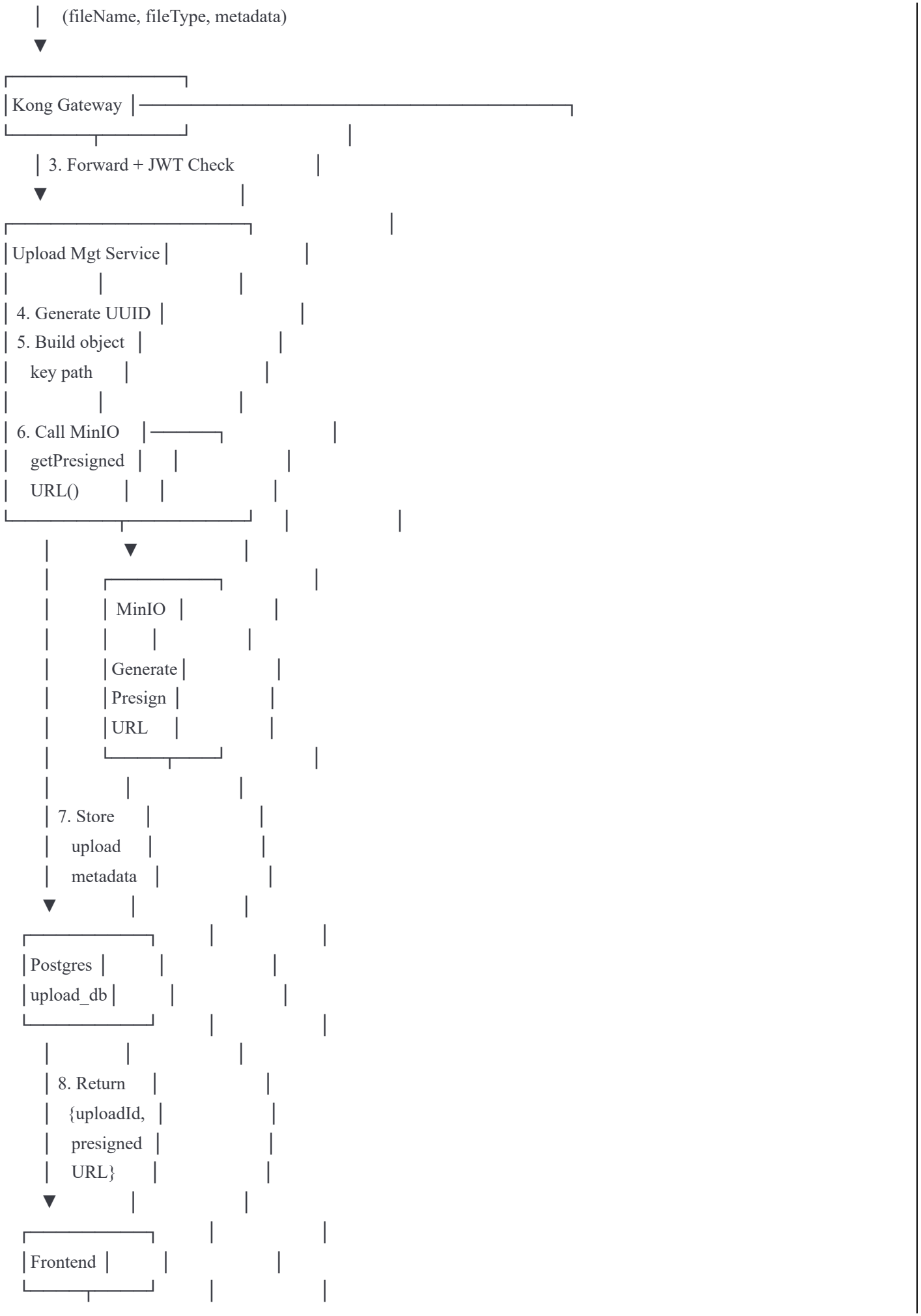
- **PostgreSQL**: Transactional data (ACID)
- **MongoDB**: Document storage (flexible schema)
- **ClickHouse**: OLAP analytics (columnar)

