Kushim System Architecture Deep Dive

Document Purpose

This document translates the Kushim PRD into a concrete, build-ready system architecture. It defines service boundaries, data flows, deployment topology, and technology choices sufficient for senior engineers to begin implementation.

---

1. Architectural Principles

The architecture is guided by the following principles:

1. **Context First**: Relationships are first-class citizens, not derived views.
2. **Event-Driven by Default**: Work context is temporal and evolving.
3. **Explainability**: Every inferred relationship must be traceable.
4. **Progressive Intelligence**: Deterministic systems precede ML dependency.
5. **Least Privilege & Zero Trust**: OAuth scopes and data isolation are mandatory.

---

2. High-Level System Overview

Kushim is composed of six major subsystems:

1. Client Applications (Web + Desktop)
2. API Gateway
3. Ingestion & Sync Layer
4. Context Intelligence Layer
5. Data Stores
6. Action Execution Layer

```
[ Clients ]
     |
[ API Gateway ]
     |
-------------------------------------------------
| Ingestion | Context Engine | Action Services |
-------------------------------------------------
     |
[ Data Stores ]
```

---

3. Client Applications

3.1 Platforms

- Web (React)
- Desktop (Electron / Tauri)

3.2 Responsibilities

- Authentication initiation
- Global Command Bar UI
- Ambient Feed rendering
- Context Group visualization
- User feedback capture (implicit + explicit)

3.3 Design Constraints

- Stateless clients
- All business logic server-side
- Optimistic UI for actions

---

4. API Gateway

4.1 Role

The API Gateway is the single ingress point for all clients.

4.2 Responsibilities

- Auth token validation
- Rate limiting
- Request routing
- Response shaping

4.3 Suggested Stack

- GraphQL (preferred for context queries)
- REST (for ingestion callbacks)

---

5. Ingestion & Sync Layer

5.1 Adapter Services

One adapter per external platform: - Jira Adapter - GitHub Adapter - Slack Adapter - Google Workspace Adapter - Gmail Adapter

Each adapter is an independent service.

5.2 Ingestion Flow

1. User authorizes platform via OAuth
2. Adapter stores encrypted tokens
3. Initial full sync triggered
4. Webhooks registered where available
5. Incremental updates streamed as events

5.3 Event Emission

All changes are emitted as normalized events:

```
ArtifactCreated
ArtifactUpdated
ArtifactDeleted
```

---

6. Normalization Pipeline

6.1 KushimStandardRecord (KSR)

All ingestion outputs are transformed into KSRs.

Pipeline Stages: 1. Raw Payload Validation 2. Field Mapping 3. Metadata Enrichment 4. Identity Resolution (users, emails)

6.2 Failure Handling

- Invalid payloads quarantined
- Partial records allowed with confidence penalties

---

7. Context Intelligence Layer

This is the core differentiation layer.

7.1 Feature Extraction Service

Inputs: KSR events

Outputs: Feature vectors

Features: - Explicit identifiers - Temporal deltas - Actor overlap - Keyword TF-IDF (Phase 1) - Embeddings (Phase 2)

7.2 Relationship Engine

Phase 1: Deterministic Engine

- Rule-based scoring
- Weighted signal aggregation
- Threshold-based link creation

Example:

```
if ticket_id_match: +0.7
if shared_author: +0.2
if <24h delta: +0.1
link if score >= 0.8
```

Phase 2: ML-Augmented Engine

- Sentence/document embeddings
- Approximate nearest neighbor search
- Graph clustering (Louvain / Leiden)

7.3 Context Group Manager

Responsibilities: - Create, update, merge, split groups - Maintain membership weights - Version group state over time

Groups are stored as graph nodes.

---

8. Action Execution Layer

8.1 Command Abstraction

User actions are expressed as:

```
<verb> <object> <context>
```

Examples: - comment PR "LGTM" - assign ticket to Alice

8.2 Execution Flow

1. Parse command
2. Resolve target artifact
3. Validate permissions
4. Invoke platform API
5. Emit action result event

---

9. Data Stores

9.1 Relational Database (PostgreSQL)

Stores: - Users - OAuth credentials - Platform connections - Audit logs

9.2 Graph Database (Neo4j / Neptune)

Stores: - Artifacts - Relationships - Context Groups

9.3 Vector Store (Phase 2)

Stores: - Artifact embeddings

---

10. Eventing & Messaging

10.1 Event Bus

- Kafka / Pulsar

10.2 Event Types

- Ingestion Events
- Linking Events
- Group Mutation Events
- User Interaction Events

Event sourcing enables replay and explainability.

---

11. Security Architecture

- OAuth tokens encrypted at rest
- Per-tenant data isolation
- Fine-grained RBAC
- Audit logging for all actions

12. Deployment Architecture

12.1 Infrastructure

- Kubernetes
- Horizontal pod autoscaling
- Separate clusters for ingestion vs core

12.2 Environment Separation

- Dev
- Staging
- Production

13. Observability

- Distributed tracing (OpenTelemetry)
- Structured logging
- Link confidence dashboards

14. Scalability Considerations

- Adapter services scale independently
- Graph queries optimized for locality
- Context group caching layer (Redis)

15. Failure Modes & Recovery

Failure	Strategy
Adapter outage	Backfill on recovery
Bad linking	User feedback rollback
API rate limits	Adaptive throttling

16. Build Order Recommendation

1. Core ingestion + normalization
2. Deterministic relationship engine

3. Graph storage + context groups
4. Read-only UI
5. Action execution
6. ML augmentation

---

17. Definition of Architectural Done

- End-to-end flow from Slack message → Context Group
- Explainable links visible in UI
- Actions executed without platform navigation

End of System Architecture Deep Dive