Eklavya Platform Documentation

Autonomous Agent Orchestration for Software Development

Version: 1.0.0 **Date**: January 21, 2026 **Author**: Ganesh Pandey

---

Table of Contents

---

1. Executive Summary

What is Eklavya?

Eklavya is an **autonomous agent orchestration platform** designed to build software projects with minimal human intervention. It uses **Reinforcement Learning** to continuously improve agent prompts and a sophisticated **multi-agent architecture** where specialized AI agents collaborate to design, develop, test, and deploy software.

Business Model

```
┌────────────────────────────────────────────────────────────────
│                    EKLAVYA WORKFLOW
│
├────────────────────────────────────────────────────────────────
│
│
│  Client Request → Admin Reviews → Eklavya Builds → Demo Ready
```

```
│
│       ↓               ↓               ↓               ↓
│
│   "I need an      Approves      Agents work      Admin shows
│
│    e-commerce     architecture   autonomously    to client
│
│    platform"      and plan       24/7
│
│
│
└──────────────────────────────────────────────────────────────────
```

Current Readiness

Aspect	Status	Readiness
Core Architecture	Complete	90%
Agent System	Complete	95%
Learning System	Complete	95%
Workflow Integration	Partial	35%
End-to-End Autonomy	Not Ready	30%
Overall	**Beta**	**~70%**

Bottom Line: Core components work excellently. Missing the "glue" that connects them into a seamless autonomous workflow.

---

2. Project Vision

The Problem

Traditional software development faces challenges: - **Expensive**: Senior developers cost $150-300/hour - **Slow**: Projects take weeks to months - **Inconsistent**: Quality varies by developer - **Not 24/7**: Human developers need rest

The Solution

Eklavya provides: - **Autonomous Agents**: 10 specialized AI agents working 24/7 - **Self-Improving**: RL system evolves prompts based on outcomes - **Demo-First**: Quick demos validate ideas before full investment - **Cost-Controlled**: Hard budget limits prevent runaway spending - **Checkpointed**: Every state saved for failure recovery

Key Principles

1. **Autonomy After Approval** - Human approves once, agents work independently

2. **Demo Before Build** - Validate with interactive demos before full implementation
3. **Everything Logged** - Complete audit trail for learning and debugging
4. **Fail Gracefully** - Checkpoints enable recovery from any failure
5. **Cost Aware** - Hard limits prevent budget overruns
6. **Self-Improving** - RL evolves agent behaviors based on outcomes

---

3. Architecture Overview

High-Level Architecture

```
 ┌─────────────────────────────────────────────────────────────────
 │                         EKLAVYA
PLATFORM                               │
 ├─────────────────────────────────────────────────────────────────
 │
 │
 │   ┌─────────────┐     ┌─────────────┐
 ┌──────────────┐               │
 │   │    CLI      │     │  Dashboard  │    │    API
 │              │
 │   │  (eklavya)  │     │   (Next.js) │    │  (REST)
 │              │
 │   └──────┬──────┘     └──────┬──────┘
 └──────┬───────┘
 │      ┬      │                 │      │
 │                      │
 │
 └───────────────────┼───────────────────┘                    │
 │
 │                                         │
 │
 ┌──────────────────────────┴─────────────────────────┐
 │
 │  │                 ORCHESTRATION LAYER
 │              │
 │  │  ┌────────────────┐  ┌────────────────┐
 │              │
 │  │  │  Orchestrator │  │ Workflow Eng. │ (planned)
 │              │
 │  │  └────────────────┘  └────────────────┘
 │              │
 │
 └──────────────────────────────────────────────────────┘
 │
 │
 │                                         │
 │
 ┌──────────────────────────┴─────────────────────────┐
 │
 │  │                  AGENT LAYER
 │             │
 │  │
 │             │
 │  │  ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐
 │             │
 │  │  │Arch. │ │ Dev  │ │Test  │ │ QA   │ │ PM   │
 │             │
 │  │  └──────┘ └──────┘ └──────┘ └──────┘ └──────┘
 │             │
 │  │  ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐
 │             │
 │  │  │ UAT  │ │ SRE  │ │Monit.│ │Mentor│ │Orch. │
 │             │
 │  │  └──────┘ └──────┘ └──────┘ └──────┘ └──────┘
 │             │
 │
 └───────────────────────────────────────────────────┘
 │
 │
 │                                        │
 │
 ┌─────────────────────────┴─────────────────────────┐
 │
 │  │                CORE SERVICES
 │             │
 │  │  ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐
 │             │
 │  │  │ Message │ │Learning │ │Checkpoint│ │  Demo   │
 │             │
 │  │  │   Bus   │ │  (RL)   │ │ System  │ │ System  │
 │             │
 │  │  └─────────┘ └─────────┘ └─────────┘ └─────────┘
 │             │
 │
 └───────────────────────────────────────────────────┘
 │
 │
 │                                        │
 │
 ┌─────────────────────────┴─────────────────────────┐
 │
 │  │                DATA LAYER
 │             │
 │  │  ┌─────────────────┐    ┌─────────────────┐
 │             │
 │  │  │   PostgreSQL    │    │     Redis       │
 │             │
 │  │  │  (persistence)  │    │  (pub/sub)      │
 │             │
 │  │  └─────────────────┘    └─────────────────┘
 │             │
 │
 └───────────────────────────────────────────────────┘
 │
```

```
│
│
└──────────────────────────────────────────────────────────────────
```

