

# Eklavya Architecture Document

**Version:** 1.0 **Last Updated:** January 2026 **Status:** Pre-Implementation Review

## Table of Contents

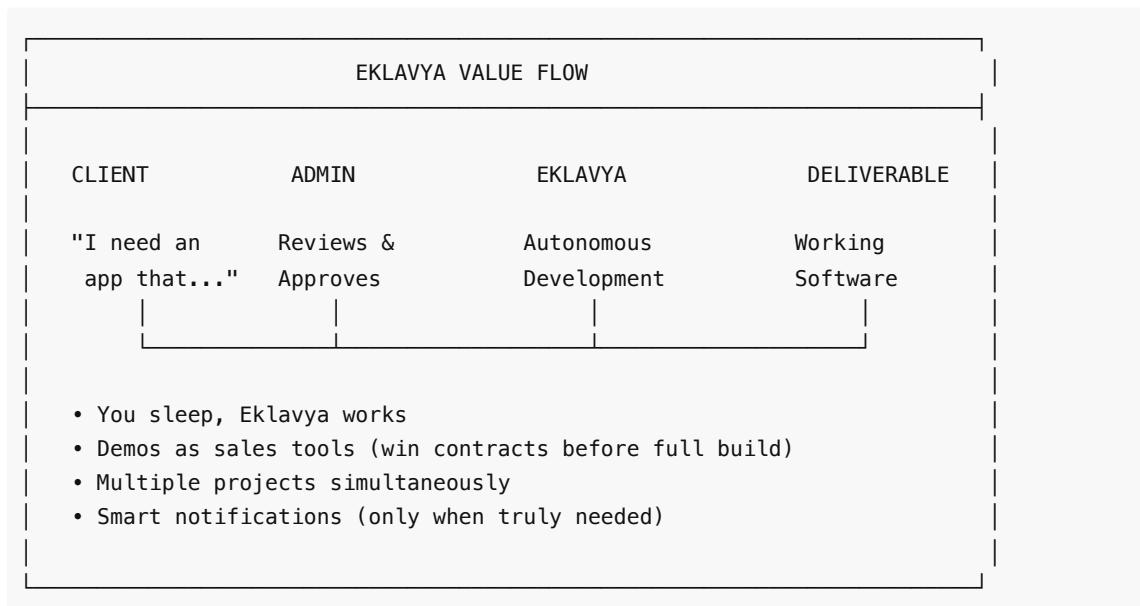
1. [Executive Summary](#)
2. [System Architecture](#)
3. [Component Overview](#)
4. [Technology Stack](#)
5. [Agent System](#)
6. [Data Architecture](#)
7. [Communication Patterns](#)
8. [Security Model](#)
9. [Scalability Considerations](#)
10. [Integration Points](#)

## 1. Executive Summary

### What is Eklavya?

Eklavya is an **autonomous agent orchestration platform** designed to run a software development business. It takes project requirements, generates plans, builds demos for client approval, and executes full project builds - all with minimal human intervention.

