NexusAEC - Architecture Documentation

Last Updated: 2026-01-09 **Version:** 1.0 **Architecture Type:** Unified LiveKit Voice Stack

---

Table of Contents

---

1. System Overview

1.1 Purpose

NexusAEC is a **voice-driven AI executive assistant** that enables professionals to manage email communications through natural voice interactions while on the go. The system:

- Aggregates emails from multiple providers (Outlook + Gmail)
- Identifies critical/urgent messages using AI-powered red flag detection
- Generates personalized voice briefings (podcast-style)
- Executes email actions via voice commands
- Maintains safety through desktop-based draft review

1.2 Core Principles

1. **Voice-First**: All interactions designed for hands-free operation
2. **Safety-First**: High-risk actions require explicit approval
3. **Privacy-First**: Minimal data retention, transparent audit trails
4. **Provider-Agnostic**: Unified interface across Outlook and Gmail
5. **Scalable Intelligence**: Three-tier memory for performance and personalization

1.3 System Boundaries

In Scope:

- Email reading, prioritization, and simple actions (mark read, flag, move)
- Voice-based briefing and command execution
- Draft creation (requires review before sending)
- Calendar and contact integration for context

Out of Scope (MVP):

- Sending emails without review
- File attachment handling

- Complex workflow automation
- Multi-user/team features
- Email composition beyond simple replies

---

2. High-Level Architecture

2.1 System Context Diagram

```
┌─────────────────────────────────────────────────────────────────────┐
│                        External Services                             │
├─────────────────────────────────────────────────────────────────────┤
│  Microsoft Graph  │  Google APIs  │  LiveKit Cloud  │  OpenAI  │    │
│    (Outlook)      │   (Gmail)     │   (Voice)       │ (GPT-4o) │    │
└────────┬──────────┴───────┬───────┴────────┬────────┴────┬─────┴────┘
         │                  │                │             │
         │                  │                │             │
┌────────▼──────────────────▼────────────────▼─────────────▼──────────┐
│                      NexusAEC System                                  │
│  ┌─────────────┐  ┌──────────────┐  ┌──────────────┐                │
│  │   Mobile    │  │   Desktop    │  │  Backend API │                │
│  │    App      │  │    App       │  │              │                │
│  │ (React      │  │  (Electron)  │  │(Express/     │                │
│  │  Native)    │  │              │  │ Fastify)     │                │
│  └──────┬──────┘  └──────┬───────┘  └──────┬───────┘                │
│         │                │                  │                        │
│         └────────────────┼──────────────────┘                        │
│                          │                                            │
│  ┌──────────────────────▼───────────────────────────┐                │
│  │          Shared Packages (Monorepo)              │                │
│  │                                                 │                │
│  │  ┌─────────────┐  ┌──────────────┐              │                │
│  │  │Email        │  │Intelligence  │              │                │
│  │  │Providers    │  │Layer         │              │                │
│  │  └─────────────┘  └──────────────┘              │                │
│  │                                                 │                │
│  │  ┌─────────────┐  ┌──────────────┐              │                │
│  │  │Encryption   │  │Secure Storage│              │                │
│  │  └─────────────┘  └──────────────┘              │                │
│  │                                                 │                │
│  └──────────────────────────────────────────────────┘                │
│                                                                       │
│  ┌───────────────────────────────────────────────────┐               │
│  │           Infrastructure Services                 │               │
│  │  ┌───────────┐  ┌──────────┐  ┌──────────┐       │                │
│  │  │   Redis   │  │ Supabase │  │LiveKit   │       │                │
│  │  │ (Session  │  │(Vector   │  │ Agent    │       │                │
│  │  │   State)  │  │  Store)  │  │          │       │                │
│  │  └───────────┘  └──────────┘  └──────────┘       │                │
│  └───────────────────────────────────────────────────┘               │
└───────────────────────────────────────────────────────────────────────┘
```

2.2 Container Architecture

```
┌─────────────────────────────────────────────────────────────────────┐
│                          Client Layer                                │
├──────────────────────┬──────────────────────────────────────────────┤
│   Mobile App         │   Desktop App                                 │
│   (React Native)     │   (Electron)                                  │
│   - Voice Briefing   │   - Draft Review                              │
│   - PTT Commands     │   - Audit Trail                               │
│   - Settings         │   - Batch Operations                          │
└──────────┬───────────┴────────────┬─────────────────────────────────┘
           │                        │
           │        HTTPS/WSS       │
           │                        │
┌──────────▼────────────────────────▼─────────────────────────────────┐
│                      Application Layer                               │
├──────────────────────┬──────────────────────────────────────────────┤
│  Backend API         │  LiveKit Agent                                │
│  (Express/Fastify)   │  (Node.js/Python)                             │
│  - OAuth Callbacks   │  - Room Management                            │
│  - Token Generation  │  - GPT-4o Reasoning                           │
│  - Sync Endpoints    │  - STT/TTS Orchestration                      │
└──────────┬───────────┴────────────┬─────────────────────────────────┘
           │                        │
           │    Internal Network    │
           │                        │
┌──────────▼────────────────────────▼─────────────────────────────────┐
│                      Service Layer                                   │
├──────────────────────┬───────────────────┬──────────────────────────┤
│  Email Providers     │  Intelligence     │  Storage & State          │
│  - Outlook Adapter   │  - Red Flags      │  - Redis (Session)        │
│  - Gmail Adapter     │  - Clustering     │  - Supabase (Vector)      │
│  - Unified Inbox     │  - Summarization  │  - Secure Storage         │
│  - Smart Drafts      │  - RAG Retrieval  │  - Encryption             │
└──────────┬───────────┴───────────┬───────┴──────────┬───────────────┘
           │                       │                  │
           │   External APIs       │                  │
           │                       │                  │
┌──────────▼───────────┐  ┌────────▼────────┐  ┌─────▼──────────────┐
│  Microsoft Graph     │  │  OpenAI API     │  │  Cloud Storage     │
│  Google APIs         │  │  - GPT-4o       │  │  - Redis Cloud     │
│                      │  │  - Embeddings   │  │  - Supabase Cloud  │
└──────────────────────┘  └─────────────────┘  └────────────────────┘
```

3. Three-Tier Memory Model

The system uses a **three-tier memory architecture** to balance performance, personalization, and cost:

3.1 Architecture Overview

```
┌─────────────────────────────────────────────────────────────────────┐
│                         Tier 1: Ephemeral                            │
│                     (In-Memory Processing)                      │
├─────────────────────────────────────────────────────────────────┤
│  Purpose: Real-time email analysis and scoring                  │
│  Lifecycle: Request-scoped (discarded after processing)         │
│  Location: Application memory (Node.js/Python process)          │
│                                                                 │
│  Components:                                                    │
│  ┌──────────────┐  ┌──────────────┐  ┌───────────────┐          │
│  │Red Flag      │  │Topic         │  │VIP            │          │
│  │Scorer        │  │Clusterer     │  │Detector       │          │
│  └──────────────┘  └──────────────┘  └───────────────┘          │
│  ┌──────────────┐  ┌──────────────┐                             │
│  │Thread        │  │Calendar      │                             │
│  │Velocity      │  │Proximity     │                             │
│  └──────────────┘  └──────────────┘                             │
│                                                                 │
│  Data: StandardEmail[], RedFlag[], Topic[]                      │
└─────────────────────────────────────────────────────────────────┘
                              │
                              │ Session State
                              ▼
┌─────────────────────────────────────────────────────────────────┐
│                       Tier 2: Session State                     │
│                          (Redis Cache)                          │
├─────────────────────────────────────────────────────────────────┤
│  Purpose: Live "Drive State" for active voice sessions          │
│  Lifecycle: Session-scoped (24-hour TTL)                        │
│  Location: Redis (in-memory key-value store)                    │
│                                                                 │
│  Components:                                                    │
│  ┌───────────────────────────────────────────────┐              │
│  │ DriveState                                    │              │
│  │ {                                             │              │
│  │   sessionId: "uuid",                          │              │
│  │   currentTopicIndex: 2,                       │              │
│  │   currentItemIndex: 5,                        │              │
│  │   itemsRemaining: 12,                         │              │
│  │   interruptStatus: "none",                    │              │
│  │   lastPosition: 1234,    // milliseconds      │              │
│  │   startedAt: "2026-01-09T10:00:00Z",          │              │
│  │   updatedAt: "2026-01-09T10:15:23Z"           │              │
│  │ }                                             │              │
│  └───────────────────────────────────────────────┘              │
│                                                                 │
│  ┌───────────────────────────────────────────────┐              │
│  │ Shadow Processor (Background Service)         │              │
│  │ - Listens to LiveKit transcript events        │              │
│  │ - Updates Redis state in real-time            │              │
│  │ - "Ack & Act" pattern for responsiveness      │              │
│  └───────────────────────────────────────────────┘              │
│                                                                 │
│  TTL: 24 hours (auto-expire stale sessions)                     │
```

```
└──────────────────────────────────────────────────────────────────────┘
                                │
                                │ Domain Knowledge
                                ▼
┌──────────────────────────────────────────────────────────────────────┐
│                     Tier 3: Knowledge Base                            │
│                    (Supabase Vector Store)                            │
├──────────────────────────────────────────────────────────────────────┤
│  Purpose: Long-term domain knowledge (assets, manuals, procedures)  │
│  Lifecycle: Persistent (until explicitly deleted)                    │
│  Location: Supabase (PostgreSQL + pgvector)                          │
│                                                                        │
│  Schema:                                                              │
│  ┌───────────────────────────────────────────────────┐               │
│  │ documents                                          │               │
│  │ ├─ id: UUID                                        │               │
│  │ ├─ content: TEXT                                   │               │
│  │ ├─ embedding: VECTOR(1536)  -- OpenAI ada-002    │             │
│  │ ├─ source_type: ENUM('ASSET','SAFETY_MANUAL',    │             │
│  │ │                       'PROCEDURE')              │              │
│  │ ├─ metadata: JSONB {                              │              │
│  │ │    asset_id?: "P-104",                          │              │
│  │ │    category?: "Pump",                           │              │
│  │ │    location?: "Riverside Bridge",               │              │
│  │ │    source_file?: "manual_v2.pdf",               │              │
│  │ │    chunk_index?: 3                              │              │
│  │ │  }                                               │               │
│  │ └─ created_at: TIMESTAMP                          │              │
│  └───────────────────────────────────────────────────┘              │
│                                                                        │
│  Indexes:                                                             │
│  - Vector similarity (HNSW for fast retrieval)                       │
│  - source_type (for filtering)                                       │
│  - metadata->>'asset_id' (for asset lookup)                          │
│                                                                        │
│  Data Sources:                                                        │
│  - MVP: Hardcoded seed files (20-50 assets)                          │
│  - Production: CSV import from client asset management system        │
│  - Safety Manuals: PDF extraction and chunking                       │
└──────────────────────────────────────────────────────────────────────┘
```

3.2 Data Flow Between Tiers

```
User Voice Command → LiveKit Agent
                           ↓
                    GPT-4o Reasoning
                           ↓
                     ┌─────┴─────┐
                     │           │
          ┌──────────▼───────┐ ┌─▼──────────┐
          │ Tier 2: Redis    │ │ Tier 3:    │
```

```
         │ (Check state)  │  │ Vector Store│
         │                │  │ (RAG lookup)│
         └────────┬───────┘  └──┬──────────┘
                  │             │
                  └──────┬──────┘
                         ▼
              ┌──────────────────────┐
              │ Tier 1: Ephemeral    │
              │ (Process emails,     │
              │  score red flags)    │
              └──────────┬───────────┘
                         ▼
                 Generate Response
                         ↓
                  ElevenLabs TTS → User
```

3.3 Memory Tier Selection Guide

Use Case	Tier	Rationale
Email content analysis	Tier 1	Ephemeral, no need to persist email bodies
Red flag scoring	Tier 1	Computed per-request, discarded after
Current briefing position	Tier 2	Session state, needs to survive interruptions
User interrupt handling	Tier 2	Real-time updates from transcript
Asset knowledge (NCE IDs)	Tier 3	Persistent domain knowledge
Safety manual excerpts	Tier 3	Persistent, rarely changes
User preferences (VIPs)	Tier 3	Persistent, sync across devices

---

4. Voice Processing Stack

4.1 LiveKit Unified Architecture

Key Decision: Use LiveKit Cloud as the central hub for ALL voice processing, eliminating custom WebRTC implementation.

```
┌─────────────────────────────────────────────────────────────────────┐
│                        LiveKit Cloud                                 │
│                  (Real-time Media Server)                            │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│                                                                     │
│  ┌────────────────────────────────────────────────────────────┐     │
│  │                    LiveKit Room                            │     │
│  │  ┌──────────────┐           ┌─────────────────┐           │     │
│  │  │ Participant  │◄─────────►│  Participant    │           │     │
│  │  │   (Mobile    │  WebRTC   │  (Agent)        │           │     │
│  │  │     App)     │   Audio   │                 │           │     │
│  │  └──────┬───────┘           └────────┬────────┘           │     │
│  │         │                             │                    │     │
│  │         │ Audio Track                 │ Audio Track        │     │
│  │         ▼                             ▼                    │     │
│  │  ┌───────────────────────────────────────────────┐        │     │
│  │  │         LiveKit Media Pipeline                │        │     │
│  │  │  - Audio routing                              │        │     │
│  │  │  - Track management                           │        │     │
│  │  │  - Connection quality monitoring              │        │     │
│  │  │  - Network resilience (packet loss, jitter)   │        │     │
│  │  └───────────────────────────────────────────────┘        │     │
│  └──────────────────────────────────────────────────────────────┘     │
│                                                                        │
│  ┌──────────────────────────────────────────────────────────────┐     │
│  │              STT Plugin (Deepgram Nova-2)                    │     │
│  │  - Real-time transcription                                   │     │
│  │  - Custom vocabulary (NCE Asset IDs, project names)          │     │
│  │  - Interim results for responsiveness                        │     │
│  │  - Multi-language support (en-US, en-GB, en-IN, en-AU)     │     │
│  └──────────────────────────┬───────────────────────────────────┘     │
│                             │                                          │
│                             │ Transcript Events                        │
│                             ▼                                          │
│  ┌──────────────────────────────────────────────────────────────┐     │
│  │              TTS Plugin (ElevenLabs Turbo v2.5)              │     │
│  │  - Streaming text-to-speech                                  │     │
│  │  - Low-latency (<300ms)                                      │     │
│  │  - Professional voice for in-motion listening                │     │
│  └──────────────────────────────────────────────────────────────┘     │
│                                                                        │
└────────────────────────────┬─────────────────────────────────────┘
                             │
                             │ Transcript + Room Events
                             ▼
┌────────────────────────────────────────────────────────────────────────┐
│                     LiveKit Backend Agent                              │
│                    (Node.js/Python Service)                            │
├────────────────────────────────────────────────────────────────────────┤
│                                                                        │
│  ┌──────────────────────────────────────────────────────────────┐     │
│  │              GPT-4o Reasoning Loop                           │     │
│  │                                                               │     │
│  │  Transcript → Intent Parsing → Function Calling →            │     │
│  │               Response Generation                            │     │
│  │                                                               │     │
│  │  Function Tools:                                             │     │
│  │  - Email Actions (mute, flag, draft, search)               │     │
│  │  - Navigation (skip, repeat, go deeper, pause)             │     │
│  │  - Disambiguation (clarify ambiguous commands)             │     │
│  └──────────────────────────┬───────────────────────────────────┘     │
│                             │                                          │
│                             │ Execute Actions                          │
│                             ▼                                          │
│  ┌────────────────────────────────────────────────────────────┐     │
│  │            Shadow Processor (Background)                   │     │
│  │  - Listens to transcript stream                            │     │
│  │  - Updates Redis DriveState in real-time                   │     │
│  │  - "Ack & Act" pattern (acknowledge → update state)        │     │
│  └────────────────────────────────────────────────────────────┘     │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

4.2 Audio Processing Flow

