Agent Runtime Architecture Review

doany.ai — Platform Team

April 2026 · Engineering Leadership Review

Agenda

- **System Overview** — What the agent runtime does
- **End-to-End Data Flow** — Full path from user input to rendered result
- **Orchestrator** — ExecutionGraph, DAG scheduling, retry logic
- **Sandbox Runtime** — Isolation model with Firecracker / Docker
- **Streaming Pipeline** *(new)* — Ring buffer, backpressure, SSE protocol
- **Fault Tolerance** *(new)* — Circuit breakers, checkpoint/resume, DLQ
- **Streaming + Checkpoints** — How the new systems interact
- **Infrastructure & Observability**

System Overview

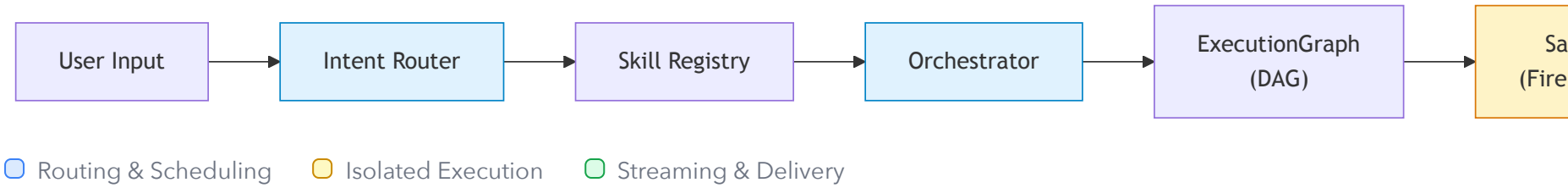
What the Agent Runtime Does

The runtime is the execution engine behind every skill invocation on doany.ai.

- Receives **user intents** (natural language or structured)
- **Selects and composes** skills via the Skill Registry
- **Orchestrates** execution as a DAG of sandboxed steps
- **Streams** results back to the frontend in real time
- **Recovers** from failures without losing progress

Key invariant: Every skill step runs in an isolated sandbox. The runtime never executes user-triggered code in the host process.

End-to-End Data Flow



Orchestrator

Orchestrator Deep-Dive

Receives a `SkillPlan`, builds an `ExecutionGraph` (DAG)

- **Parallel fan-out** for independent skill steps
- **Typed channels** pass data between steps
- **Retry**: exponential backoff + jitter, max 3 attempts
- **Timeouts**: 120s per step, 600s per graph
- **Checkpoints**: graph state → Redis every 10s

```
interface SkillPlan {
  intentId: string;
  skills: SkillStep[];
  constraints: ExecutionConstraints;
}

interface SkillStep {
  skillId: string;
  inputs: Record<string, unknown>;
  dependsOn: string[];
  timeout: number;
  retryPolicy: RetryPolicy;
}

interface ExecutionGraph {
  id: string;
  steps: Map<string, SkillStep>;
  edges: [string, string][];
  status: 'pending' | 'running'
    | 'completed' | 'failed';
  checkpointKey?: string;
}
```

Sandbox Runtime

Firecracker microVMs in production, Docker containers in dev/staging.

Resource Limits

- **2 vCPU** per sandbox
- **4 GB RAM**
- **10 GB ephemeral disk**
- Network **deny-by-default**
- Skills must declare required domains

Filesystem Model

- Root FS: **read-only**
- `/workspace` : read-write (skill working dir)
- `/output` : read-write (results written here)
- Mounts destroyed after step completes

Why Firecracker? Sub-second boot time (~125ms), strong isolation via KVM, smaller attack surface than containers. Docker is used in dev for faster iteration.

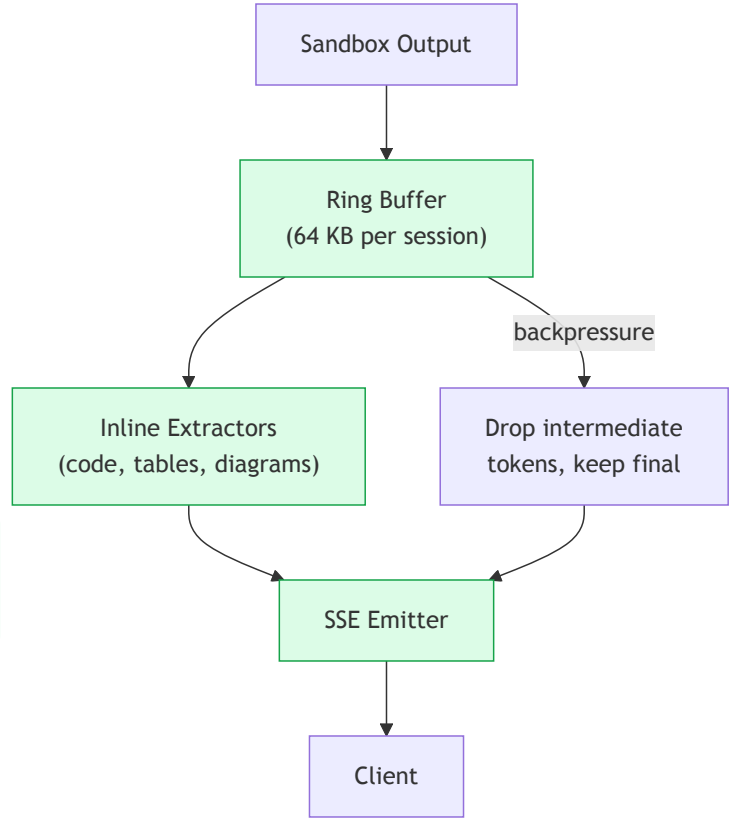
Streaming Pipeline

Shipped 2026-04-09 — replaces the batch-then-send model

Streaming Architecture

- **Token-level streaming** replaces old batch model
- **Ring buffer** per active session (64 KB capacity)
- **Backpressure**: if client falls behind, buffer drops intermediate tokens but preserves final state
- **Inline extractors** detect code blocks, tables, and diagrams as they stream

Impact: Time-to-first-token dropped from ~3s (batch) to ~180ms (streaming).