### Core Value Proposition



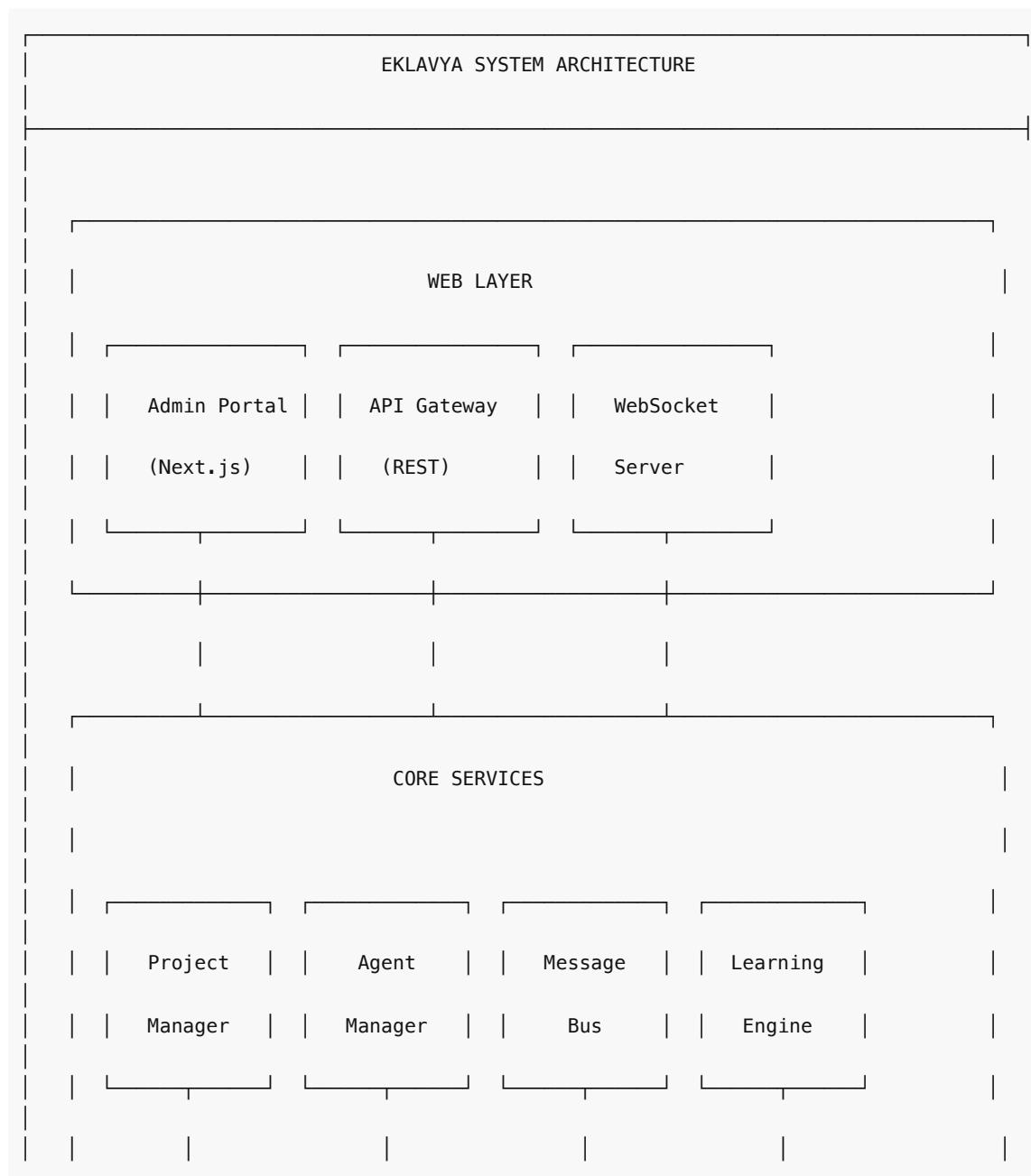
### Key Architectural Decisions

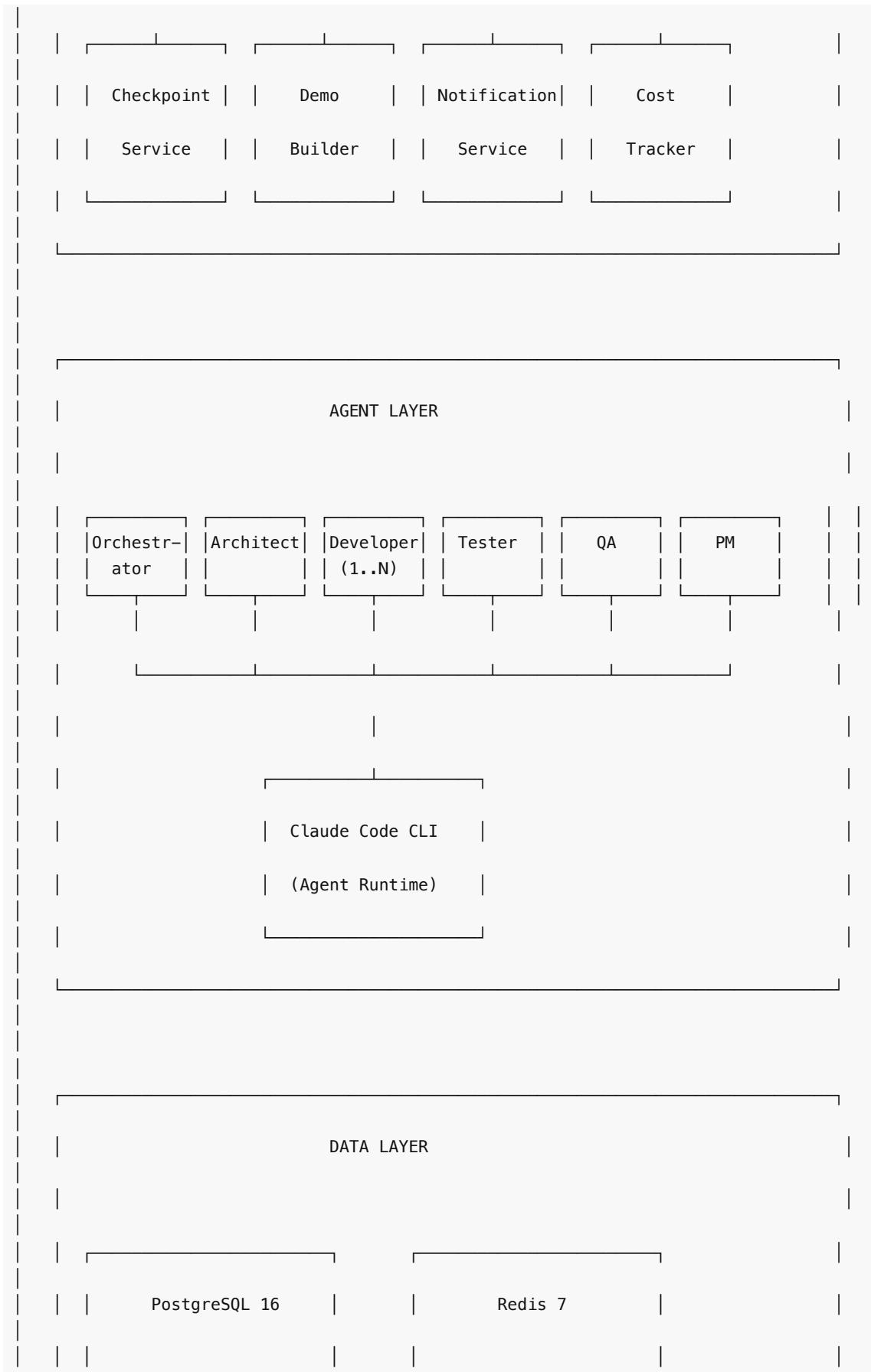
Decision	Choice	Why
Agent Runtime	Claude Code CLI	Battle-tested tools, sandboxing, proven reliability

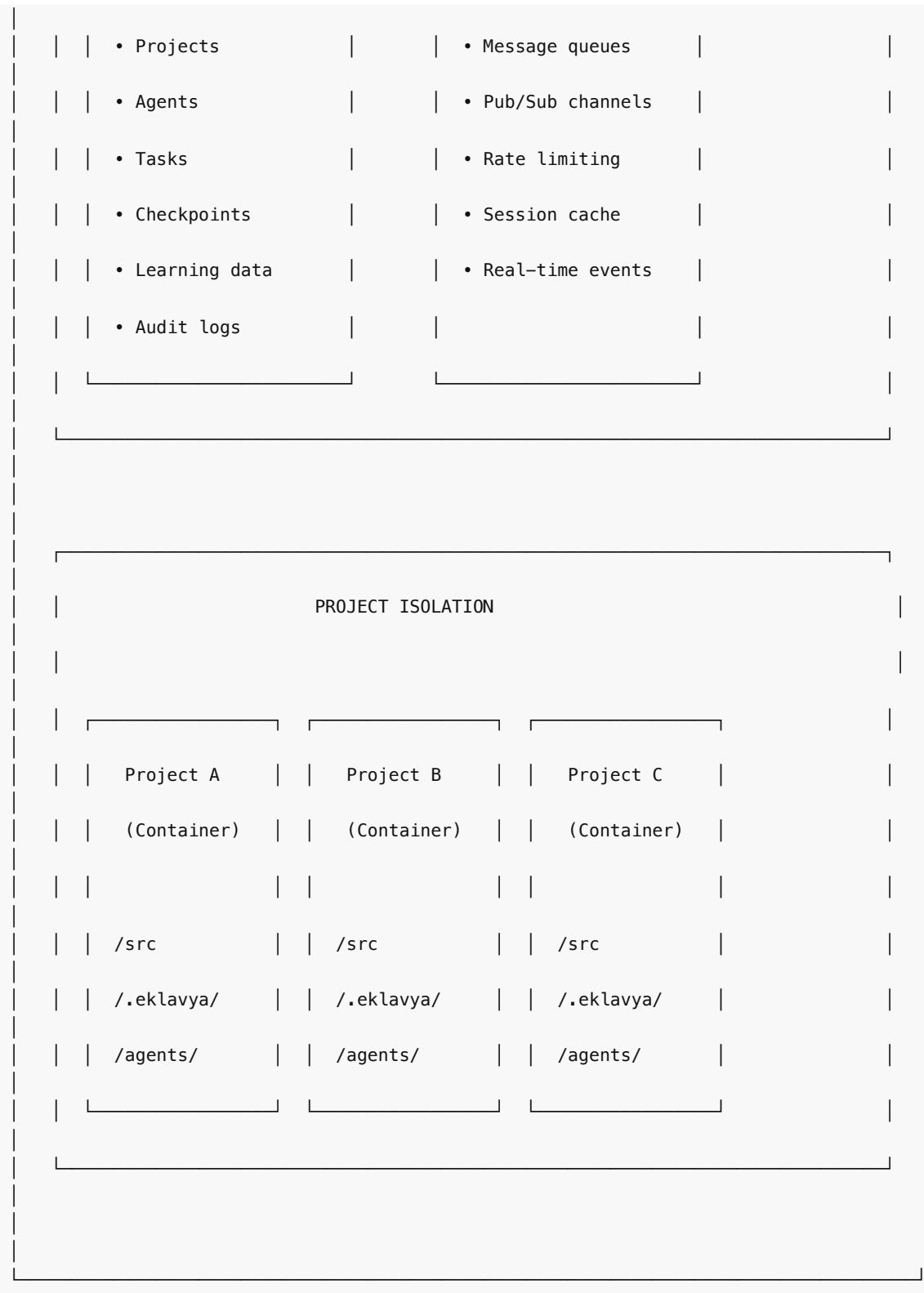
Database	PostgreSQL 16	JSONB flexibility, LISTEN/NOTIFY for real-time events
Message Queue	Redis 7	Fast pub/sub, caching, rate limiting
Web Framework	Next.js 14	Full-stack, real-time WebSocket support
Container	Docker	Project isolation, reproducible environments
AI Provider	Anthropic Claude	Best coding capability, tool use support