```
User Speaks
    │
    ├─► Mobile App captures audio
    │       │
    │       ├─► @livekit/react-native SDK
    │       │       │
    │       │       ├─► WebRTC transport to LiveKit Cloud
    │       │               │
    │       │               ├─► Deepgram STT Plugin
    │       │               │       │
    │       │               │       ├─► Transcript: "Flag all emails from John as
priority"
    │       │               │               │
    │       │               │               └─► LiveKit Agent receives transcript
    │       │               │                       │
    │       │               │                       ├─► GPT-4o parses intent
    │       │               │                       │       │
    │       │               │                       │       ├─► Function:
flag_priority_vip(name: "John")
    │       │               │                       │               │
    │       │               │                       │               ├─► Email Provider
API call
    │       │               │                       │               │
    │       │               │                       │               └─► Response:
"Flagged 3 emails from John Smith"
    │       │               │                       │                       │
    │       │               │                       │                       ├─► GPT-4o
generates natural response
    │       │               │                       │                               │
    │       │               │                       │
└─► "Done. I've flagged 3 emails from John."
    │       │               │                       │
│
    │       │               │
└─────────────────────────────────────────┘
    │       │               │
│
    │       │               ├─► ElevenLabs TTS Plugin
│
    │       │               │       │
```

```
│
    │       │               │       ├─► Audio stream
◄───────────────────────────────────┘
    │       │               │               │
    │       │               │               └─► WebRTC transport to Mobile App
    │       │               │                       │
    │       └───────────────┴───────────────────────┘
    │
    └─► User hears: "Done. I've flagged 3 emails from John."
```

4.3 Barge-in Handling (LiveKit Native)

```
Agent is speaking (TTS playing)
    │
    ├─► User starts speaking (interruption detected by VAD)
    │       │
    │       ├─► LiveKit detects audio from mobile participant
    │       │       │
    │       │       ├─► Agent pauses TTS playback
    │       │               │
    │       │               ├─► Deepgram transcribes user input
    │       │                       │
    │       │                       ├─► GPT-4o decides:
    │       │                       │       │
    │       │                       │       ├─► New command? → Execute new action
    │       │                       │       │
    │       │                       │       └─► Clarification? → Resume after answer
    │       │                               │
    │       └───────────────────────────────┘
    │
    │
    └─► Agent responds to interruption
```

4.4 Dead Zone Recovery (Network Resilience)

```
Mobile App monitors ConnectionQuality events from LiveKit SDK


┌──────────────────────────────────────────────────────────────────────┐
│  Good Quality                                                         │
│  └─► Green indicator, normal operation                               │
└──────────────────────────────────────────────────────────────────────┘
                            │
                            │ Network degrades
                            ▼
┌──────────────────────────────────────────────────────────────────────┐
│  Poor Quality                                                         │
│  └─► Yellow indicator, show "Connection degraded" warning            │
└──────────────────────────────────────────────────────────────────────┘
                            │
                            │ Network lost
                            ▼
┌──────────────────────────────────────────────────────────────────────┐
```

```
│  Lost Connection                                                    │
│  └─► Red indicator, pause UI, show "Connection lost" overlay       │
│  └─► Save last position to Redis DriveState                        │
└─────────────────────────────────────────────────────────────────────┘
                              │
                              │ Network restored
                              ▼
┌─────────────────────────────────────────────────────────────────────┐
│  Auto-Reconnect                                                     │
│  └─► Reconnect to LiveKit room                                     │
│  └─► Fetch DriveState from Redis                                   │
│  └─► Resume briefing from last position                            │
│  └─► Green indicator, "Connection restored"                        │
└─────────────────────────────────────────────────────────────────────┘
```

5. Email Integration Layer

5.1 Unified Adapter Pattern

Key Decision: Abstract provider differences behind a common `EmailProvider` interface.

```
┌─────────────────────────────────────────────────────────────────────┐
│                  Application Layer                                  │
│  (UnifiedInboxService, SmartDraftService, etc.)                     │
└─────────────────────────────┬───────────────────────────────────────┘
                              │
                              │ Uses EmailProvider interface
                              ▼
┌─────────────────────────────────────────────────────────────────────┐
│                 EmailProvider Interface                             │
├─────────────────────────────────────────────────────────────────────┤
│  Methods:                                                           │
│  - fetchThreads(filters): Promise<StandardThread[]>                │
│  - fetchUnread(): Promise<StandardEmail[]>                         │
│  - createDraft(input): Promise<StandardDraft>                      │
│  - sendDraft(draftId): Promise<void>                               │
│  - markRead(emailIds): Promise<void>                               │
│  - markUnread(emailIds): Promise<void>                             │
│  - moveToFolder(emailIds, folderId): Promise<void>                 │
│  - applyLabel(emailIds, label): Promise<void>                      │
│  - getContacts(): Promise<Contact[]>                               │
│  - getCalendarEvents(filters): Promise<CalendarEvent[]>            │
└─────────────┬───────────────────────────┬───────────────────────────┘
              │                           │
              │                           │
   ┌──────────▼──────────┐     ┌──────────▼──────────┐
   │ OutlookAdapter      │     │  GmailAdapter       │
   │                     │     │                     │
   │ - Microsoft Graph   │     │  - Gmail API        │
   │ - OAuth 2.0 + PKCE  │     │  - OAuth 2.0        │
   │ - Token management  │     │  - Token refresh    │
   └─────────┬─────────┘      └─────────┬─────────┘
             │                          │
             │                          │
   ┌─────────▼─────────┐      ┌─────────▼─────────┐
   │ Microsoft Graph   │      │  Google APIs      │
   │ API               │      │  (Gmail, Cal,     │
   │                   │      │   Contacts)       │
   └───────────────────┘      └───────────────────┘
```

5.2 Data Normalization

All provider-specific data is normalized to standard types:

```
// Provider-Specific (Microsoft Graph)
{
  "@odata.type": "#microsoft.graph.message",
  "id": "AAMkAGI2T...",
  "subject": "Project Update",
  "from": {
    "emailAddress": {
      "name": "John Doe",
      "address": "john@example.com"
    }
  },
  "receivedDateTime": "2026-01-09T10:00:00Z",
  "isRead": false,
  ...
}

// ↓ Normalized to StandardEmail

{
  id: "OUTLOOK:AAMkAGI2T...",
  source: "OUTLOOK",
  providerMessageId: "AAMkAGI2T...",
  threadId: "OUTLOOK:AAMkAGI2T...",
  subject: "Project Update",
  from: {
    email: "john@example.com",
    name: "John Doe"
  },
  to: [...],
  receivedAt: "2026-01-09T10:00:00Z",
  isRead: false,
  ...
}
```

Benefits:

- Single interface for all email operations
- Easy to add new providers (Yahoo, ProtonMail, etc.)
- Simplified testing (mock one interface, not multiple APIs)

- Consistent data shape throughout the system

5.3 Unified Inbox Service

```
┌─────────────────────────────────────────────────────────────────────┐
│              UnifiedInboxService                                     │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│  1. Poll all active adapters in parallel                             │
│                                                                      │
│     ┌───────────────┐         ┌───────────────┐                      │
│     │ OutlookAdapter│         │ GmailAdapter  │                      │
│     │ .fetchUnread()│         │ .fetchUnread()│                      │
│     └───────┬───────┘         └───────┬───────┘                      │
│             │                         │                               │
│             └────────┬────────────────┘                               │
│                      │                                                │
│  2. Normalize to StandardEmail[]                                     │
│     (source discriminator added)                                     │
│                      │                                                │
│                      ▼                                                │
│     [                                                                 │
│       { id: "OUTLOOK:123", source: "OUTLOOK", ... },                 │
│       { id: "GMAIL:456", source: "GMAIL", ... },                     │
│       { id: "OUTLOOK:789", source: "OUTLOOK", ... }                  │
│     ]                                                                 │
│                      │                                                │
│  3. Merge timelines by receivedAt (sort descending)                  │
│                      │                                                │
│                      ▼                                                │
│     [                                                                 │
│       { id: "OUTLOOK:789", receivedAt: "2026-01-09T10:30:00Z" },   │
│       { id: "GMAIL:456", receivedAt: "2026-01-09T10:15:00Z" },     │
│       { id: "OUTLOOK:123", receivedAt: "2026-01-09T09:45:00Z" }    │
│     ]                                                                 │
│                      │                                                │
│  4. Return unified timeline                                          │
│                      ▼                                                │
│     StandardEmail[] (sorted, tagged with source)                     │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

5.4 Smart Draft Routing

```
User: "Reply to that email saying I'll join the meeting"


┌─────────────────────────────────────────────────────────────────────┐
│              SmartDraftService                                       │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│  1. Identify original email's source                                 │
│     ┌─────────────────────────────────────────┐                      │
│     │ Email: { id: "GMAIL:456",              │                       │
│     │          source: "GMAIL",               │                       │
│     │          threadId: "GMAIL:thread123" }  │                       │
│     └─────────────────────────────────────────┘                       │
│                      │                                                │
│  2. Route to appropriate adapter                                      │
│     source === "GMAIL" → Use GmailAdapter                             │
│                      │                                                │
│                      ▼                                                │
│     GmailAdapter.createDraft({                                        │
│       threadId: "GMAIL:thread123",                                    │
│       inReplyToMessageId: "GMAIL:456",                                │
│       body: "Thanks! I'll join the meeting.",                         │
│       isPendingReview: true  // Requires desktop approval             │
│     })                                                                │
│                      │                                                │
│  3. Return draft with source tag                                      │
│                      ▼                                                │
│     StandardDraft {                                                   │
│       id: "GMAIL:draft789",                                           │
│       source: "GMAIL",  // Desktop app knows where to send            │
│       ...                                                             │
│     }                                                                 │
│                                                                       │
│  Default for new emails (not replies): OUTLOOK                        │
│  Dev Mode fallback: GMAIL                                             │
│                                                                       │
└───────────────────────────────────────────────────────────────────────┘
```

6. Intelligence Layer

6.1 Red Flag Detection Pipeline

```
StandardEmail[] (from UnifiedInbox)
     │
     ├─► Parallel Processing
     │       │
     │       ├─► Keyword Matcher
     │       │   - Regex + fuzzy matching
     │       │   - Default patterns: "urgent", "ASAP", "incident", "outage"
     │       │   - User-defined keywords
     │       │   - Score: 0.0 - 1.0
     │       │
     │       ├─► VIP Detector
     │       │   - Check sender against VIP list
     │       │   - Infer importance from interaction frequency
     │       │   - Score: 0.0 (not VIP) or 0.8 (VIP)
     │       │
     │       ├─► Thread Velocity
     │       │   - Count replies in last 24 hours
     │       │   - Detect escalation language ("following up", "3rd reminder")
```

```
    │       │   - Score: 0.0 - 1.0
    │       │
    │       ├─► Calendar Proximity
    │       │   - Check if email relates to upcoming event (keyword match)
    │       │   - Events within next 24h: high score
    │       │   - Score: 0.0 - 1.0
    │       │
    │       └─► Composite Scorer
    │           - Weighted combination:
    │             score = (keyword * 0.3) +
    │                     (vip * 0.4) +
    │                     (velocity * 0.2) +
    │                     (calendar * 0.1)
    │           - Threshold: > 0.7 = RED FLAG
    │
    └─► RedFlag[] (emails exceeding threshold)
            │
            ├─► Explanation Generator (GPT-4o)
            │   - "This is a red flag because John Smith (your VIP) sent 3 follow-ups
            │      about the incident, and you have a meeting with him in 2 hours."
            │
            └─► Return to briefing generator
```

6.2 Topic Clustering

```
StandardEmail[] → Topic Clusterer


┌─────────────────────────────────────────────────────────────────────┐
│                    Topic Clustering Algorithm                       │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│  1. Extract features per email:                                     │
│     - Thread ID (natural grouping)                                  │
│     - Subject (normalized: remove "RE:", "FW:")                     │
│     - Participants (common senders/recipients)                      │
│     - Semantic embedding (optional: OpenAI embeddings)              │
│                                                                     │
│  2. Group by thread ID (same conversation = same topic)             │
│                                                                     │
│  3. Cluster remaining emails by subject similarity                  │
│     - Levenshtein distance for fuzzy matching                       │
│     - Threshold: 80% similarity → same topic                        │
│                                                                     │
│  4. Merge clusters with overlapping participants                    │
│     - If 50%+ participants overlap → merge                          │
│                                                                     │
│  5. Label clusters:                                                 │
│     - User-defined topics (match keywords)                          │
│     - Auto-generated from most common subject                       │
│     - Example: "Q1 Budget Review" (12 emails)                       │
│                                                                     │
│  Output: Topic[] with grouped emails                                │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘


Example Output:
[
  {
    id: "topic-1",
    name: "Q1 Budget Review",
    emails: [ /* 12 emails */ ],
    redFlagCount: 2,
    lastActivityAt: "2026-01-09T10:30:00Z"
  },
  {
    id: "topic-2",
    name: "P-104 Pump Maintenance",
    emails: [ /* 5 emails */ ],
    redFlagCount: 1,
    lastActivityAt: "2026-01-09T09:15:00Z"
  },
  // ... more topics
]
```