Agent Architecture

Each agent is a **Claude Code instance** with: - Specialized system prompt loaded via CLAUDE.md - Scoped tool permissions - Message queue access for coordination - Checkpoint capability for state recovery

```
┌─────────────────────────────────────────┐
│            AGENT INSTANCE               │
├─────────────────────────────────────────┤
│                                         │
│  ┌─────────────────────────────────┐    │
│  │      System Prompt (CLAUDE.md)  │    │
│  │  - Role definition              │    │
│  │  - Tool permissions             │    │
│  │  - Behavioral guidelines        │    │
│  └─────────────────────────────────┘    │
│                  │                      │
│  ┌───────────────┴───────────────┐      │
│  │       Claude Code Process     │      │
│  │  - File system access         │      │
│  │  - Git operations             │      │
│  │  - Terminal commands          │      │
│  │  - Browser automation (QA)    │      │
│  └───────────────────────────────┘      │
│                  │                      │
│  ┌───────────────┴───────────────┐      │
│  │       Message Bus Client      │      │
│  │  - Receive task assignments   │      │
│  │  - Report completion/failure  │      │
│  │  - Request help from Mentor   │      │
│  └───────────────────────────────┘      │
│                                         │
└─────────────────────────────────────────┘
```

---

4. Core Components

4.1 Orchestrator

Purpose: Coordinates parallel agent execution based on task dependencies.

Capabilities: - Dependency graph analysis with topological sort - Phase-based parallel execution - Agent spawning and lifecycle management - Outcome recording for RL feedback - Timeout handling (30 min default)

Status: ✅ Fully implemented, needs workflow integration

4.2 Agent Manager

Purpose: Spawns and manages individual agent processes.

Capabilities: - Single and parallel agent spawning - RL prompt selection via Thompson Sampling - CLAUDE.md injection into agent working directories - Heartbeat monitoring with timeout detection - Outcome recording with nuanced rewards

Status: ✅ Fully implemented

4.3 Message Bus

Purpose: Inter-agent communication via Redis pub/sub with PostgreSQL persistence.

Capabilities: - Real-time messaging between agents - Channel-based routing (project/agent specific) - Broadcast and point-to-point messaging - Message persistence for audit trail

Channels:

```
eklavya:{projectId}:orchestrator  - Orchestrator inbox
eklavya:{projectId}:{agentId}     - Specific agent inbox
eklavya:{projectId}:broadcast     - All project agents
```

Status: ✅ Fully implemented

4.4 Learning System (RL)

Purpose: Evolves agent prompts based on task outcomes using Reinforcement Learning.

Algorithm: Thompson Sampling with Beta distribution

Stratification: - **Experimental** (10% traffic): New, untested prompts - **Candidate** (30% traffic): Promising prompts under evaluation - **Production** (60% traffic): Proven, reliable prompts

Reward Scale:

```
+1.0  Perfect execution, exceeds expectations
+0.5  Good execution, meets requirements
 0.0  Acceptable, some minor issues
-0.5  Poor execution, significant issues
-1.0  Failed execution, critical bugs
```

Status: ✅ Fully implemented

4.5 Checkpoint System

Purpose: State persistence for failure recovery.

Triggers: - Every 15 minutes (automatic) - After task completion - Before risky operations

Captures: - Agent state (current task, progress, working memory) - File state (modified files, git status) - Conversation state (compressed history) - Recovery instructions

Status: ✅ Fully implemented

4.6 Demo System

Purpose: Manage demo lifecycle from creation to client approval.