## 2. System Architecture

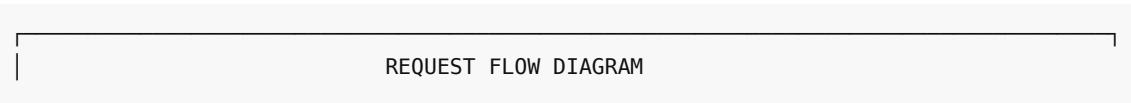
### High-Level Architecture

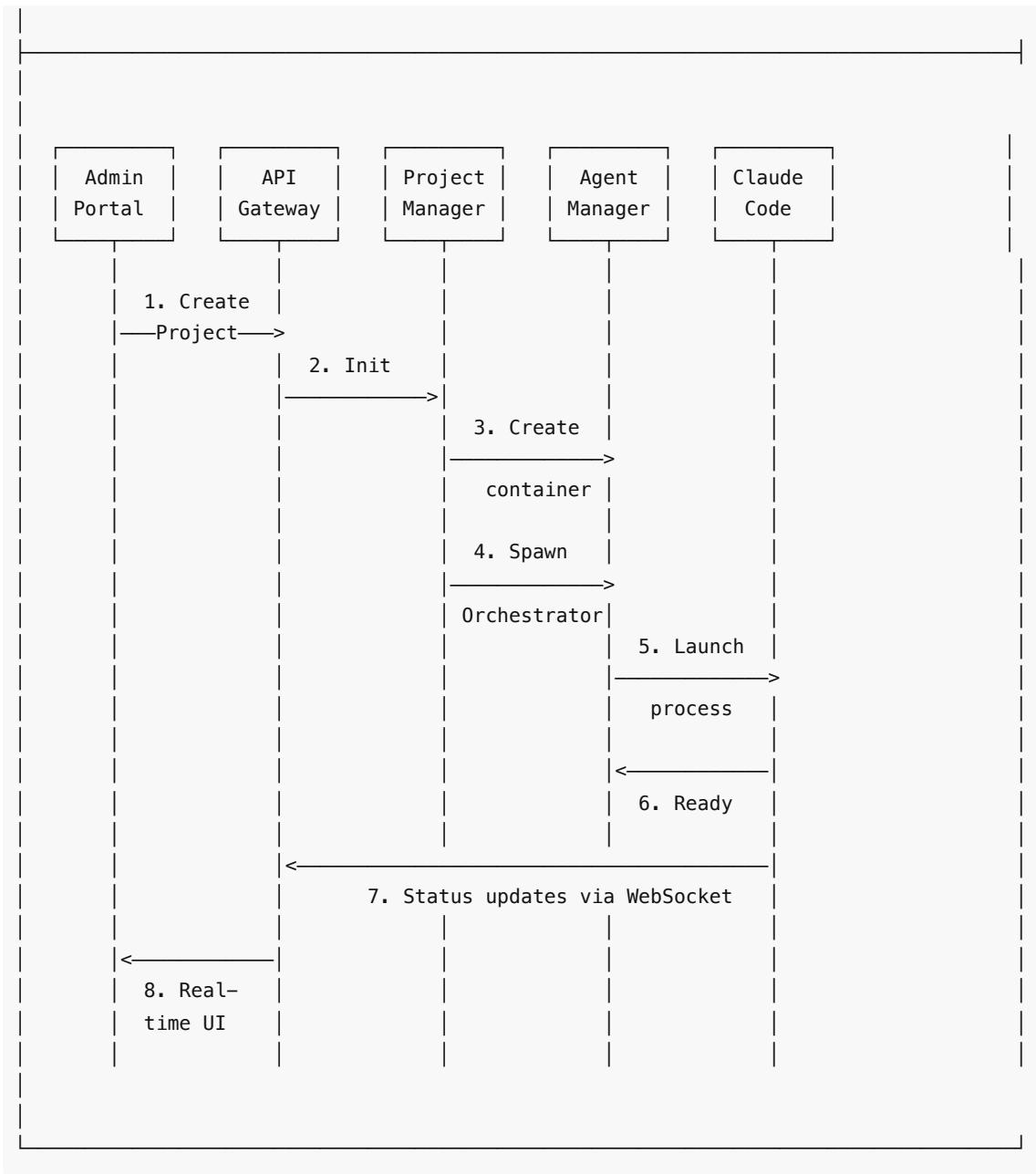






## Request Flow





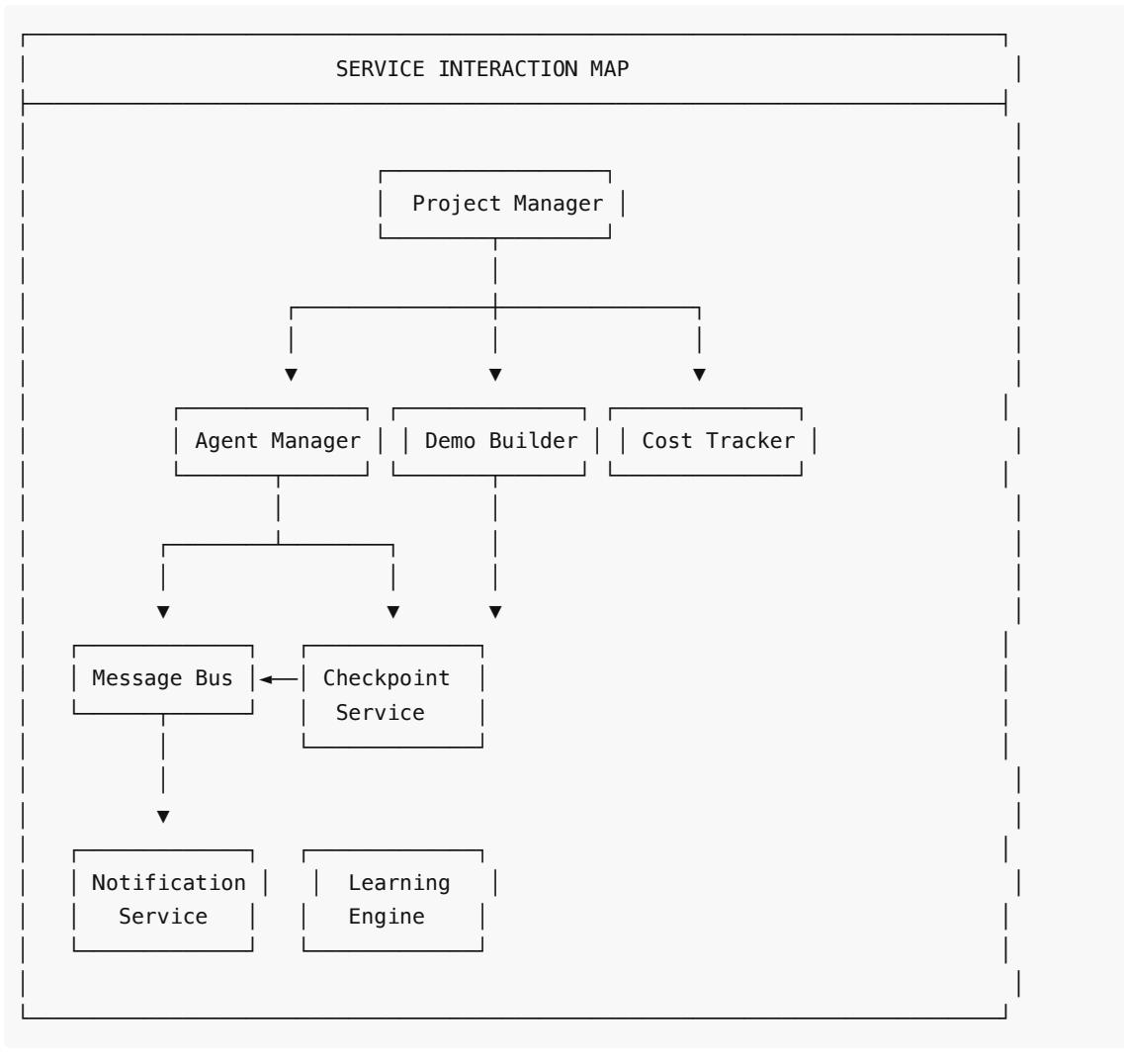
### 3. Component Overview

#### Core Services

Service	Responsibility	Key Functions
<b>Project Manager</b>	Project lifecycle management	Create, configure, archive projects
<b>Agent Manager</b>	Agent lifecycle control	Spawn, monitor, terminate agents
<b>Message Bus</b>	Inter-agent communication	Route messages, handle broadcasts
<b>Learning Engine</b>	Prompt optimization	Track outcomes, evolve prompts