6.3 Briefing Generation

```
Topics[] + RedFlags[] → Narrative Generator (GPT-4o)


┌─────────────────────────────────────────────────────────────────────┐
│                    Briefing Script Structure                         │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│  1. Opening                                                          │
│     "Good morning! You have 27 new emails. I've found 3 red flags   │
│      that need your attention. Let's start with those."             │
│                                                                       │
│  2. Red Flags Section (High Priority)                               │
│     "First red flag: John Smith sent 3 follow-ups about the pump    │
│      incident at P-104. You have a meeting with him in 2 hours.     │
│      The latest email says the issue is escalating. Would you like  │
│      me to flag this for follow-up?"                                │
│                                                                       │
│     [Wait for user response or auto-continue after 3 seconds]       │
│                                                                       │
│  3. Topics Section (Grouped by Importance)                           │
│     "Next, you have 12 emails about Q1 Budget Review. The latest    │
│      from Sarah mentions the deadline was moved to Friday. Should   │
│      I mark these as read?"                                          │
│                                                                       │
│  4. Closing                                                          │
│     "That's everything important. You have 10 other emails I can   │
│      summarize if you'd like, or we can skip to the end. What would │
│      you prefer?"                                              │
│                                                                 │
│  Navigation Commands:                                           │
│  - "Skip this topic"                                            │
│  - "Go deeper" (read full email)                                │
│  - "Next item"                                                  │
│  - "Repeat that"                                                │
│  - "Pause briefing"                                             │
│  - "Stop"                                                       │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

---

7. Application Layer

7.1 Mobile App Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                      Mobile App (React Native)                  │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  Navigation (React Navigation)                                  │
│                                                                 │
│  ┌────────────┐   ┌────────────┐   ┌────────────┐   ┌────────────┐    │
│  │  Welcome   │→ │ Onboarding │→ │  Briefing  │   │  Settings  │    │
│  │  Screen    │   │   Flow     │   │    Room    │   │            │    │
│  └────────────┘   └────────────┘   └────────────┘   └────────────┘    │
│                                          │                      │
│                                          │                      │
│  ┌───────────────────────────────────────▼──────────────────┐   │
│  │               LiveKit Room Component                     │   │
│  │  (@livekit/react-native SDK)                             │   │
│  │                                                          │   │
│  │  <LiveKitRoom url={...} token={...}>                     │   │
│  │    <RoomAudioRenderer />                                 │   │
│  │    <ConnectionQualityIndicator />                        │   │
│  │    <PTTButton onPress={...} />                           │   │
│  │  </LiveKitRoom>                                          │   │
│  └──────────────────────────────────────────────────────────┘   │
│                                                                 │
│  State Management (Zustand)                                     │
│  ┌──────────────────────────────────────────────────────────┐   │
│  │  useAuthStore                                            │   │
│  │  - OAuth tokens (secure storage)                         │   │
│  │  - Connected accounts (Outlook, Gmail)                   │   │
│  │                                                          │   │
│  │  usePreferencesStore                                     │   │
│  │  - VIPs, keywords, topics                                │   │
│  │  - Verbosity setting                                     │   │
│  │  - Quiet mode                                            │   │
│  │                                                          │   │
│  │  useBriefingStore                                        │   │
│  │  - Current session state                                 │   │
│  │  - Connection quality                                      │     │
│  └────────────────────────────────────────────────────────────┘     │
│                                                                     │
│  Services                                                           │
│  ┌────────────────────────────────────────────────────────────┐     │
│  │  livekit-token.ts                                          │     │
│  │  - Fetch room token from backend API                       │     │
│  │                                                            │     │
│  │  offline-queue.ts                                          │     │
│  │  - Queue failed commands (mark read, flag, etc.)           │     │
│  │  - Retry on network restore                                │     │
│  │  - Persist to AsyncStorage                                 │     │
│  └────────────────────────────────────────────────────────────┘     │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

7.2 Desktop App Architecture

```
┌─────────────────────────────────────────────────────────────────────┐
│                     Desktop App (Electron + React)                  │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│  Main Process (Node.js)                                             │
│  ┌────────────────────────────────────────────────────────────┐     │
│  │  - Window management                                       │     │
│  │  - Deep link handling (OAuth callbacks)                    │     │
│  │  - System tray integration                                 │     │
│  │  - Auto-updater                                            │     │
│  └────────────────────────────────────────────────────────────┘     │
│                                                                     │
│  Renderer Process (React)                                           │
│  ┌────────────────────────────────────────────────────────────┐     │
│  │  Pages:                                                    │     │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │     │
│  │  │ Drafts List  │  │ Draft Detail │  │   Activity   │      │     │
│  │  │              │→ │              │  │   History    │      │     │
│  │  │ - Filters    │  │ - Edit draft │  │ - Per session│      │     │
│  │  │ - Sort       │  │ - Approve &  │  │ - All time   │      │     │
│  │  │ - Count      │  │   Send       │  │ - Undo       │      │     │
│  │  │   badge      │  │ - Thread ctx │  │ - Export     │      │     │
│  │  └──────────────┘  └──────────────┘  └──────────────┘      │     │
│  │                                                            │     │
│  │  Services:                                                 │     │
│  │  ┌──────────────────────────────────────────────────┐      │     │
│  │  │  draft-sync.ts                                   │      │     │
│  │  │  - Poll backend for new drafts                   │      │     │
│  │  │  - Real-time updates via WebSocket               │      │     │
│  │  │                                                  │      │     │
│  │  │  audit-trail.ts                                  │      │     │
│  │  │  - Store actions in encrypted local DB           │      │     │
│  │  │  - 30-day retention (configurable)               │      │     │
│  │  │  - Export to CSV/JSON                          │      │     │
│  │  │                                                │      │     │
│  │  │  preferences-sync.ts                           │      │     │
│  │  │  - Sync VIPs, keywords with mobile             │      │     │
│  │  │  - Conflict resolution: last-write-wins        │      │     │
│  │  └────────────────────────────────────────────────┘      │     │
│  └──────────────────────────────────────────────────────────┘     │
│                                                                   │
└───────────────────────────────────────────────────────────────────┘
```

7.3 Backend API Architecture