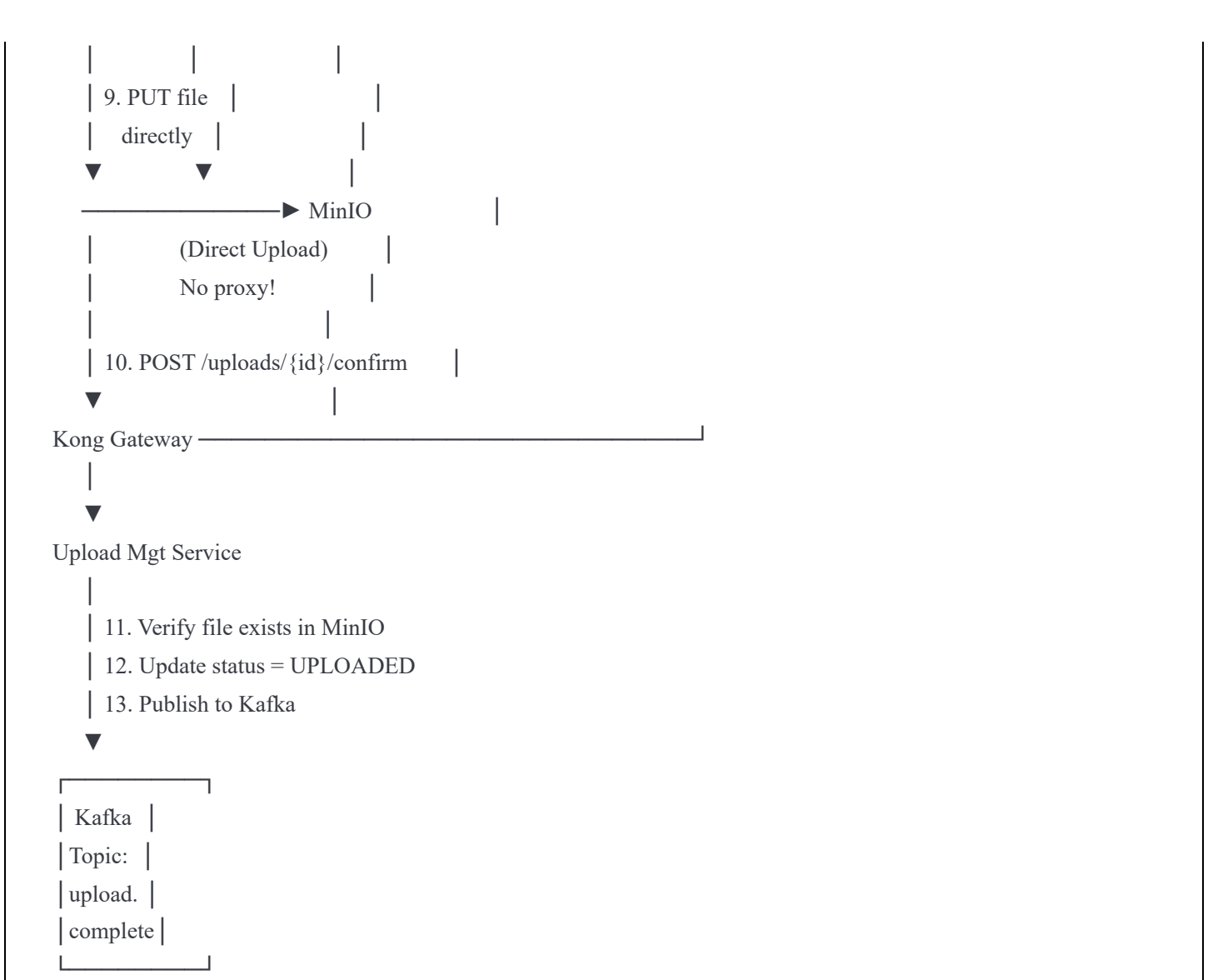
---

2. LUỒNG DỮ LIỆU CHI TIẾT

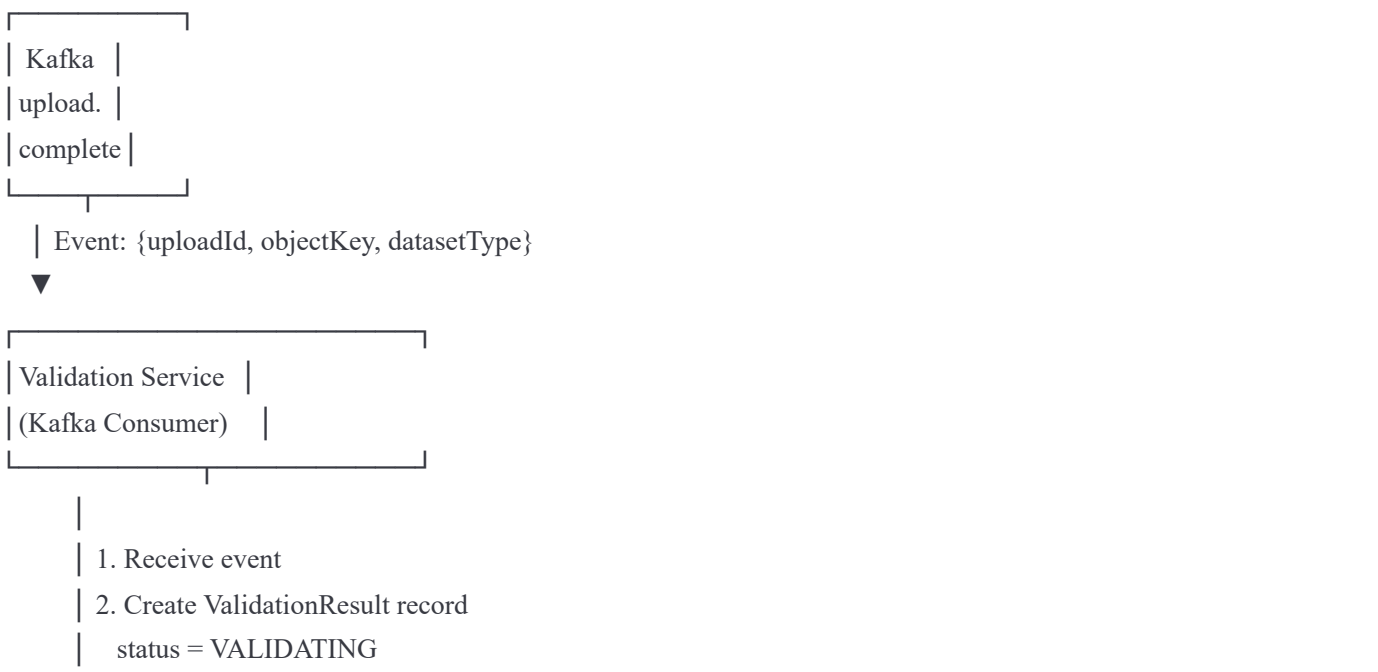
2.1 Upload Flow

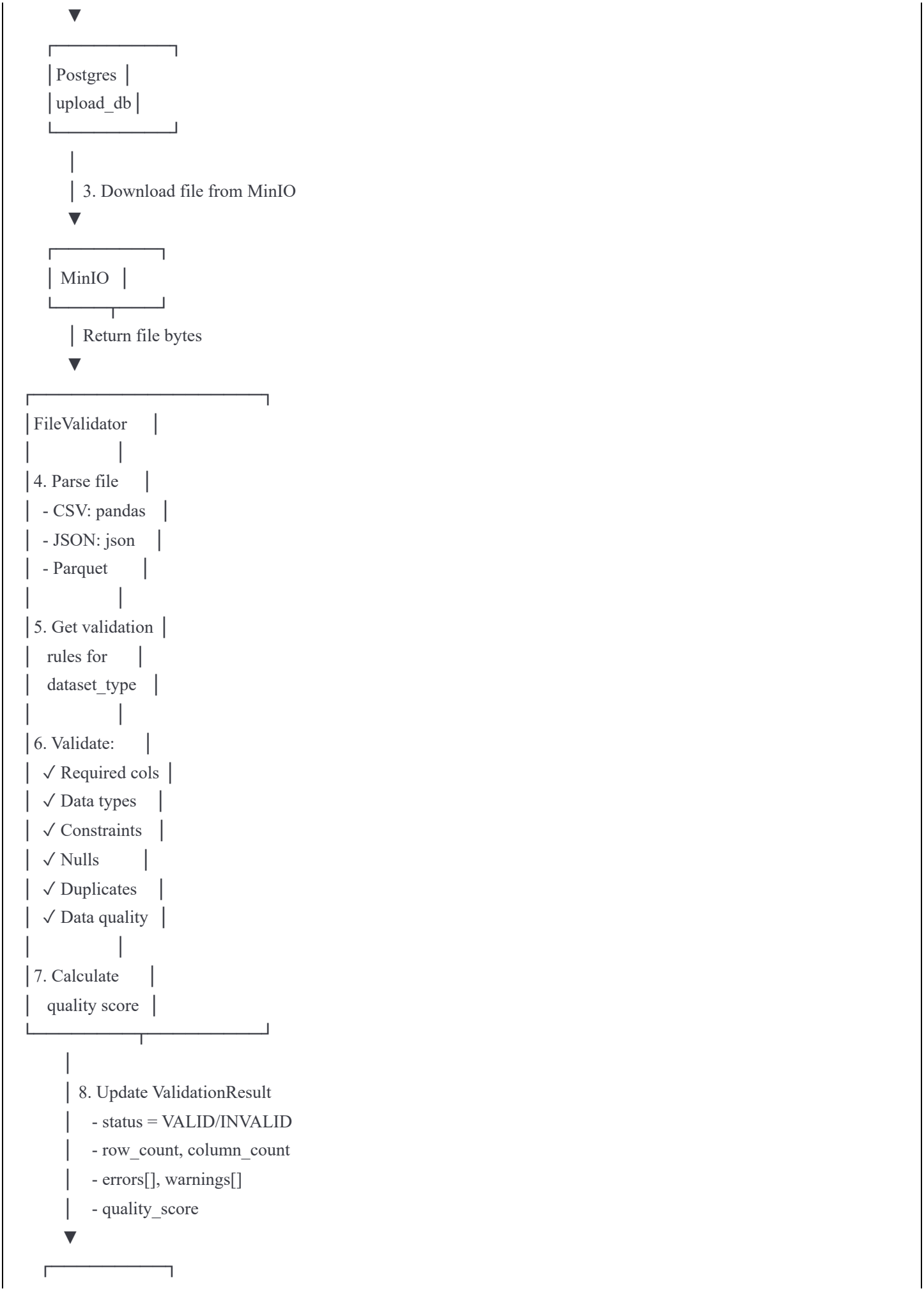



(fileName, fileType, metadata)
Kong Gateway
3. Forward + JWT Check
Upload Mgt Service
4. Generate UUID
5. Build object
key path
6. Call MinIO
getPresigned
URL()
MinIO
Generate
Presign
URL
7. Store
upload
metadata
Postgres
upload_db
8. Return
{uploadId,
presigned
URL}
Frontend