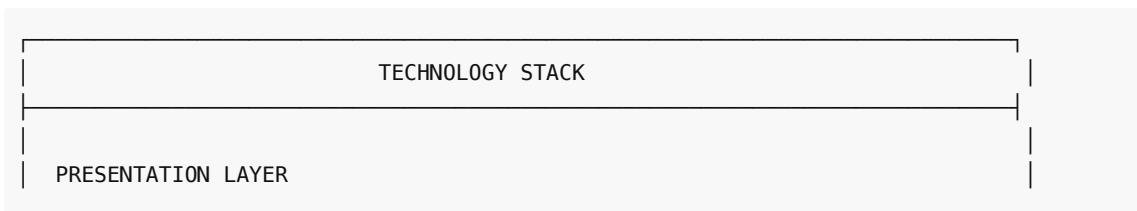
<b>Checkpoint Service</b>	State persistence	Save/restore agent state
<b>Demo Builder</b>	Demo orchestration	Parallel demo builds, preview URLs
<b>Notification Service</b>	Smart alerts	Prioritize, route, escalate
<b>Cost Tracker</b>	Budget management	Track spend, enforce limits

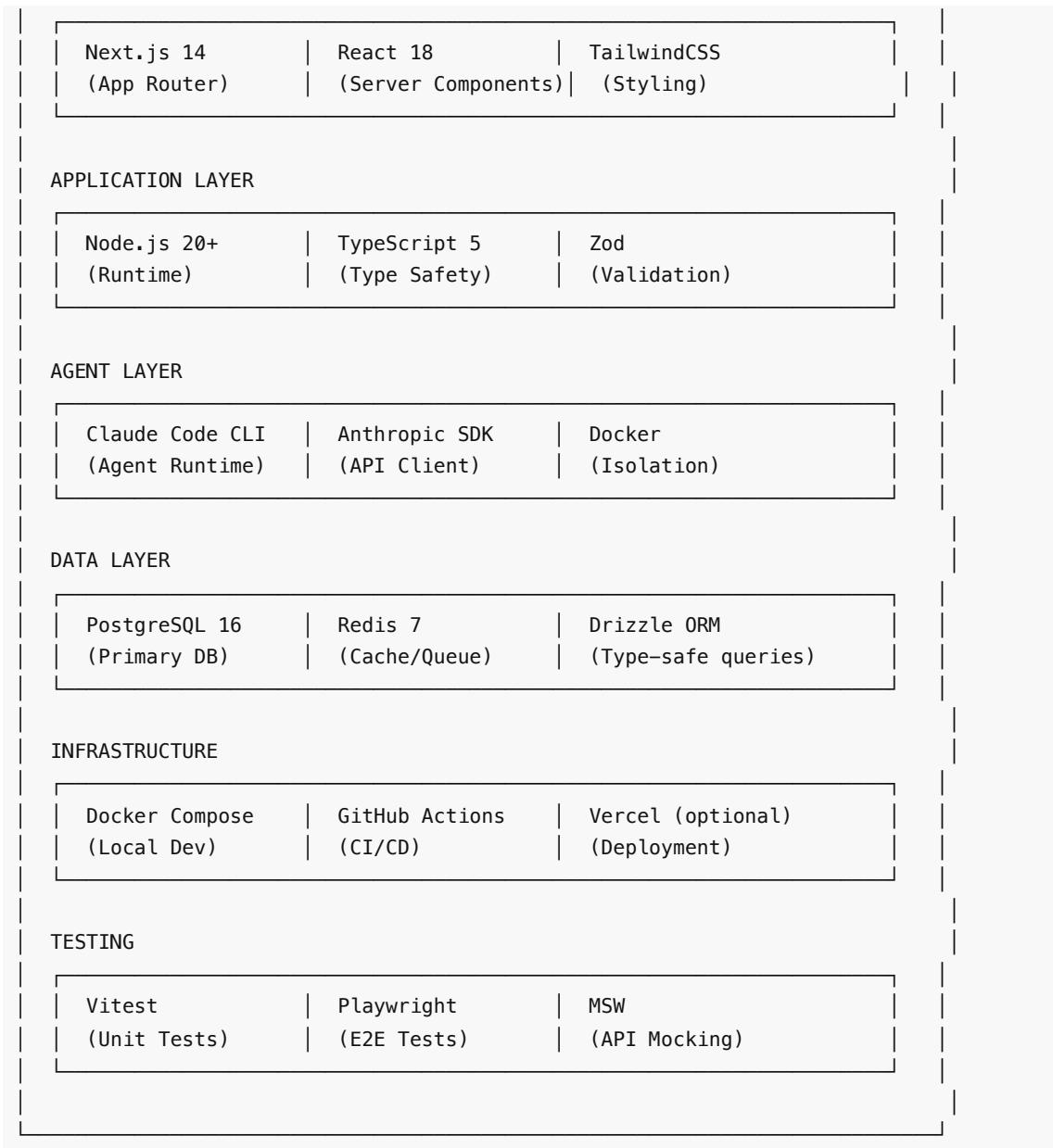
## Service Interactions



## 4. Technology Stack

### Runtime Environment



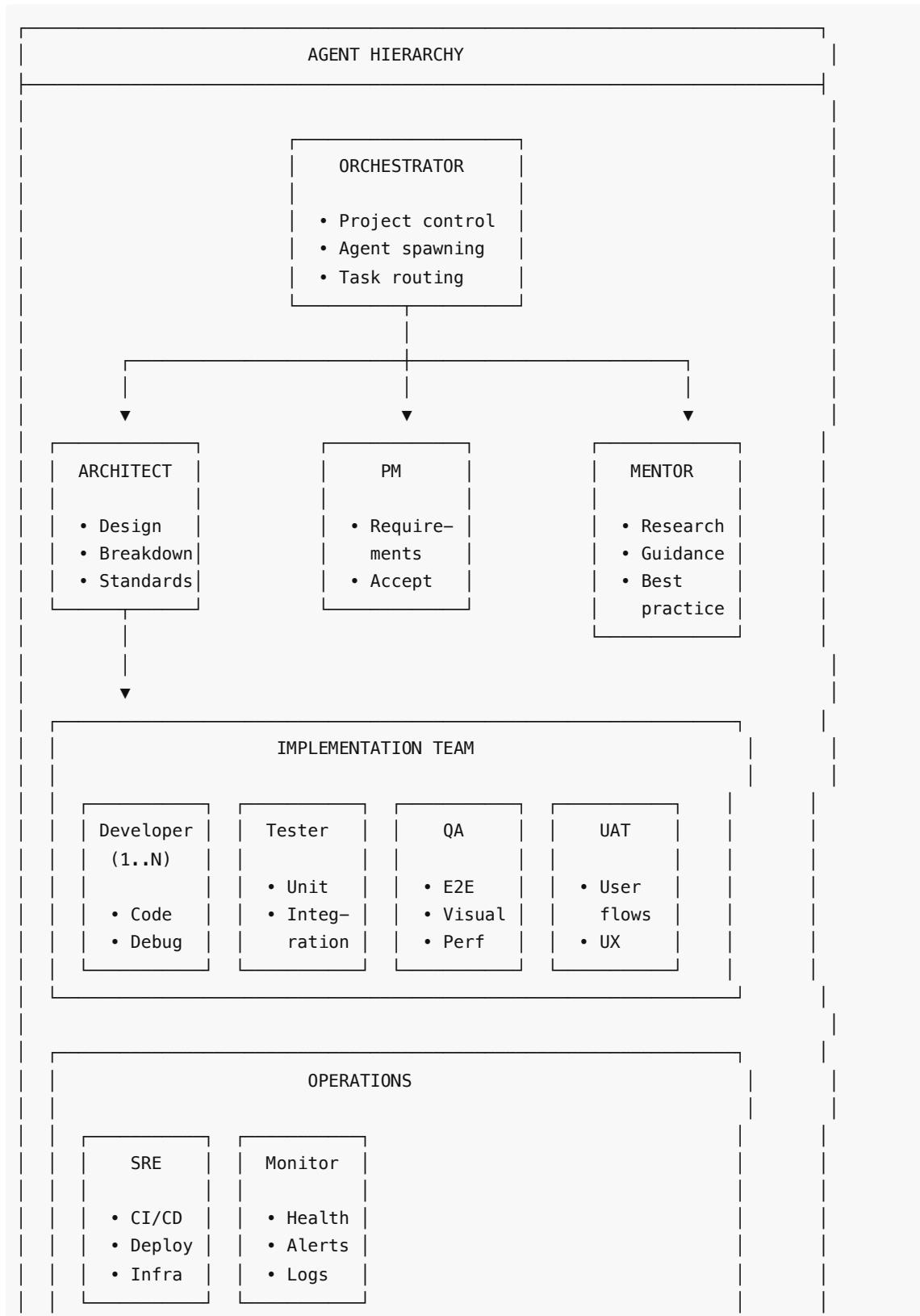


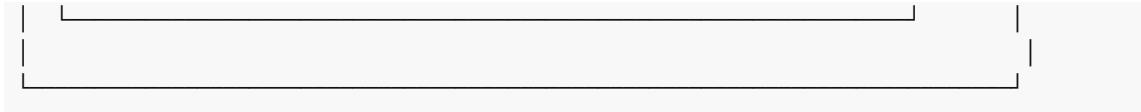
## Why These Choices?

Technology	Alternatives Considered	Why We Chose This
<b>Next.js 14</b>	Remix, SvelteKit	Best ecosystem, server components, Vercel integration
<b>PostgreSQL</b>	MySQL, MongoDB	JSONB flexibility, LISTEN/NOTIFY, mature ecosystem
<b>Redis</b>	RabbitMQ, Kafka	Simple, fast, perfect for our scale