Demo Types:

Type	Purpose	Time	Cost
Demo$_0$ (Wow)	Beautiful UI, clickable prototype	20-30 min	$8-15
Demo$_1$ (Trust)	Core feature working, real-ish data	30-45 min	$15-25
Milestone	Progress checkpoint during build	Varies	Varies
Final	Complete product	Varies	Varies

Status: ✅ Fully implemented

---

5. Technology Stack

Component	Technology	Version
Runtime	Node.js	20+
Language	TypeScript	5.x
Database	PostgreSQL	16
Cache/Queue	Redis	7
AI Provider	Anthropic Claude	claude-sonnet-4-20250514
Web Framework	Next.js	14
Testing	Vitest, Playwright	Latest

---

6. What's Implemented

Fully Functional (90%+)

Component	Description	Readiness
Orchestrator	Parallel task execution with dependencies	95%
Agent Manager	Agent spawning with RL prompt selection	95%
Message Bus	Redis pub/sub + PostgreSQL persistence	100%
Learning System	Thompson Sampling RL	95%
Checkpoint System	State persistence and recovery	95%
Tester Agent	Bug reporting with RL feedback	95%

Component	Description	Readiness
Demo Service	Demo lifecycle management	95%
Approval Workflow	Demo approval with feedback	90%

Partially Implemented (50-85%)

Component	Description	Readiness
API Server	50+ REST endpoints	75%
CLI	8 commands implemented	70%
Frontend Dashboard	Project/agent views	50%
Architect Agent	Quality analysis	85%
Coordination	File locks, conflict detection	75%

Scaffolding Only (< 50%)

Component	Description	Readiness
Workflow Engine	Phase transitions	30%
QA Agent	End-to-end testing	40%
PM Agent	Requirements management	40%
UAT Agent	User acceptance testing	40%
Monitor Agent	Health monitoring	40%
Mentor Agent	Guidance system	40%
Self-Build Manager	Autonomous project building	40%

---

7. Demo Implementations

Eight progressive demos have been implemented, each adding capabilities:

Demo1: Agent Lifecycle Management

Focus: Basic agent spawning and monitoring - Agent creation and termination - Status tracking - Health monitoring

Demo2: Messaging System

Focus: Inter-agent communication - Message bus implementation - Channel-based routing - Message persistence

Demo3: Task Execution

Focus: Task assignment and tracking - Task queues - Assignment logic - Completion tracking

Demo4: Lifecycle Management

Focus: Advanced agent lifecycle - Heartbeat monitoring - Automatic recovery - Resource cleanup

Demo5: Coordination

Focus: Multi-agent coordination - File locking - Conflict detection - Work distribution

Demo6: Real-Time Portal

Focus: Monitoring and notifications - Smart notifications (4 levels) - Activity stream - Progress tracking - Notification settings

Demo7: Demo System

Focus: Demo management - Demo creation and versioning - Approval workflow - Client feedback - Verification system

Demo8: Self-Build

Focus: Autonomous building capability - Execution plan generation - Phase management - Sample project templates - Build orchestration

Test Results

Demo	Tests	Pass Rate	Grade
Demo6	21	100%	B (83%)
Demo7	25	100%	A (100%)
Demo8	33	100%	A (100%)
Total	**79**	**100%**	**A-**

---

8. CLI Reference

Installation

```
# Development mode
npx tsx src/cli/index.ts --help
```

```
# After build
npm run build
npm link
eklavya --help
```

Commands

`eklavya new <name>`

Create a new project.

```
eklavya new my-app -d "E-commerce platform" -b 150

Options:
  -d, --description <text>  Project description
  -b, --budget <amount>     Budget limit in USD (default: 100)
  -t, --type <type>         Project type: new or existing
  -p, --path <path>         Path to existing codebase
```

`eklavya list`

List all projects.

```
eklavya list --status active --limit 20

Options:
  -s, --status <status>  Filter by status
  -l, --limit <n>        Maximum projects to show
  -a, --all              Include completed projects
```

`eklavya status [project-id]`

Check project status.

```
eklavya status                  # All projects overview
eklavya status <id>             # Specific project
eklavya status <id> -v          # Verbose with agents/tasks

Options:
  -v, --verbose    Show all details
  -a, --agents     Show agent details
  -t, --tasks      Show task details
```