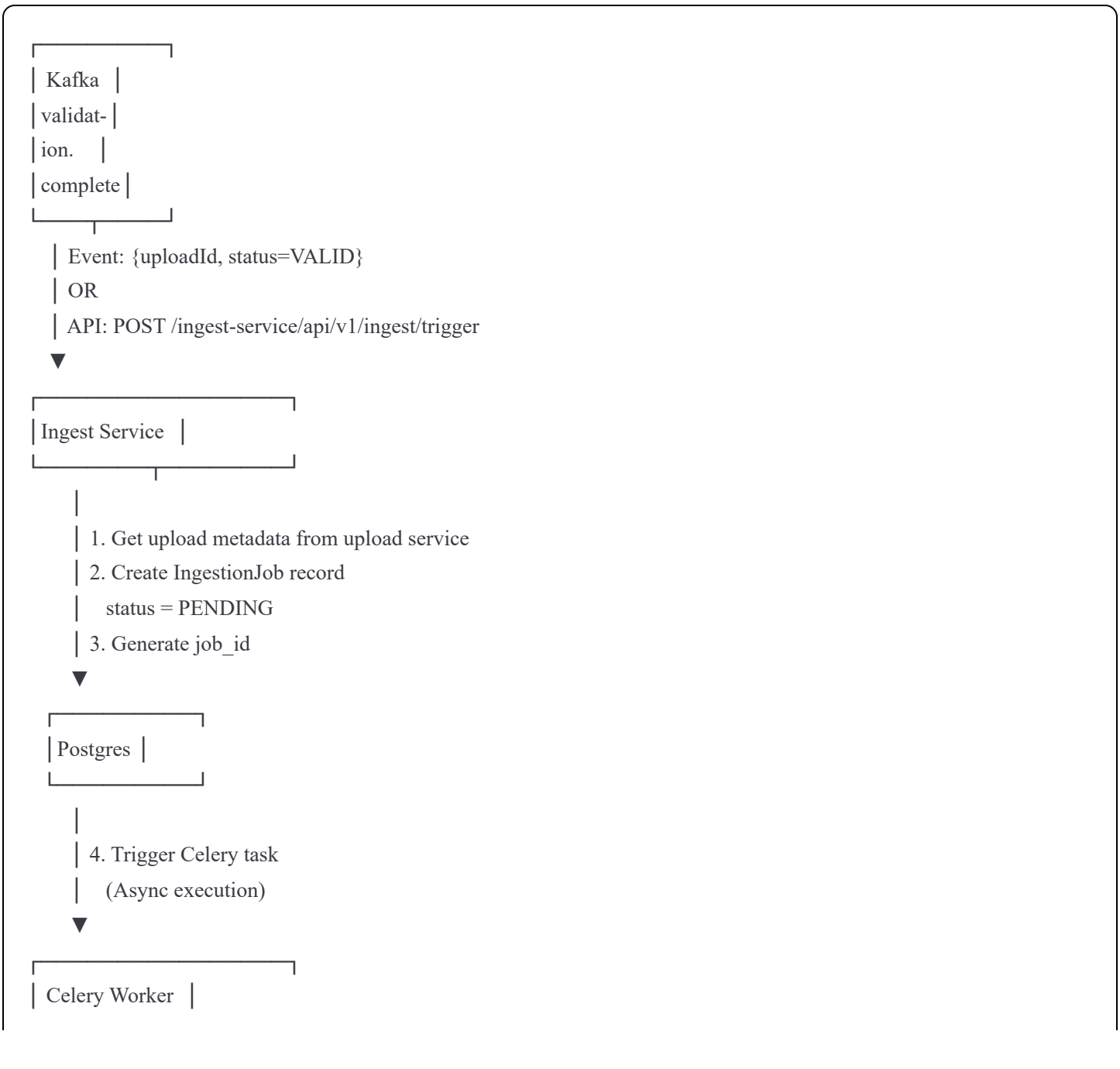


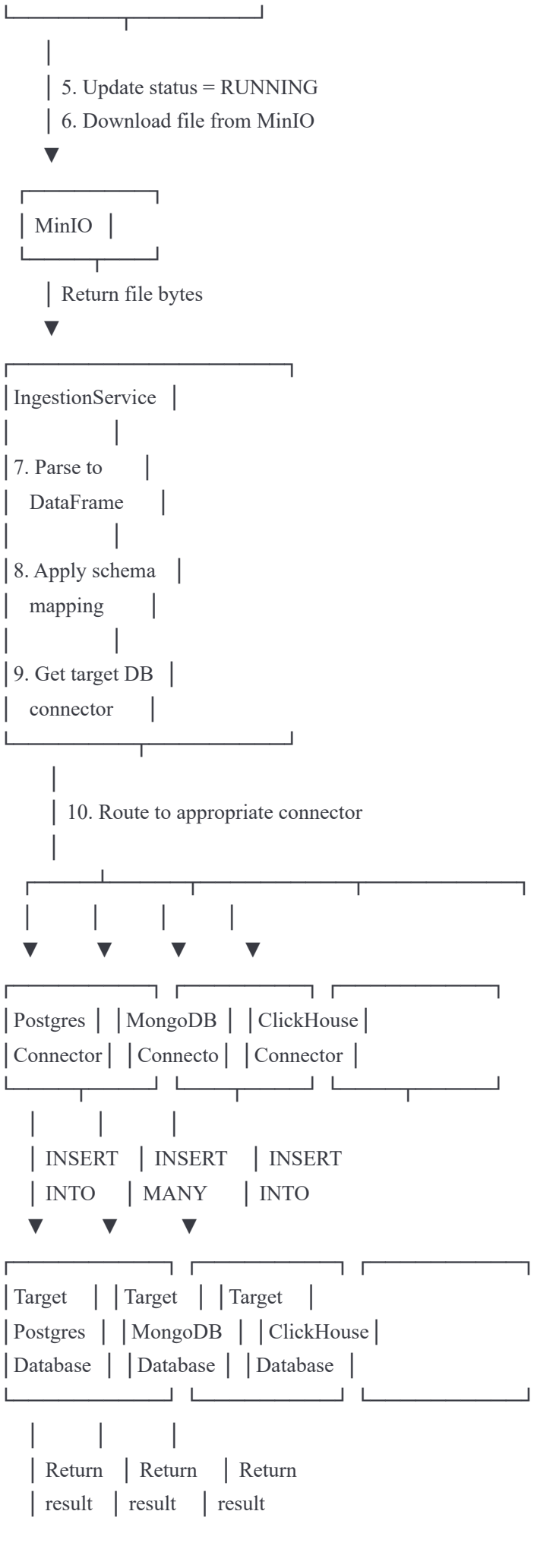
2.2 Validation Flow

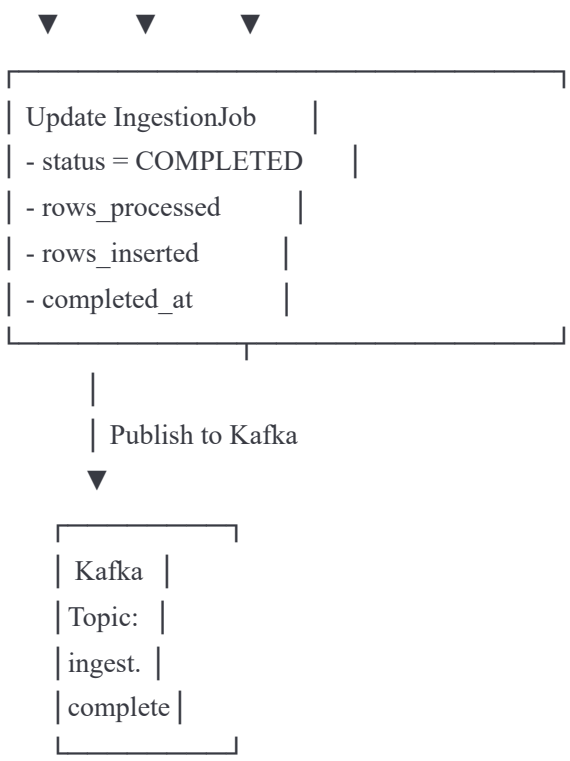



Postgres
upload_db
3. Download file from MinIO
MinIO
Return file bytes
FileValidator
4. Parse file
- CSV: pandas
- JSON: json
- Parquet
5. Get validation rules for dataset_type
6. Validate:
✓ Required cols
✓ Data types
✓ Constraints
✓ Nulls
✓ Duplicates
✓ Data quality
7. Calculate quality score
8. Update ValidationResult
- status = VALID/INVALID
- row_count, column_count
- errors[], warnings[]
- quality_score

2.3 Ingestion Flow




5. Update status = RUNNING
6. Download file from MinIO
MinIO
Return file bytes
IngestionService
7. Parse to DataFrame
8. Apply schema mapping
9. Get target DB connector
10. Route to appropriate connector
Postgres Connector
MongoDB Connecto
ClickHouse Connector
INSERT INTO
INSERT MANY
INSERT INTO
Target Postgres Database
Target MongoDB Database
Target ClickHouse Database
Return result
Return result
Return result



---

3. DYNAMIC DATABASE ROUTING

3.1 Cấu Hình Dataset

yaml

```yaml
dataset_configs:
  - type: CSV_TRANSACTION
    name: Transaction Data
    default_database: postgresql://transaction-db:5432/transactions
    target_table: transactions
    validation_rules:
      required_columns: [transaction_id, amount, date, customer_id]
