PII Detection & Redaction Deploy Plan

Executive Summary

Based on the architectural diagram along with the security incidence use case outlined above, I would suggest deploying the product for PII detection as an API Gateway Plugin combined with sidecar containers for micro-services in a multi-layer design.

Proposed Architecture.

Main Usage: API Gateway Plugin

Location: API Routes Proxy layer (as indicated in the architecture diagram)

Rationale

- It is a single-point-of-control in which all outward API calls travel through.
- Low Latency: Handles in-flight processing without any further network hops
- Centralized Management: A single deployment point for the entire platform.
- Cost Effective: No separate infrastructure required

Secondary Deployment: Sidecar Containers

Location: Side-by-side with microservices in backend (Express + MCP)

Rationale:

- Deep Integration: Extracts PII which is internally generated
- Service-Specific Rules: May personalize detection for PII
- Backup Layer: To cover for in case API gateway layer fails

Technical Implementation

API Gateway Plugin

An external request first enters the PII Plugin (Fast Path), in which there is an opportunity for sensitive information to be identified and highlighted for redaction. Then the request goes for regular processing in Backend Services.

Technology Stack:

- Language: Python (for quick development)

- Framework: Kong/Nginx plugin architecture
- Processing: Real-time streaming parser for JSON for redactions
- Caching: Redis pattern matching cache

Features:

- Sub-millisecond processing for most requests
- Sensitivity levels configurable for each end-point
- Asynchronous logging for detection events for PII
- Circuit breaker for high availability

Sidecar Container Implementation

Main Service Container also speaks directly with the PII Sidecar Container. Both function next to each other, and communication goes in both directions between them.

Features:

- Powered by Envoy: Lightweight HTTP proxy
- Common volume for config updates
- Health check integration
- Auto-scaling based on traffic

Staging Sites and Rationale

Layer 1: Network Layer (Main)

- Where: API Routes Proxy
- Why: Catches external threats, handles 80% of PII exposure
- Latency Impact: <5ms additional processing time
- Coverage includes all outgoing API traffic.

Layer 2: Application Layer (Secondary)

- Where: Microservice sidecars
- Why: Generating internal PII in service-to-service
- Latency Impact: <2ms (local communication)
- Coverage: Data flows between

Layer 3: Data Layer

- Where: Database proxy layer
- Why: Final safety net, audit trail

- Latency Influence: async processing (no real-time influence)
- Coverage: Data persistence layer

Scalability and Performance

Horizontal Scaling

- API Gateway: Load balancer distributes traffic
- Sidecars: Parent service automatic scaling
- Monitoring: Event-driven scaling

Performance Optimizations

- Regex Compilation: Pre-compiled patterns in memory
- Streaming Processing: JSON streaming parser for not allocating large memory
- Async Operations: Asynchronous non-blocking PII logging and alert
- Smart Sampling: Process 100% of suspicious traffic, sample normal traffic

Projected Performance Measures

- Throughput: 10,000+ requests/second per gateway instance
- Latency: 95th percentile < 10
- Memory: <512MB per instance
- CPU: <20% overhead in maximum traffic

Computation Cost

Monthly infra cost

- API Gateway Plugin: ₹16,000 (using existing infrastructure)
- Sidecar Containers: ₹65,000 (additional container supplies
- Monitoring and Logging: ₹25,000 for storing and processing.
- Total: ~ ₹1,02,000/month

Cost Savings

- Prevented Fraud: ₹40 lakh+ per occurrence
- Compliance: Do not attract regulatory fines (~₹40k+)
- ROI: Break-even within first prevented incident

Integration Roadmap

Phase 1: Proof of Concept (2 Weeks)

1. Install plugin on staging API gateway

2. Test with sample traffic
3. Measure performance impact
4. Hone detection rules

Phase 2: Gradual Rollout (4 weeks)

1. Deploy on 10% of production traffic
2. Monitor false positives/negatives
3. Adjust sensitivity settings
4. Scale to 100% of traffic

Phase 3: Full Coverage (2 weeks)

1. Deploy sidecar containers
2. Enable database monitoring
3. Complete audit trail implementation
4. Staff training on monitoring tools

Monitoring and Alerting

Metrics

- Detection rate of PII per endpoint
- Proportion of false
- Processing latency
- System resource utilization

Alert thresholds

- Critical: >1000 PII detections/hour per single
- Warning: >50 false positives
- Info: Performance degradation > 10ms

Risk Mitigation

High Availability

- Multi-zone deployment
- Circuit breaker patterns.
- Graceful degradation (fail-open with logging

Security

- Encryption of PII
- Short-term holding (30 days)
- Access control for detection rules
- Systematic security audits

Alternative Option

Browser Extension.

A **browser extension** was considered since it can catch client-side data leaks. However, it offers limited coverage, is harder to manage, and can introduce performance issues. Because of these scalability challenges, it was not adopted.

Database Triggers

They are effective at capturing all data persistence events, but introduce high latency and negatively affect database performance. For this reason, their use was limited only to auditing and monitoring.

Conclusion

The proposed multi-layer security plan, which favors first-line deployment at the API Gateway, achieves an ideal balance between:

- Speculative Execution: Very low
- Coverage: Full detection of
- Cost: Uses available infrastructure
- Maintenance: Centralized management

This solution directly addresses the breach scenario by intercepting Personally Identifiable Information (PII) leaks at the network ingress layer, thereby preventing them from reaching external logs or unmonitored endpoints.