`eklavya logs <project-id>`

Stream project logs.

```
eklavya logs <id> --follow --agent developer

Options:
  -f, --follow         Real-time log streaming
  -a, --agent <type>   Filter by agent type
  -l, --level <level>  Minimum log level
  -n, --limit <n>      Number of entries
```

`eklavya demo <project-id>`

View and manage demos.

```
eklavya demo <id> --open

Options:
  -o, --open             Open demo in browser
  -s, --screenshots      Show screenshot paths
  -n, --number <n>       Show specific demo
```

`eklavya approve <project-id>`

Approve demos and decisions.

```
eklavya approve <id> --demo 0 --feedback "Looks great!"

Options:
  -d, --demo <number>     Specific demo to review
  -f, --feedback <text>   Provide feedback
  -s, --skip              Skip to full build
  -r, --reject            Request changes
```

`eklavya stop <project-id>`

Stop projects or agents.

```
eklavya stop <id> --all --force

Options:
  -a, --agent <id>    Stop specific agent
  -f, --force         Force stop
      --all           Stop all agents
```

`eklavya config`

Manage configuration.

```
eklavya config                                 # Show all
eklavya config get database.host               # Get value
eklavya config set defaults.maxBudget 200      # Set value
```

---

9. Current Limitations

9.1 Not End-to-End Autonomous

Current State: - Project can be created via CLI - Agents CAN be spawned (code exists) - BUT no automatic trigger connects them

What's Missing:

```
Project Created
       ↓
   (GAP) → No automatic architect phase
       ↓
   (GAP) → No automatic task generation
       ↓
   (GAP) → No automatic agent spawning
       ↓
   Nothing happens without manual intervention
```

9.2 No Workflow Engine

The orchestrator works but isn't connected to a workflow that: - Triggers architect on project creation - Generates tasks from architect output - Spawns agents via orchestrator - Handles approval gates - Progresses through phases automatically

9.3 Security Not Implemented

Critical Security Gaps: - No authentication on API endpoints - No authorization/ACL - CORS allows all origins - No rate limiting - No request size limits

Impact: Cannot safely deploy to production

9.4 Incomplete Agent Types

Agent	Status
Orchestrator	✅ Complete
Architect	✅ Complete
Developer	✅ Complete
Tester	✅ Complete
QA	⚠️ Placeholder
PM	⚠️ Placeholder
UAT	⚠️ Placeholder
SRE	⚠️ Partial
Monitor	⚠️ Placeholder
Mentor	⚠️ Placeholder

9.5 Frontend Incomplete

Missing pages: - Project detail view - Agent detail view - Orchestrator visualization - Learning system analytics - Settings page

---

10. Roadmap & Improvements

Phase 1: Critical Integration (1-2 weeks)

10.1 Workflow Engine

Priority: CRITICAL

Create `src/core/workflow/engine.ts`:

```
class WorkflowEngine {
  async executeProjectBuild(projectId: string) {
    // 1. Run architect phase
    await this.runArchitectPhase(projectId);

    // 2. Generate tasks from architecture
    const tasks = await this.generateTasks(projectId);

    // 3. Create and execute plan
    const plan = this.orchestrator.createExecutionPlan(tasks);
    await this.orchestrator.executePlan(plan);

    // 4. Build demo
    await this.buildDemo(projectId, 'wow');

    // 5. Wait for approval
    await this.waitForApproval(projectId);

    // 6. Continue or revise based on decision
  }
}
```

10.2 CLI Build Command

Priority: CRITICAL

Add `eklavya build <project-id>` command that: - Triggers workflow engine - Shows real-time progress - Handles errors gracefully

10.3 Auto-Trigger on Project Create

Priority: HIGH

When `eklavya new` is called: - Automatically spawn architect agent - Begin design phase - Notify admin when ready for review

Phase 2: Security Hardening (1 week)

10.4 Authentication

- JWT-based auth with access/refresh tokens
- Token validation middleware
- Secure token storage

10.5 Authorization

- Role-based access control (admin/user)
- Project-level permissions
- API endpoint protection

10.6 Security Headers

- CORS whitelist configuration
- Rate limiting (sliding window)
- Request size limits
- CSRF protection

Phase 3: Complete Agent Types (2 weeks)

10.7 QA Agent

- End-to-end test execution
- Visual regression testing
- Performance testing

10.8 Mentor Agent

- Knowledge base queries
- Guidance for blocked agents
- Best practice suggestions