      column_types:
        transaction_id: string
        amount: numeric
        date: datetime
      constraints:
        amount: {min: 0, max: 1000000}
        transaction_id: {unique: true}

  - type: JSON_USER_PROFILE
    name: User Profile Data
    default_database: mongodb://mongo:27017/users
    target_collection: user_profiles
    validation_rules:
      required_fields: [user_id, email, created_at]

  - type: PARQUET_ANALYTICS
    name: Analytics Events
    default_database: clickhouse://clickhouse:8123/analytics
    target_table: events
    validation_rules:
      required_columns: [event_id, event_type, timestamp]
```

3.2 Luồng Dynamic Routing

```
User uploads file
      |
      ▼
Selects dataset_type: "CSV_TRANSACTION"
      |
      ▼
Frontend looks up config:
  - default_database: "postgresql://transaction-db:5432/transactions"
  - target_table: "transactions"
      |
      ▼
```

Sends to Upload Management Service:

```
{
  fileName: "sales_data.csv",
  datasetType: "CSV_TRANSACTION",
  targetDatabase: "postgresql://transaction-db:5432/transactions"
}
```

▼

Validation Service validates against rules for CSV_TRANSACTION

▼

Ingest Service:

1. Reads targetDatabase from metadata
2. Parses connection string
3. Routes to PostgreSQLConnector
4. Inserts into "transactions" table