```
┌───────────────────────────────────────────────────────────────────┐
│                    Backend API (Express/Fastify)                  │
├───────────────────────────────────────────────────────────────────┤
│                                                                   │
│  Routes                                                           │
│  ┌──────────────────────────────────────────────────────────┐     │
│  │  POST   /auth/microsoft/callback                         │     │
│  │  POST   /auth/google/callback                            │     │
│  │  - Handle OAuth redirects                                │     │
│  │  - Exchange code for tokens                              │     │
│  │  - Store in secure storage                               │     │
│  │  - Return JWT for API access                             │     │
│  │                                                          │     │
│  │  POST   /livekit/token                                   │     │
│  │  - Generate room access token                            │     │
│  │  - Set user identity, permissions                        │     │
│  │  - TTL: 1 hour                                           │     │
│  │                                                          │     │
│  │  GET    /sync/drafts                                     │     │
│  │  POST   /sync/drafts                                     │     │
│  │  - CRUD for draft references                             │     │
│  │  - Filter by isPendingReview                             │     │
│  │                                                          │     │
│  │  GET    /sync/preferences                                │     │
│  │  PUT    /sync/preferences                                │     │
│  │  - Sync VIPs, keywords, topics                           │     │
│  │  - Merge strategy: last-write-wins                       │     │
│  │                                                          │     │
│  │  POST   /webhooks/livekit                                │     │
│  │  - Room events (participant joined/left)                 │     │
│  │  - Track published/unpublished                           │     │
│  │  - Analytics logging                                     │     │
│  └──────────────────────────────────────────────────────────┘     │
│                                                                   │
│  Middleware                                                       │
│  ┌──────────────────────────────────────────────────────────┐     │
│  │  JWT Authentication                                      │     │
│  │  - Verify JWT on all protected routes                    │     │
│  │  - Extract userId from token                             │     │
│  │                                                             │     │
│  │  Rate Limiting                                              │     │
│  │  - Per-user limits (100 req/min)                            │     │
│  │  - Global limits (10k req/min)                              │     │
│  │                                                             │     │
│  │  Error Handling                                             │     │
│  │  - Structured error responses                               │     │
│  │  - Logging with @nexus-aec/logger                           │     │
│  └─────────────────────────────────────────────────────────────┘     │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

---

8. Data Flow & Interactions

8.1 End-to-End: Voice Command Execution

```
1. User speaks: "Flag all emails from John as priority"
        │
        ▼
2. Mobile App (@livekit/react-native)
   - Captures audio via microphone
   - Sends audio track to LiveKit Room
        │
        ▼
3. LiveKit Cloud
   - Routes audio to Deepgram STT plugin
   - Transcript: "Flag all emails from John as priority"
   - Sends transcript to Backend Agent
        │
        ▼
4. Backend Agent (GPT-4o Reasoning Loop)
   - Parses intent: "flag_priority_vip"
   - Extracts parameters: { name: "John" }
   - Searches contacts for "John" (disambiguate if multiple)
   - Assumes "John Smith" (most frequent contact)
        │
        ▼
5. Email Provider Layer
   - UnifiedInboxService.searchByContact("John Smith")
   - Returns: [email1, email2, email3] (3 emails from John)
   - For each email:
       - Determine source (OUTLOOK or GMAIL)
       - Call appropriate adapter:
           - OutlookAdapter.applyLabel(emailIds, "Priority")
           - GmailAdapter.applyLabel(emailIds, "Priority")
        │
        ▼
6. Response Generation (GPT-4o)
   - Generate natural language response:
     "Done. I've flagged 3 emails from John Smith as priority."
        │
        ▼
7. LiveKit Cloud
   - Send text to ElevenLabs TTS plugin
   - Generate audio stream
   - Send audio track to Mobile App
        │
        ▼
8. Mobile App
   - Play audio response
   - Update UI (show confirmation)
        │
        ▼
9. Shadow Processor (Background)
   - Update Redis DriveState:
       - lastAction: "flag_priority_vip"
       - lastActionTarget: "John Smith"
       - itemsProcessed: +3
        │
        ▼
10. Audit Trail (Desktop App)
    - Log action:
        - type: "apply_label"
        - target: ["email1", "email2", "email3"]
        - label: "Priority"
        - timestamp: "2026-01-09T10:15:23Z"
        - outcome: "success"
```

8.2 Typical User Journey: Morning Briefing

```
1. User opens mobile app
   └─► Tap "Start Briefing"

2. Mobile app requests LiveKit room token from Backend API
   └─► POST /livekit/token → Returns token

3. Mobile app connects to LiveKit room
   └─► Joins as participant "user-123"

4. Backend Agent auto-joins same room
   └─► Joins as participant "agent"

5. Agent starts briefing generation:
   a. Fetch emails from UnifiedInboxService
      └─► Polls OutlookAdapter + GmailAdapter in parallel
      └─► Returns 27 new emails (merged timeline)

   b. Run red flag detection (Tier 1: Ephemeral)
      └─► Keyword matcher, VIP detector, velocity, calendar proximity
      └─► Returns 3 red flags

   c. Cluster emails by topic
```

```
      └─► Returns 5 topics

   d. Generate briefing script (GPT-4o)
      └─► "Good morning! You have 27 new emails. I've found 3 red flags..."

   e. Send script to ElevenLabs TTS
      └─► Stream audio to room

6. User hears briefing
   └─► Listens to first red flag

7. User interrupts (barge-in): "Flag that for follow-up"
   └─► LiveKit detects user speech
   └─► Agent pauses TTS
   └─► Deepgram transcribes command
   └─► GPT-4o executes: flag_followup(emailId: "current")
   └─► Agent responds: "Done"
   └─► Resumes briefing

8. User says: "Skip to next topic"
   └─► GPT-4o function call: skip_topic()
   └─► Shadow Processor updates Redis:
       - currentTopicIndex: 1 → 2
   └─► Agent continues with topic 2

9. User loses connection (enters tunnel)
   └─► Mobile app detects ConnectionQuality.Lost
   └─► Shows "Connection lost" overlay
   └─► Shadow Processor saves last position to Redis:
       - lastPosition: 145000 (145 seconds into briefing)

10. User regains connection
    └─► Mobile app auto-reconnects to room
    └─► Fetches DriveState from Redis
    └─► Agent resumes from position 145000
    └─► "Welcome back. You were at topic 2, item 3..."

11. Briefing completes
    └─► Agent: "That's everything. Have a great day!"
    └─► User ends session
    └─► Redis DriveState expires after 24 hours
```

9. Package Dependencies

9.1 Dependency Graph

```
┌─────────────────────────────────────────────────────────────────────┐
│                       Application Layer                               │
│                                                                     │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐            │
│  │  mobile  │  │ desktop  │  │   api    │  │  agent   │            │
│  └────┬─────┘  └────┬─────┘  └────┬─────┘  └────┬─────┘            │
│       │             │              │              │                   │
│       └─────────────┴──────────────┴──────────────┘                   │
│                                    │                                  │
│                                    │ depends on                       │
│                                    ▼                                  │
│  ┌────────────────────────────────────────────────────────────┐       │
│  │              Shared Packages Layer                         │       │
│  │  ┌────────────────┐  ┌────────────────┐  ┌─────────────┐   │       │
│  │  │email-providers │  │intelligence    │  │livekit-agent│   │       │
│  │  └───────┬────────┘  └────────┬───────┘  └────────┬────┘   │       │
│  │          │                    │                   │        │       │
│  │          └────────────────────┼───────────────────┘        │       │
│  │                               │                            │       │
│  │                               │ depends on                 │       │
│  │                               ▼                            │       │
│  │  ┌────────────────┐  ┌────────────────┐  ┌─────────────┐   │       │
│  │  │secure-storage  │  │encryption      │  │logger       │   │       │
│  │  └───────┬────────┘  └────────┬───────┘  └────────┬────┘   │       │
│  │          │                    │                   │        │       │
│  │          └────────────────────┼───────────────────┘        │       │
│  │                               │                            │       │
│  │                               │ depends on                 │       │
│  │                               ▼                            │       │
│  │  ┌──────────────────────────────────────────────────┐      │       │
│  │  │          shared-types (root, no deps)            │      │       │
│  │  └──────────────────────────────────────────────────┘      │       │
│  └────────────────────────────────────────────────────────────┘       │
└───────────────────────────────────────────────────────────────────────┘
```

9.2 Build Order (Managed by Turborepo)

Turborepo automatically determines build order based on `package.json` dependencies:

```
1. shared-types      (no dependencies)
2. encryption        (depends on: shared-types)
3. logger            (depends on: shared-types)
4. secure-storage    (depends on: shared-types, encryption)
5. email-providers   (depends on: shared-types, encryption, logger, secure-storage)
6. intelligence      (depends on: shared-types, logger, email-providers)
7. livekit-agent     (depends on: shared-types, logger, intelligence)
8. api               (depends on: shared-types, logger, email-providers, intelligence)
9. mobile            (depends on: shared-types, email-providers, intelligence)
10. desktop          (depends on: shared-types, email-providers, intelligence)
```

Command: `pnpm turbo run build` handles this automatically.