<b>Claude Code</b>	Custom runtime	Battle-tested, sandboxed, full tool support
<b>Docker</b>	VM, bare metal	Reproducible, isolated, industry standard
<b>Drizzle</b>	Prisma, TypeORM	Type-safe, lightweight, SQL-like syntax

## 5. Agent System

### Agent Types and Roles

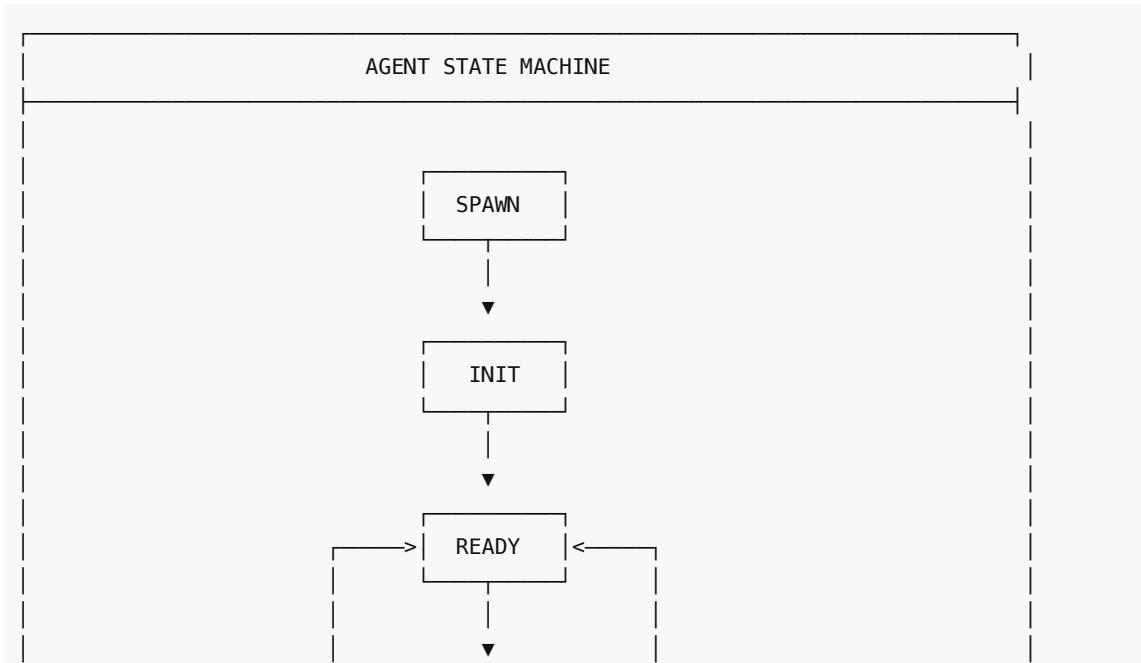


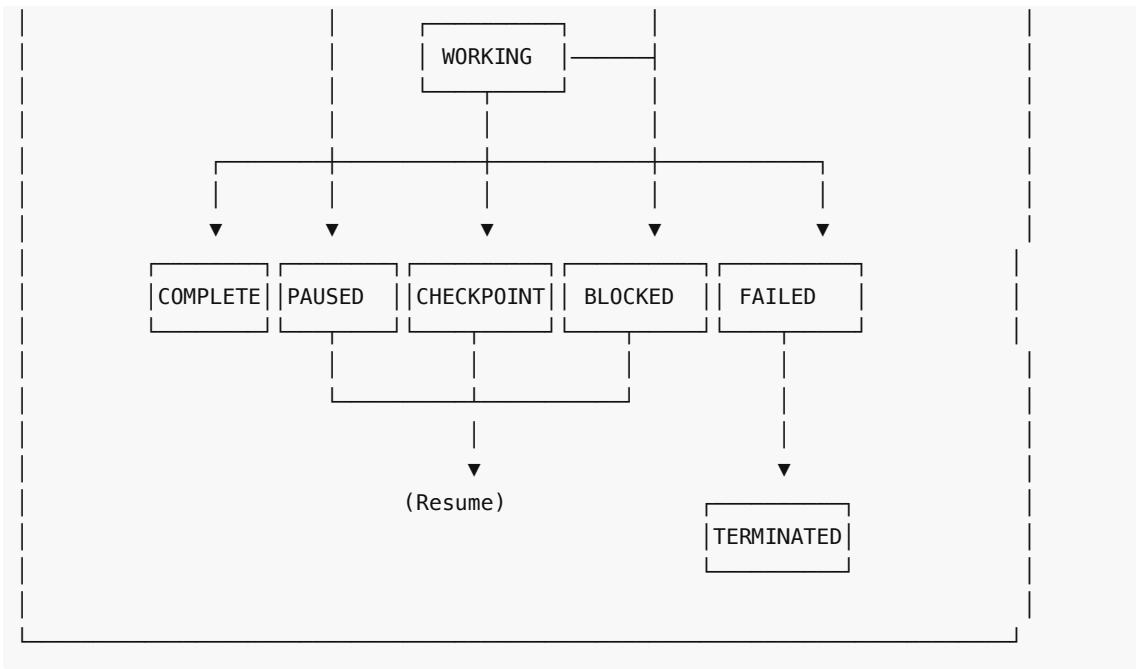


## Agent Permissions Matrix

Agent	File System	Git	Terminal	Browser	Spawn Agents	Network
Orchestrator	Read	Read	No	No	Yes	Internal
Architect	Read/Write (docs)	Read	No	No	No	Research
Developer	Full	Full	Full	No	No	Package mgr
Tester	Read/Write (tests)	Read	Test cmd	No	No	Test only
QA	Read	Read	Test cmd	Full	No	Test only
PM	Read/Write (docs)	Read	No	No	No	Research
UAT	Read	No	No	Full	No	Test only
SRE	Full	Full	Full	No	No	Full
Monitor	Read	Read	Diagnostic	No	No	Metrics
Mentor	Read	Read	No	Full	No	Research