3.3 Connector Selection Logic

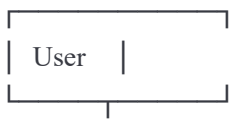
python

```python
def get_connector(database_url: str):
    if database_url.startswith("postgresql://"):
        return PostgreSQLConnector(database_url)
    elif database_url.startswith("mongodb://"):
        return MongoDBConnector(database_url)
    elif database_url.startswith("clickhouse://"):
        return ClickHouseConnector(database_url)
    else:
        raise ValueError(f"Unsupported database: {database_url}")
```

---

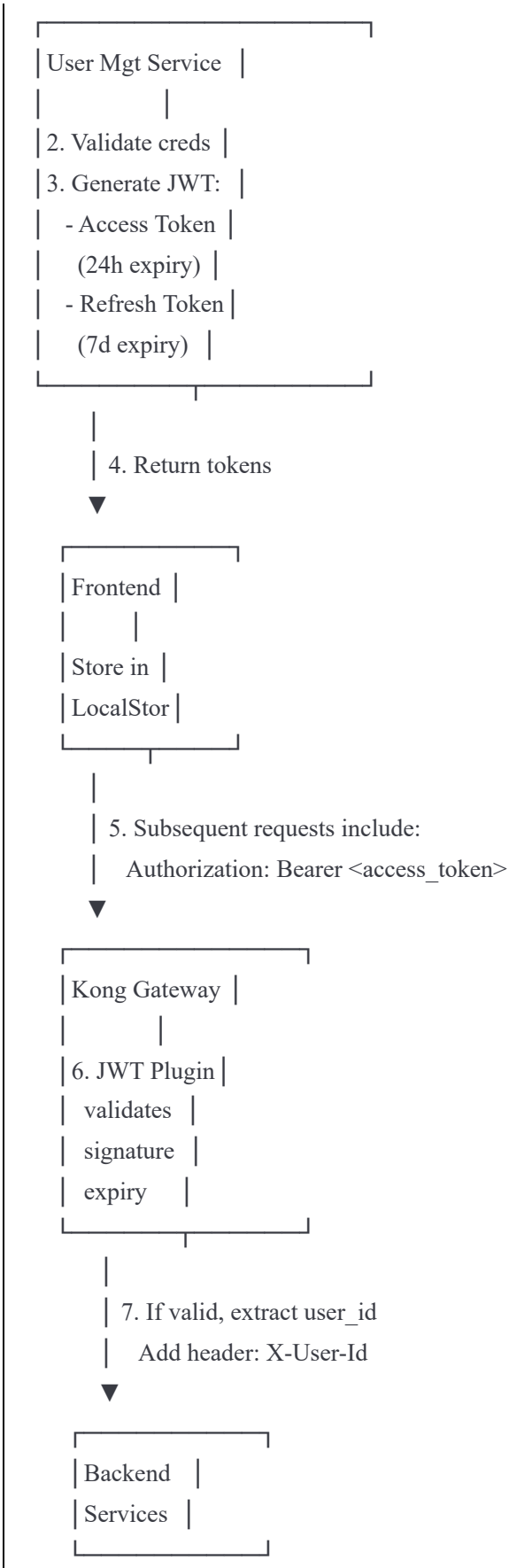
4. SECURITY & AUTHENTICATION

4.1 JWT Flow



1. POST /user-service/auth/login
   {username, password}

▼



4.2 Authorization Levels

USER ROLES:

```
├── ROLE_USER (default)
│   ├── Upload files
│   ├── View own files
│   └── Download own files
│
├── ROLE_DEPARTMENT_MANAGER
│   ├── All ROLE_USER permissions
│   ├── View department files
│   └── Manage department uploads
│
└── ROLE_ADMIN
    ├── All permissions
    ├── Manage users
    ├── View all files
    └── System configuration
```

---

5. ERROR HANDLING & RETRY LOGIC

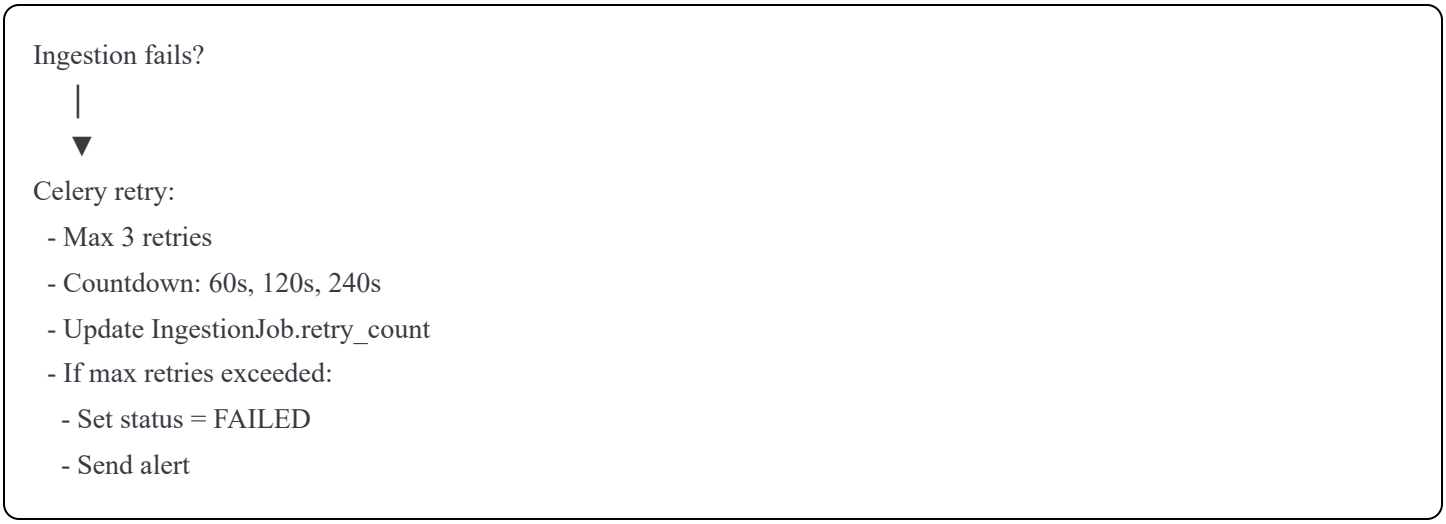
5.1 Upload Retry

```
Upload fails?
    │
    ▼
Frontend retry logic:
  - Max 3 retries
  - Exponential backoff
  - User notification
```