SSE Event Protocol

All events use `text/event-stream` with a monotonic sequence number for ordering.

Event	Payload	When
`token`	`{ text: string }`	Each token from the skill
`artifact`	`{ type, url, meta }`	File/image/code generated
`status`	`{ stepId, state }`	Step state transitions
`error`	`{ code, message, stepId }`	Recoverable errors
`done`	`{ resultId }`	Skill execution complete

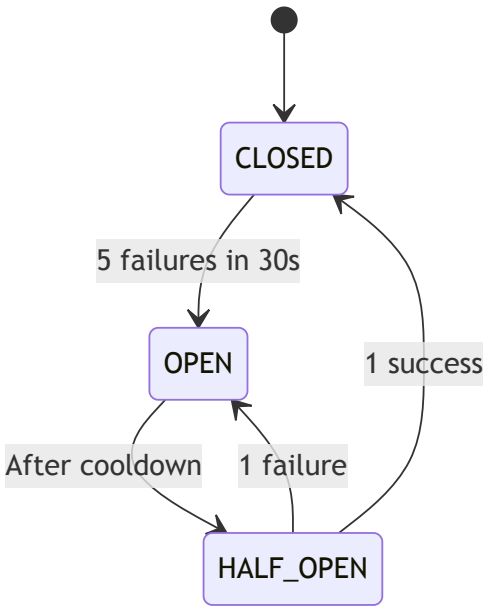
```
interface StreamEvent {
  type: 'token' | 'artifact' | 'status' | 'error' | 'done';
  sessionId: string;
  stepId: string;
  payload: unknown;
```

Fault Tolerance

Circuit breakers + checkpoint/resume — live since 2026-04-03

Circuit Breaker

One breaker per external dependency: LLM provider, storage, search.



- **CLOSED** – Normal operation, requests pass through
- **OPEN** – Failing dependency, reject fast (no waiting for timeouts)
- **HALF-OPEN** – Probe with a single request to test recovery

Checkpoint / Resume & DLQ

Checkpoint/Resume

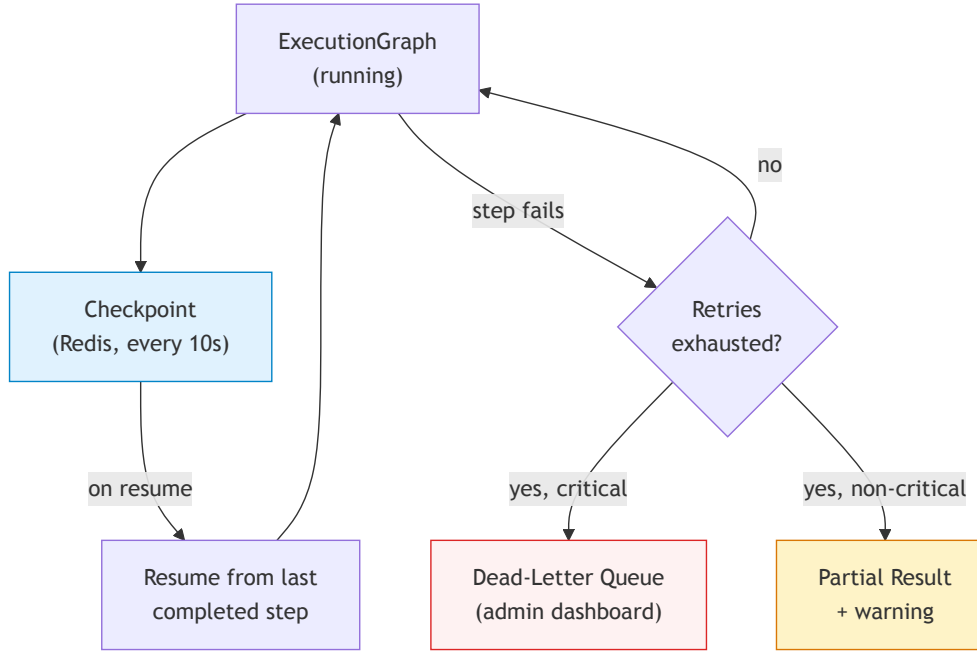
- Graph state serialized to **Redis every 10s**
- On failure, resume from **last completed step**
- Eliminates re-running expensive steps (LLM calls, data processing)

Dead-Letter Queue

- Permanently failed steps → **DLQ**
- Surfaced in **admin dashboard**
- Operators can inspect, retry, or discard

Graceful Degradation

- Non-critical skill failure → **partial result + warning**
- User sees what succeeded, not a blank error



How Streaming + Checkpoints Interact

Infrastructure

Runtime & Compute

- **Node.js 22** on Fly.io
- **4 regions**: iad, cdg, nrt, syd
- **Firecracker** microVMs (prod)
- **Docker** containers (dev/staging)

State & Storage

- **Redis Cluster** – checkpoints + sessions
- **Vercel Blob** – artifacts (images, files, code)
- **Supabase Postgres** – metadata

Observability

- **OpenTelemetry** → Grafana Cloud
- Traces, metrics, and logs unified
- Per-step trace spans for debugging

Deployment

- **GitHub Actions** CI/CD
- Canary rollout: 10% → 50% → 100%
- Rollback on error-rate spike

Questions?

Agent Runtime Architecture Review – doany.ai