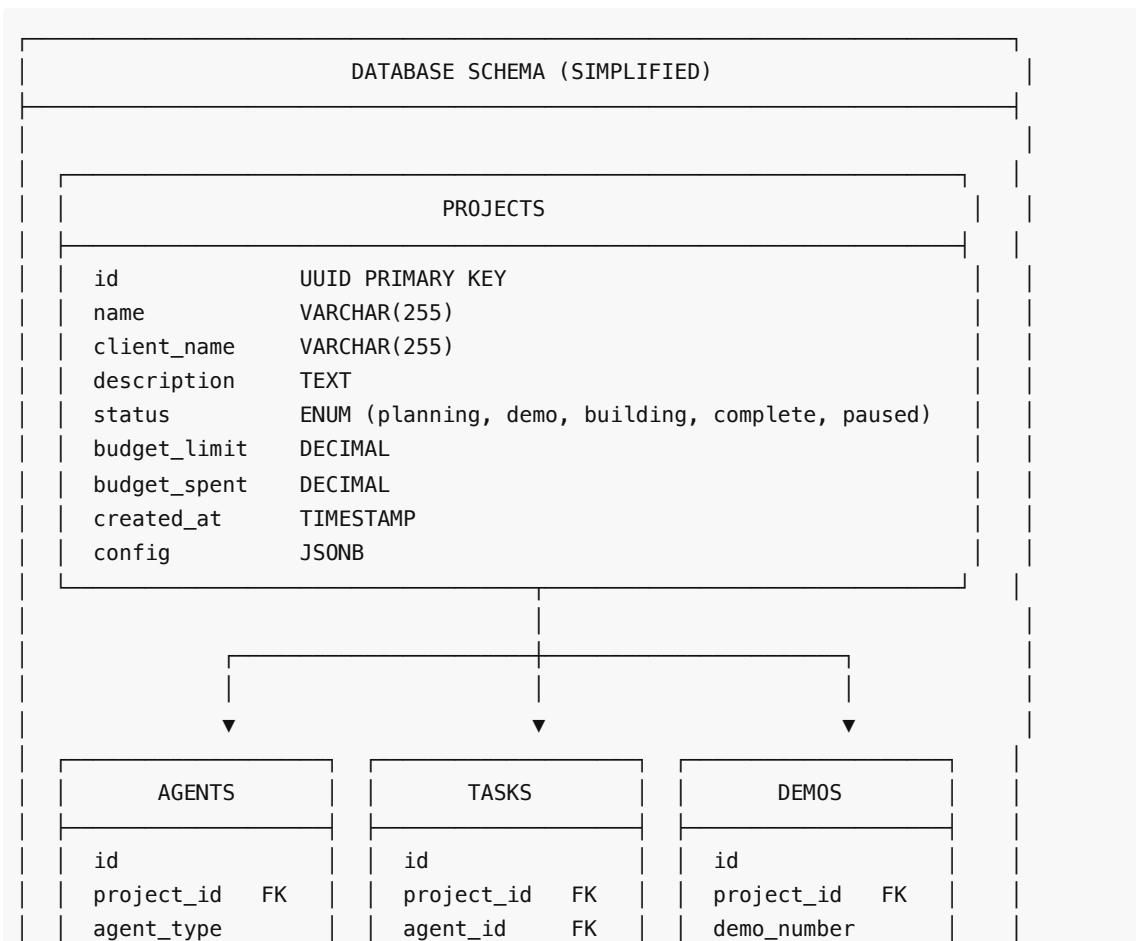
## Agent Lifecycle





## 6. Data Architecture

### Database Schema Overview



status	title	status
pid	status	preview_url
working_dir	priority	feedback
tokens_used	result	created_at
last_heartbeat	created_at	approved_at