---

10. Security Architecture

10.1 Security Layers

```
┌─────────────────────────────────────────────────────────────────────┐
│                      Layer 1: Transport Security                     │
├─────────────────────────────────────────────────────────────────────┤
│  - HTTPS/WSS for all network communication                          │
│  - TLS 1.3 minimum                                                  │
│  - Certificate pinning (mobile apps)                                │
│  - LiveKit Cloud: End-to-end encrypted WebRTC (SRTP)               │
└─────────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌─────────────────────────────────────────────────────────────────────┐
│                      Layer 2: Authentication                         │
├─────────────────────────────────────────────────────────────────────┤
│  OAuth 2.0 with PKCE                                                │
│  ┌───────────────────────────────────────────────────────────┐      │
│  │ Microsoft Graph (Outlook)                                 │      │
│  │ - Authorization Code Flow + PKCE                          │      │
│  │ - Scopes: Mail.Read, Mail.ReadWrite, Calendars.Read     │      │
│  │ - Refresh tokens stored in secure storage                 │      │
│  │                                                            │      │
│  │ Google APIs (Gmail)                                       │      │
│  │ - Authorization Code Flow + PKCE                          │      │
│  │ - Scopes: gmail.readonly, gmail.modify, calendar.readonly│      │
│  │ - Refresh tokens stored in secure storage                 │      │
│  └───────────────────────────────────────────────────────────┘      │
│                                                                       │
│  JWT for Backend API Access                                         │
│  - Signed with HS256 (secret key)                                   │
│  - Payload: { userId, exp, iat }                                    │
│  - TTL: 1 hour (refresh via refresh token)                          │
└─────────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌─────────────────────────────────────────────────────────────────────┐
│                      Layer 3: Data Encryption                        │
├─────────────────────────────────────────────────────────────────────┤
│  At Rest                                                             │
│  ┌───────────────────────────────────────────────────────────┐      │
│  │ AES-256-GCM (via @nexus-aec/encryption)                   │      │
│  │ - Master key from ENCRYPTION_MASTER_KEY env var           │      │
│  │ - Encrypted data:                                         │      │
│  │   - OAuth tokens                                          │      │
│  │   - User preferences (VIPs, keywords)                     │      │
│  │   - Audit trail entries                                   │      │
│  │ - Platform-specific secure storage:                       │      │
│  │   - iOS/macOS: Keychain                                   │      │
│  │   - Android: EncryptedSharedPreferences                   │      │
│  │   - Windows: Credential Manager                           │      │
│  │   - Linux: Secret Service API                             │      │
│  └───────────────────────────────────────────────────────────┘      │
│                                                                       │
│  In Transit                                                          │
│  - LiveKit WebRTC: SRTP (end-to-end encrypted audio)                 │
│  - HTTPS for all API calls                                           │
│  - No email content stored (Tier 1: ephemeral only)                  │
└──────────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌──────────────────────────────────────────────────────────────────────┐
│                      Layer 4: Privacy & PII Protection               │
├──────────────────────────────────────────────────────────────────────┤
│  Data Minimization                                                   │
│  - Email content: Never stored persistently (ephemeral only)         │
│  - Session state: 24-hour TTL in Redis                               │
│  - Audit trail: 30-day default retention (configurable)              │
│  - Knowledge base: Only asset metadata, no user data                 │
│                                                                      │
│  PII Filtering in Logs                                               │
│  - @nexus-aec/logger filters:                                        │
│    - Email addresses                                                 │
│    - Names                                                           │
│    - Message content                                                 │
│  - Logs only: hashed user IDs, counts, durations                     │
│                                                                      │
│  User Controls                                                       │
│  - Privacy Dashboard: Show all stored data                           │
│  - "Clear My Data" button: Delete all user data                      │
│  - OAuth revocation: Link to provider settings                       │
│  - Export audit trail: CSV/JSON for transparency                     │
└──────────────────────────────────────────────────────────────────────┘
                                │
                                ▼
┌──────────────────────────────────────────────────────────────────────┐
│                      Layer 5: Authorization & Access Control         │
├──────────────────────────────────────────────────────────────────────┤
│  Draft Approval Workflow (Safety-First)                              │
│  ┌────────────────────────────────────────────────────────────┐      │
│  │ Voice command: "Send a reply"                              │      │
│  │       ↓                                                    │      │
│  │ Agent creates draft with isPendingReview: true             │      │
│  │       ↓                                                    │      │
│  │ Desktop app shows draft for review                         │      │
│  │       ↓                                                    │      │
│  │ User approves → Send via appropriate adapter               │      │
│  │ User rejects → Delete draft                                │      │
│  └────────────────────────────────────────────────────────────┘      │
│                                                                      │
│  Confirmation Verbosity (Risk-Based)                                 │
│  - Low risk (mark read): "Done" (no confirmation)                    │
│  - Medium risk (flag, move): "Flagged 3 emails" (count)              │
│  - High risk (draft, delete): Require desktop approval               │
│                                                                      │
│  Undo Window (24 hours)                                              │
│  - All actions stored in audit trail                                  │
│  - Desktop app: Undo individual or batch actions                      │
│  - After 24 hours: Undo disabled (action finalized)                   │
└───────────────────────────────────────────────────────────────────────┘
```

10.2 Threat Model

Threat	Mitigation
OAuth token theft	Secure storage (Keychain, etc.), never log tokens
Man-in-the-middle	TLS 1.3, certificate pinning, HTTPS everywhere
PII leak in logs	PII filtering via @nexus-aec/logger
Unauthorized email access	OAuth scopes limited to read + draft only, no send without approval
Session hijacking	JWT with short TTL (1h), rotate on refresh
Email content exposure	Tier 1 ephemeral only, never persist email bodies
Unintended actions	Confirmation verbosity, desktop draft approval, undo window
Credential stuffing	Rate limiting (100 req/min per user)

---

11. Deployment Architecture

11.1 Local Development

```
Developer Machine
├─ Docker Compose (infra/docker-compose.yml)
│  ├─ Redis (port 6379)
│  ├─ PostgreSQL + pgvector (port 5432)
│  ├─ Redis Commander (port 8081) [optional, profile: tools]
│  └─ pgAdmin (port 5050) [optional, profile: tools]
│
├─ pnpm dev (all packages in watch mode)
│  ├─ packages/shared-types
│  ├─ packages/encryption
│  ├─ packages/logger
│  ├─ packages/secure-storage
│  └─ packages/email-providers
│
└─ External Services (cloud)
   ├─ LiveKit Cloud (wss://your-app.livekit.cloud)
   ├─ Deepgram API (STT)
   ├─ ElevenLabs API (TTS)
   ├─ OpenAI API (GPT-4o + embeddings)
   ├─ Microsoft Graph (Outlook)
   └─ Google APIs (Gmail)
```