10.9 Monitor Agent

- Health check automation
- Alerting on failures
- Resource monitoring

Phase 4: Production Readiness (2 weeks)

10.10 Cost Tracking

- Token usage calculation

- API call cost tracking
- Budget enforcement
- Cost notifications

10.11 Enhanced Learning

- Richer outcome metrics
- A/B testing for prompts
- Performance dashboards

10.12 Frontend Completion

- Project detail pages
- Real-time updates via WebSocket
- Learning system analytics

Phase 5: Advanced Features (4+ weeks)

10.13 Multi-Project Management

- Dashboard showing all projects
- Resource allocation across projects
- Priority-based scheduling

10.14 Client Portal

- Read-only project views for clients
- Demo access links
- Feedback submission

10.15 Plugin System

- Custom agent types
- Integration hooks
- External tool connectors

---

11. Getting Started

Prerequisites

```
# Required
- Node.js 20+
- PostgreSQL 16
- Redis 7
- Claude Code CLI (claude)
```

```
# Optional
- Docker (for containerized deployment)
```

Installation

```
# Clone repository
git clone https://github.com/ganeshpandeyvns/eklavya.git
cd eklavya

# Install dependencies
npm install

# Set up environment
cp .env.example .env
# Edit .env with your database credentials

# Run migrations
npm run db:migrate

# Start development
npm run dev
```

Environment Variables

```
# Database
DB_HOST=localhost
DB_PORT=5432
DB_NAME=eklavya
DB_USER=eklavya
DB_PASSWORD=your_password

# Redis
REDIS_URL=redis://localhost:6379

# API
API_PORT=4000

# Authentication (when implemented)
JWT_SECRET=your_secret
JWT_REFRESH_SECRET=your_refresh_secret

# Development
AUTH_DISABLED=true  # Disable auth for testing
```

Running Tests

```
# Run all demo tests
npm run demo:6
npm run demo:7
npm run demo:8
```

```
# Run unit tests
npm test

# Run E2E tests
npm run test:e2e
```

Using the CLI

```
# Set database password
export DB_PASSWORD=your_password

# Create a project
npx tsx src/cli/index.ts new "My App" -d "Description"

# Check status
npx tsx src/cli/index.ts status

# View logs
npx tsx src/cli/index.ts logs <project-id>
```

Appendix A: Database Schema Overview

Core Tables

Table	Purpose
projects	Project definitions and status
agents	Agent instances and metrics
tasks	Task definitions and assignments
messages	Inter-agent communication
checkpoints	Agent state snapshots
prompts	Agent prompt versions
rl_outcomes	RL training data
demos	Demo lifecycle
approval_requests	Approval workflow
feedback	Client feedback

Key Relationships

```
projects (1) ──── (N) agents
projects (1) ──── (N) tasks
projects (1) ──── (N) demos
agents (1) ──── (N) tasks (assigned)
agents (1) ──── (N) checkpoints
```

```
demos (1) ──── (N) approval_requests
demos (1) ──── (N) feedback
```

Appendix B: Agent Prompt Structure

Each agent receives a CLAUDE.md file with:

```
# Agent: [Type]

## Role
[Description of agent's purpose]

## Responsibilities
- [Responsibility 1]
- [Responsibility 2]

## Tools Available
- [Tool 1]: [Description]
- [Tool 2]: [Description]

## Communication
- Channel: eklavya:{projectId}:{agentId}
- Message types: TASK_ASSIGN, TASK_COMPLETE, etc.

## Guidelines
- [Guideline 1]
- [Guideline 2]

## Current Context
- Project: {projectName}
- Task: {currentTask}
```

Appendix C: Message Types

Type	Sender	Receiver	Purpose
TASK_ASSIGN	Orchestrator	Agent	Assign new task
TASK_COMPLETE	Agent	Orchestrator	Report success
TASK_FAILED	Agent	Orchestrator	Report failure
TASK_BLOCKED	Agent	Mentor	Request help
MENTOR_SUGGESTION	Mentor	Agent	Provide guidance
STATUS_UPDATE	Agent	Broadcast	Progress update

Document Information

Created: January 21, 2026 **Last Updated**: January 21, 2026 **Repository**: https://github.com/ganeshpandeyvns/eklavya **License**: ISC

This document is auto-generated and should be updated as the platform evolves.