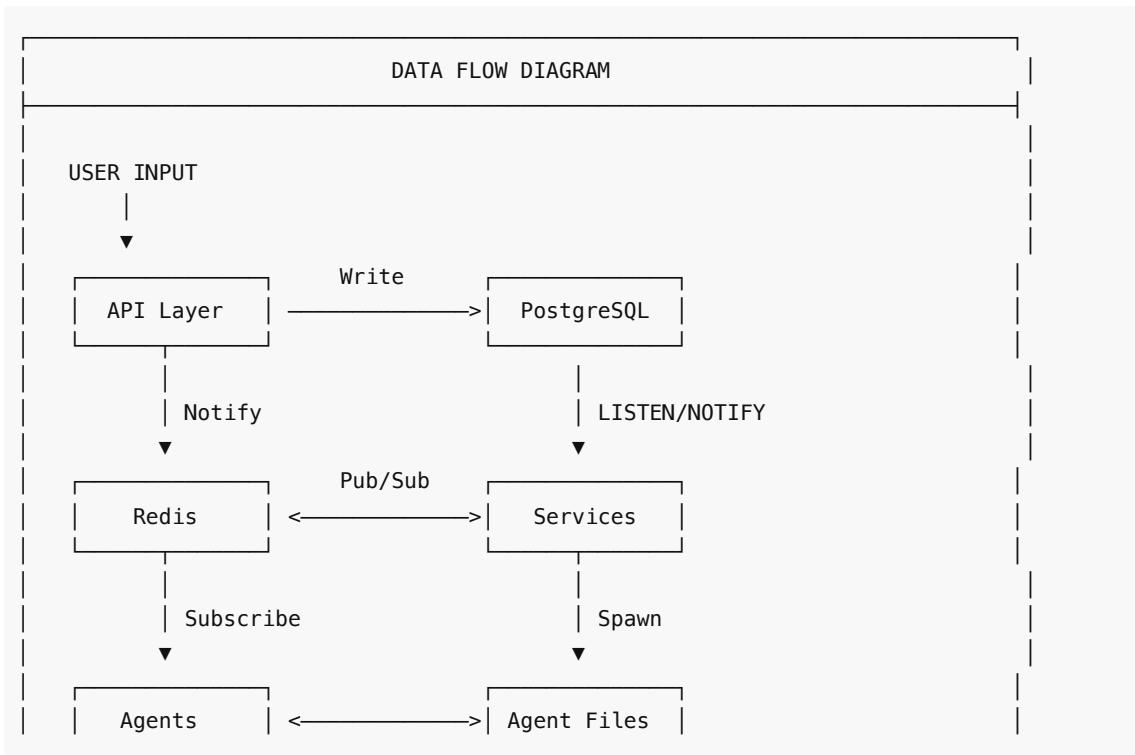
▼

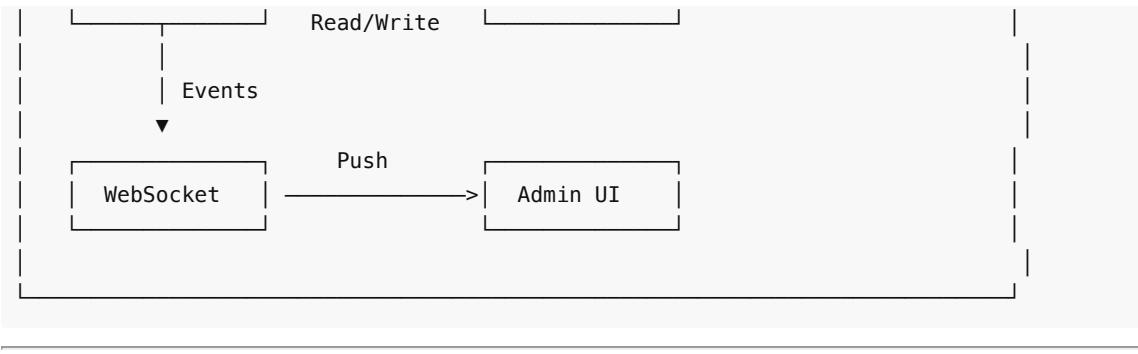
CHECKPOINTS	MESSAGES	PROMPT VERSIONS
id	id	id
agent_id FK	project_id FK	agent_type
state JSONB	from_agent FK	version
file_snapshot	to_agent FK	prompt_text
resume_instr	message_type	status
created_at	payload JSONB	success_rate

AUDIT_LOGS				
id	project_id	agent_id	action	details
timestamp	(nullable)	(nullable)	VARCHAR	JSONB

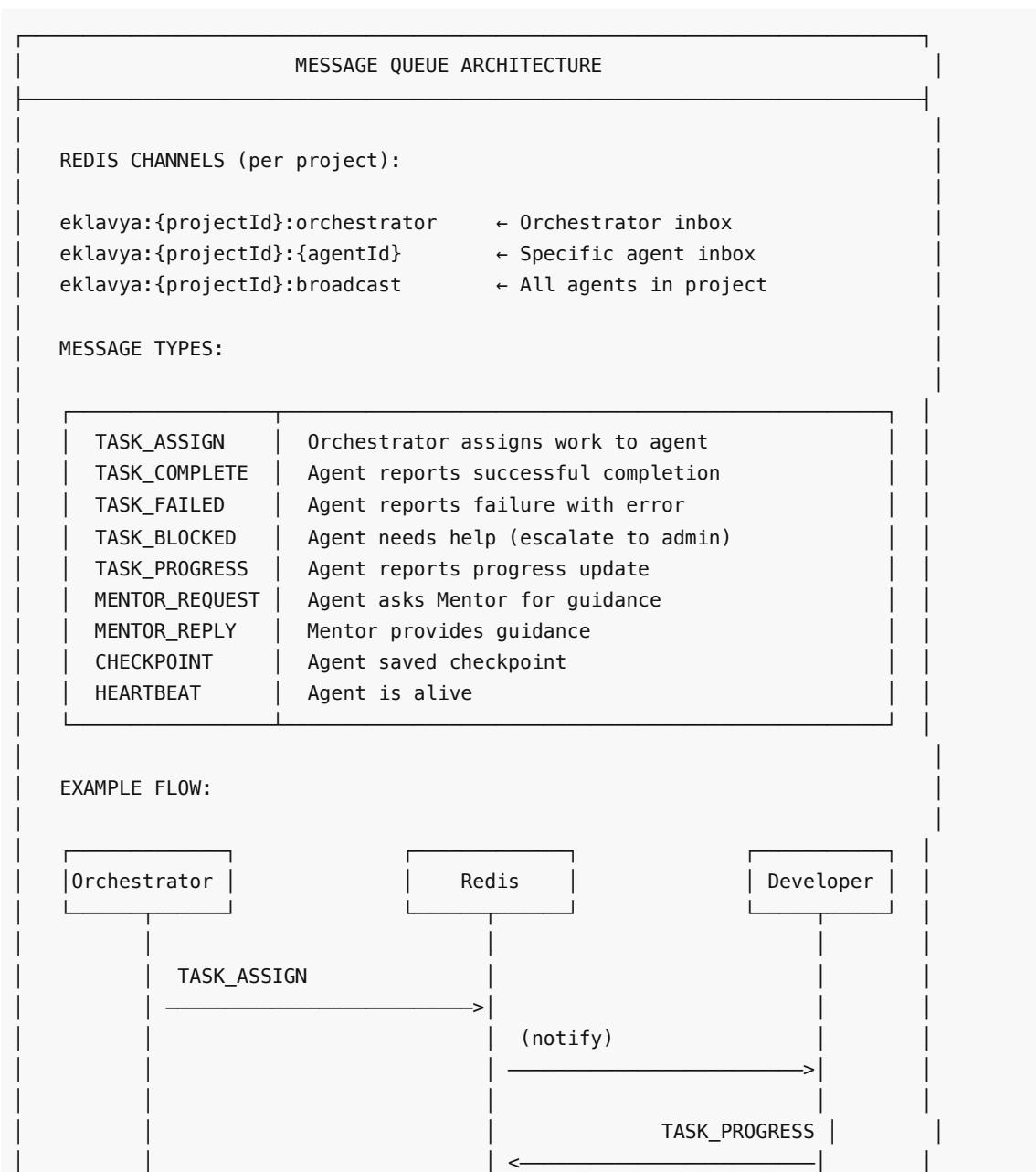
## Data Flow

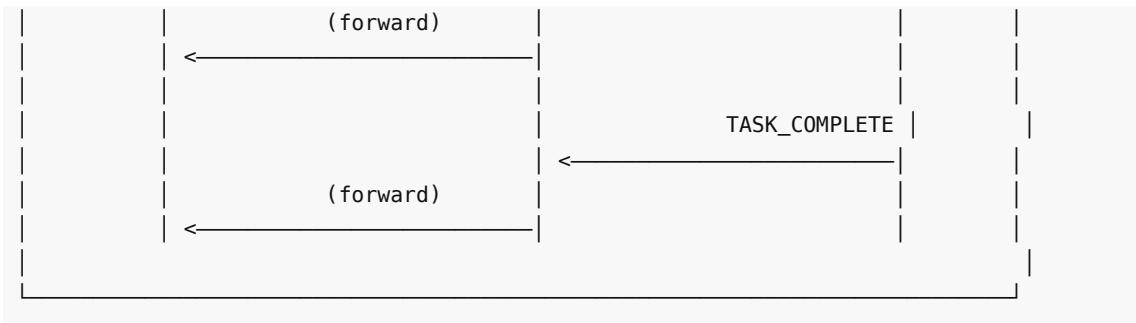




## 7. Communication Patterns

### Inter-Agent Messaging





## WebSocket Events (UI Updates)

WEBSOCKET EVENTS

Events pushed to Admin Portal in real-time:

Event Type	Payload
project:created	{ projectId, name, clientName }
project:status	{ projectId, status, progress }
agent:spawned	{ projectId, agentId, agentType }
agent:working	{ agentId, taskId, description }
agent:completed	{ agentId, taskId, result }
agent:blocked	{ agentId, reason, needsAdmin }
demo:ready	{ projectId, demoNumber, previewUrl }
demo:approved	{ projectId, demoNumber, decision }
budget:warning	{ projectId, spent, limit, percent }
budget:exceeded	{ projectId, spent, limit }
notification	{ level, title, message, actions }

## 8. Security Model

### Security Layers

SECURITY ARCHITECTURE

LAYER 1: AUTHENTICATION

- Admin authentication (session-based)
- API key authentication (for external integrations)
- No client direct access (admin controls everything)

#### LAYER 2: PROJECT ISOLATION

- Each project runs in isolated Docker container
- Separate file system per project
- Network isolation (no cross-project communication)
- Separate database schemas per project (optional)