11.2 Production Deployment

```
┌──────────────────────────────────────────────────────────────────────┐
│                        Client Devices                                 │
│                                                                      │
│  ┌───────────────┐                    ┌───────────────┐              │
│  │  Mobile App   │                    │ Desktop App   │              │
│  │(React Native)│                     │  (Electron)   │              │
│  └───────┬───────┘                    └───────┬───────┘              │
│          │                                    │                       │
│          │ HTTPS/WSS                          │ HTTPS                 │
└──────────┼────────────────────────────────────┼───────────────────────┘
           │                                    │
           │                                    │
           │                                    │
┌──────────▼────────────────────────────────────▼───────────────────────┐
│                        Cloud Infrastructure                             │
│                                                                       │
│  ┌─────────────────────────────────────────────────────────────┐      │
│  │  Load Balancer (HTTPS termination, SSL offload)            │      │
│  └─────────────────────┬───────────────────────────────────────┘      │
│                        │                                                │
│           ┌────────────┴─────────────────┐                            │
│           │                              │                            │
│  ┌────────▼─────────┐           ┌────────▼─────────┐                  │
│  │  Backend API     │           │  LiveKit Cloud   │                  │
│  │  (Kubernetes)    │           │   (Managed)      │                  │
│  │                  │           │                  │                  │
│  │  Deployment:     │           │  - Rooms         │                  │
│  │  - Replicas: 3   │           │  - Agents        │                  │
│  │  - Auto-scale    │           │  - STT/TTS       │                  │
│  │  - Rolling       │           └──────────────────┘                  │
│  │    updates       │                                                 │
│  └─────────┬────────┘                                                 │
│            │                                                            │
│  ┌─────────▼───────────────────────────────────────────────────┐      │
│  │           LiveKit Agent (Kubernetes)                        │      │
│  │                                                              │      │
│  │  Deployment:                                                │      │
│  │  - Replicas: Auto-scale based on active rooms               │      │
│  │  - Min: 2, Max: 50                                          │      │
│  │  - CPU: 2 cores, Memory: 4GB per pod                        │      │
│  │  - Health checks: /health endpoint                          │      │
│  │                                                              │      │
│  │  Container: nexus-aec-agent:latest                          │      │
│  │  - Node.js 20 runtime                                       │      │
│  │  - LiveKit Agents SDK                                       │      │
│  │  - GPT-4o client                                            │      │
│  └──────────────────────────────────────────────────────────────┘      │
│                                                                         │
│  ┌─────────────────────────────────────────────────────────────┐      │
│  │                    Managed Services                         │      │
│  │                                                            │      │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │      │
│  │  │ Redis Cloud  │  │  Supabase    │  │   OpenAI     │      │      │
│  │  │              │  │   Cloud      │  │    API       │      │      │
│  │  │ - Session    │  │ - PostgreSQL │  │ - GPT-4o     │      │      │
│  │  │   state      │  │ - pgvector   │  │ - Embeddings │     │      │
│  │  │ - TTL: 24h   │  │ - Vector     │  └──────────────┘     │      │
│  │  │              │  │   store      │                       │      │
│  │  └──────────────┘  └──────────────┘                       │      │
│  └────────────────────────────────────────────────────────────┘      │
└───────────────────────────────────────────────────────────────────────┘
```

11.3 Scaling Strategy

Component	Scaling Strategy	Rationale
Backend API	Horizontal (3-10 replicas)	Stateless, scale based on HTTP requests/sec
LiveKit Agent	Horizontal (auto-scale 2-50)	Scale based on active rooms, CPU-intensive
Redis	Vertical (managed service)	Session state is small, latency critical
Supabase	Managed (auto-scaling)	Vector queries scale with data size
LiveKit Cloud	Managed (auto-scaling)	Handles WebRTC media routing automatically

---

12. Design Decisions & Rationale

12.1 Why LiveKit (Not Custom WebRTC)?

Decision: Use LiveKit Cloud for all voice processing.

Rationale:

- WebRTC is complex (signaling, STUN/TURN, codec negotiation, network resilience)
- LiveKit provides production-ready infrastructure:
  - Auto-scaling media servers
  - Built-in STT/TTS plugins (Deepgram, ElevenLabs)
  - Network resilience (packet loss recovery, adaptive bitrate)
  - Connection quality monitoring
  - Barge-in support via VAD
- Reduces development time from months to weeks
- Eliminates need for custom audio pipeline maintenance

Trade-offs:

- Vendor lock-in to LiveKit (mitigated: open-source, self-hostable)
- Monthly cost based on usage (acceptable for MVP)

12.2 Why Unified Adapter Pattern (Not Direct API Calls)?

Decision: Abstract Outlook and Gmail behind `EmailProvider` interface.

Rationale:

- Provider APIs are different (Graph vs REST)
- Normalization simplifies application logic
- Easy to add new providers (Yahoo, ProtonMail, etc.)
- Single interface to test and mock

- Source tagging enables smart draft routing

Trade-offs:

- Abstraction overhead (mitigated: thin adapter layer)
- Potential loss of provider-specific features (acceptable for MVP)

12.3 Why Three-Tier Memory (Not Single Database)?

Decision: Ephemeral (in-memory) → Redis (session) → Supabase (knowledge).

Rationale:

- **Tier 1 (Ephemeral):** Email content is sensitive, discard after processing
- **Tier 2 (Redis):** Session state needs fast access (<10ms latency)
- **Tier 3 (Supabase):** Knowledge base requires vector search (pgvector)
- Performance: Hot path (briefing) uses in-memory only
- Privacy: Minimal data retention

Trade-offs:

- Complexity of managing three stores (mitigated: clear boundaries)
- Redis cost for session state (acceptable: TTL-based auto-expiry)

12.4 Why Desktop App for Draft Review (Not Mobile)?

Decision: Draft approval via desktop Electron app only.

Rationale:

- Safety: Large screen for reviewing draft content + thread context
- Deliberate action: Requires user to stop and focus (not in-motion)
- Audit trail: Desktop UI better suited for activity history
- Ergonomics: Easier to edit drafts on desktop

Trade-offs:

- Requires desktop installation (acceptable: enterprise use case)
- Cannot send emails purely from mobile (intentional safety feature)

12.5 Why Monorepo (Not Separate Repos)?

Decision: Single monorepo with Turborepo + pnpm workspaces.

Rationale:

- Shared types across all packages (single source of truth)
- Atomic commits (change shared types + consumers in one PR)
- Faster CI (Turborepo caching and parallel builds)
- Easier refactoring (grep across entire codebase)

Trade-offs:

- Larger repo size (mitigated: pnpm saves disk space)
- Learning curve for monorepo tools (acceptable: well-documented)

End of Architecture Documentation

For implementation details, see `.claude/RULES.md` For code conventions, see `.claude/CONVENTIONS.md` For workflows, see `.claude/WORKFLOWS.md`