5.2 Validation Retry

```
Validation fails (network error)?
    │
    ▼
Consumer retry:
  - Kafka consumer offset not committed
  - Message reprocessed
  - Max 5 retries
```

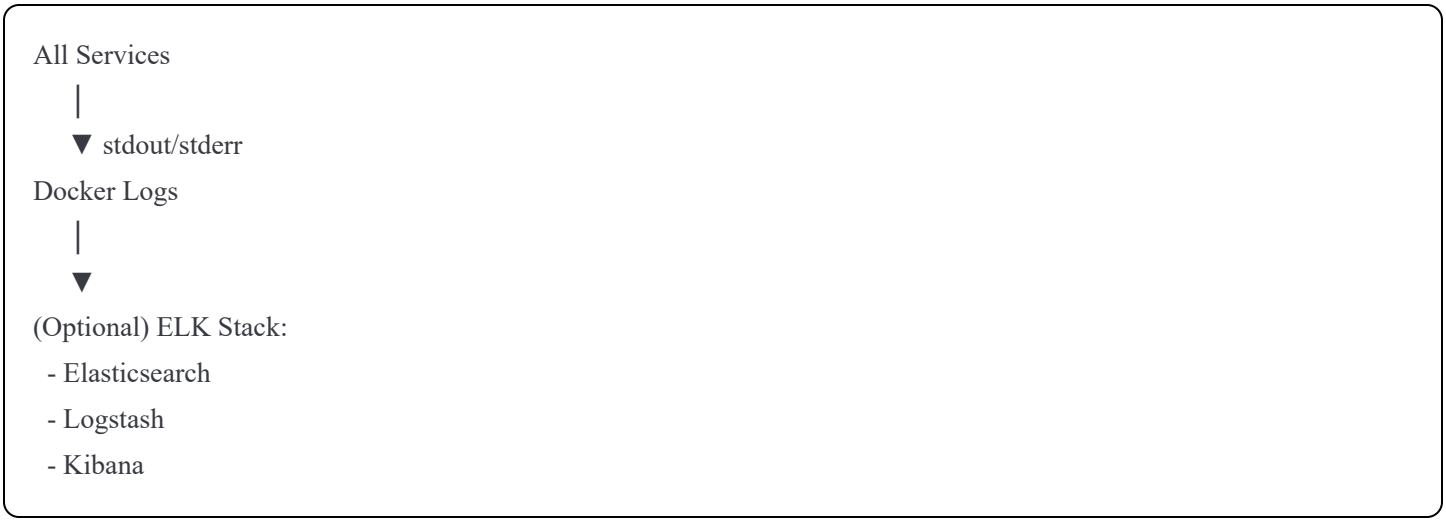
5.3 Ingestion Retry



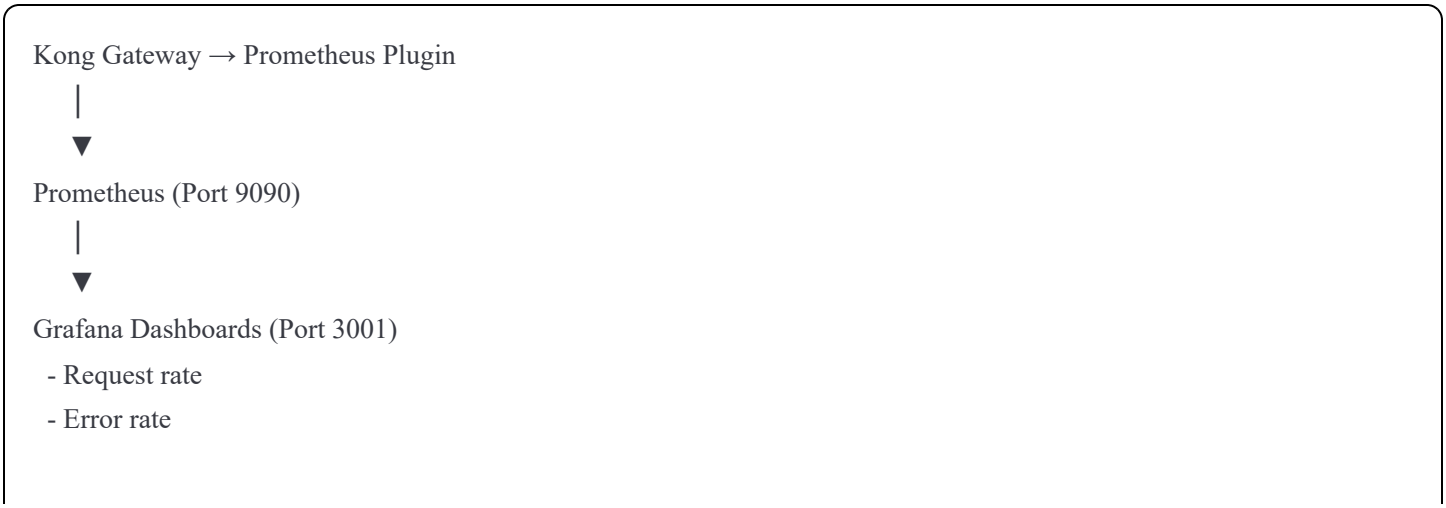
---

6. MONITORING & LOGGING

6.1 Log Aggregation



6.2 Metrics



- Latency
- Service health

---

7. DEPLOYMENT

7.1 Quick Start

```bash
# Clone repository
git clone <repo-url>
cd enterprise-upload-system

# Copy environment file
cp .env.example .env

# Build and start all services
make build
make up

# Check status
make status

# View logs
make logs

# Access points:
# Frontend: http://localhost:3000
# Kong Admin: http://localhost:8001
# MinIO Console: http://localhost:9001
# Grafana: http://localhost:3001
```

7.2 Service Ports

Service	Port(s)	Description
Frontend	3000	React UI
Kong Gateway	8000, 8001	API Gateway & Admin
Upload Management	8080	Java Spring Boot

Service	Port(s)	Description
User Management	8081	Java Spring Boot
Validation Service	8002	Python FastAPI
Ingest Service	8003	Python FastAPI
MinIO	9000, 9001	Object Storage & Console
PostgreSQL	5432	Main Database
Transaction DB	5433	Target Database
MongoDB	27017	Document Database
ClickHouse	8123, 9001	Analytics Database
Kafka	9092	Message Broker
Redis	6379	Cache
Prometheus	9090	Metrics
Grafana	3001	Dashboards

8. SCALABILITY

8.1 Horizontal Scaling

```
# Scale services independently
docker-compose up -d --scale validation-service=3
docker-compose up -d --scale ingest-service=3
docker-compose up -d --scale celery-worker=5
```

8.2 Load Balancing

Kong Gateway automatically load balances across multiple instances.

8.3 Partitioning

- Kafka topics can be partitioned
- MinIO supports distributed deployment

- Databases can be sharded

---

9. BEST PRACTICES

9.1 Security

- ✅ Always use HTTPS in production
- ✅ Rotate JWT secrets regularly
- ✅ Use strong database passwords