#### LAYER 3: AGENT SANDBOXING

- Claude Code's built-in sandboxing
- Per-agent tool permissions (see matrix in Section 5)
- File system access restricted to project directory
- Network access limited by agent type

#### LAYER 4: SECRETS MANAGEMENT

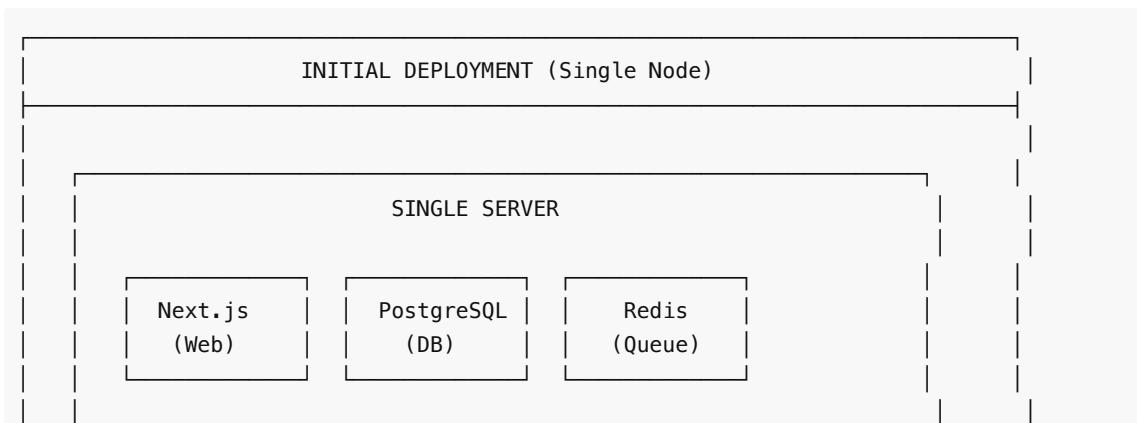
- Environment variables for API keys
- .env files excluded from git
- Secrets never logged or exposed in UI
- Per-project secrets isolation

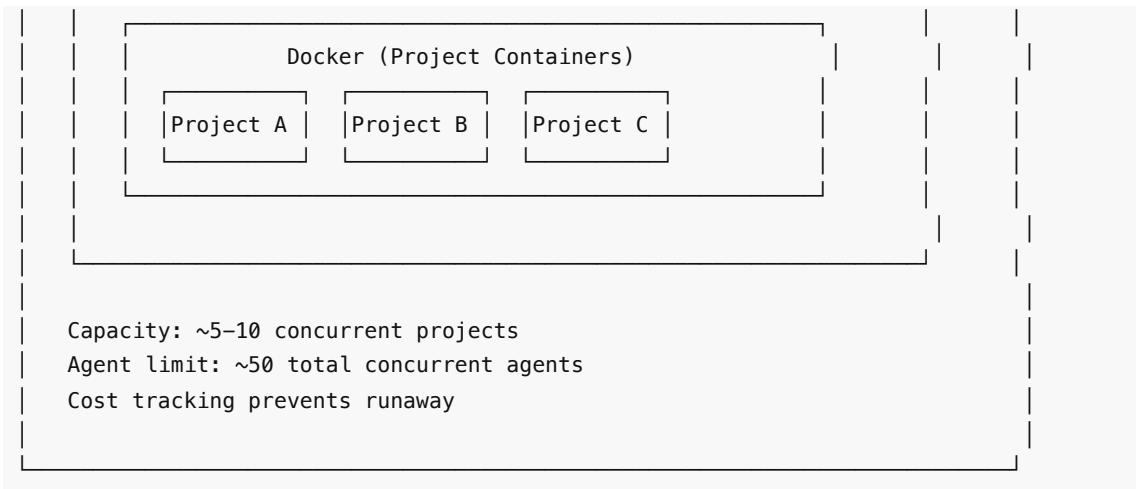
#### LAYER 5: AUDIT & COMPLIANCE

- All agent actions logged to audit\_logs table
- Immutable audit trail
- Cost tracking per action
- Checkpoint history for recovery

## 9. Scalability Considerations

### Current Design (Single Instance)





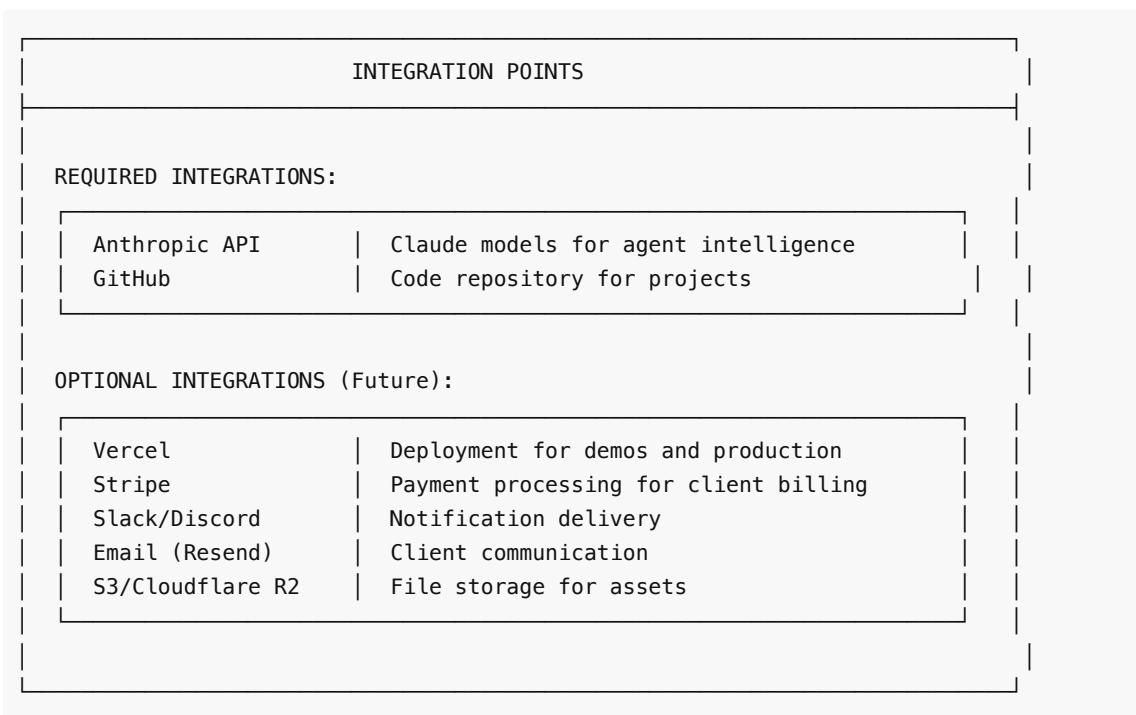
## Future Scaling (Not in v1.0)

The architecture is designed to scale horizontally when needed:

- PostgreSQL → Managed service (RDS, Supabase)
- Redis → Managed service (ElastiCache, Upstash)
- Projects → Kubernetes pods
- Load balancing → Multiple web instances

## 10. Integration Points

### External Services



## Summary

This architecture provides:

- **Simplicity:** Chat-first interface hides all complexity
- **Autonomy:** Agents work while you sleep
- **Control:** Admin approval gates at key points
- **Isolation:** Each project fully sandboxed
- **Learning:** System improves over time
- **Cost Awareness:** Hard budget limits prevent surprises

The initial implementation focuses on a single-node deployment that can handle 5-10 concurrent projects.

The architecture supports horizontal scaling when business demands it.

---

*Document generated for pre-implementation review. Subject to refinement during development.*