- ✅ Enable MinIO TLS
- ✅ Network isolation with Docker networks

9.2 Performance

- ✅ Use connection pooling
- ✅ Implement caching with Redis
- ✅ Batch database operations
- ✅ Use CDN for frontend assets
- ✅ Enable compression

9.3 Reliability

- ✅ Implement circuit breakers
- ✅ Set proper timeouts
- ✅ Use health checks
- ✅ Enable auto-restart policies
- ✅ Regular backups

---

10. TROUBLESHOOTING

10.1 Service Won't Start

bash

```
# Check logs
docker-compose logs <service-name>

# Check health
docker-compose ps

# Restart service
docker-compose restart <service-name>
```

10.2 Database Connection Issues

```
bash

# Check if database is ready
docker-compose exec postgres pg_isready

# Check network connectivity
docker-compose exec upload-management-service ping postgres
```

10.3 Kafka Issues

```
bash

# Check topics
docker-compose exec kafka kafka-topics --list --bootstrap-server localhost:9092

# Check consumer groups
docker-compose exec kafka kafka-consumer-groups --list --bootstrap-server localhost:9092
```

PHỤ LỤC: API ENDPOINTS

Upload Management Service

- `POST /api/v1/uploads/presigned-url` - Get presigned URL
- `POST /api/v1/uploads/{id}/confirm` - Confirm upload
- `GET /api/v1/uploads/{id}/status` - Get upload status
- `GET /api/v1/uploads` - List uploads
- `DELETE /api/v1/uploads/{id}` - Delete upload

User Management Service

- `POST /auth/register` - Register user
- `POST /auth/login` - Login
- `POST /auth/refresh` - Refresh token
- `POST /auth/logout` - Logout
- `GET /api/v1/users/me` - Get current user
- `GET /api/v1/users/{id}` - Get user by ID

Validation Service

- `GET /api/v1/validations/{uploadId}` - Get validation result
- `POST /api/v1/validations/{uploadId}/retry` - Retry validation

Ingest Service

- `POST /api/v1/ingest/trigger` - Trigger ingestion
- `GET /api/v1/ingest/{uploadId}/status` - Get ingestion status
- `GET /api/v1/ingest/{uploadId}/history` - Get ingestion history