Async Agents 101

Background Agents & Async Workflows

What We’ll Cover

1. **Why Async:** The multiplication effect
2. **Where to Trigger:** Cursor, GitHub, Slack, Linear, Codex
3. **Foundations for Trust:** Building on sync foundations
4. **Team Adoption**

Have you used background agents before? (yes/no + what for?)

Part 1: Why Async?

The Multiplication Effect

The Multiplication Effect

Async workflows unlock work that would otherwise never happen.

Every codebase has a backlog of small improvements that never get prioritized:

- Documentation updates
- Minor bug fixes
- Logging consistency
- Code cleanup

Humans deprioritize these because the context-switching cost is too high.

What’s one small fix you’ve been putting off that you’d love to delegate?

Work Continues Without You

The shift: from "is this worth my time?" to "is this worth kicking off?"

- Kick off a task from your phone before bed, wake up to a full implementation
- Start an agent before your 10am standup, review the PR over lunch
- Tag an issue during a meeting, merge the fix before end of day

The work happens in parallel with your life, not blocking it.

When is your best kickoff window? (before standup / lunch / after meetings / before bed / commute)

Small Wins Compound

Building confidence through momentum:

- Each small fix can be reviewed and merged quickly
- Builds your confidence in the agent’s capabilities
- Creates momentum that compounds into larger delegations
- Ten 50-line PRs compound into significant progress over a week

Part 2: Where to Trigger Async Agents

Cursor, GitHub, Slack, Linear, Codex

Three Kickoff Patterns

Pattern	Best For	Planning Overhead
Cursor Plan Mode	Complex features, architectural changes	High - but worth it
GitHub Mentions	Isolated fixes, PR feedback, test fixes	Low
Slack	Quick fixes, docs, small improvements	Minimal

The key insight: Match your kickoff method to the complexity of the task.

Cursor: Plan Mode to Background Agent

The most structured approach for complex features:

1. Enter Plan Mode in Cursor
2. Iterate on the plan until you’re satisfied
3. Review, ask clarifying questions, refine scope
4. Kick off the background agent with the detailed plan
5. Agent executes independently while you work on something else

Use this when you need to think through the implementation approach before letting the agent run.

Cursor: Automatic CI Fixing

Cursor's Cloud Agents automatically attempt to fix CI failures in PRs they create.

- Ignores failures that also fail on the base commit (pre-existing issues)
- Currently supports GitHub Actions
- Disable globally: Cursor Dashboard > Cloud Agents > My Settings
- Disable per-PR: comment `@cursor autofix off`

For your own PRs: Tag Cursor in a comment:

- `@cursor please fix the CI failures`
- `@cursor fix the lint check failure`

GitHub: @-Mentions on Issues and PRs

For small, well-defined changes:

- **On Issues:** `@cursor implement this feature` or `@codex fix this bug`
- **On Pull Requests:** `@codex please address the review feedback`
- **On failing CI:** `@cursor fix the lint errors`
- **Update context:** `@cursor fix this and update AGENTS.md with what you learned`

> When an agent misses something, tell it to fix the issue AND update AGENTS.md with what it learned. This improves future runs.

Works from your phone - respond to notifications while on the go.

Where would you most want to trigger async work? (Slack / Linear / GitHub / Cursor / Codex)

Slack: Devin’s Release Workflow

"Devin, please do a release" triggers a multi-step workflow:

1. Reviews QA results
2. Determines bug ownership using git history
3. Tags responsible engineers in Slack
4. Sends reminders if issues remain unaddressed

Why this works: The release process follows a predictable pattern but involves multiple steps. Human kicks it off with a single message and walks away.

Linear: AI-Native Command Center

Linear evolved into an AI-native command center:

- Assign Linear issues directly to coding agents
- Agents work in background sessions while you focus elsewhere
- Agents are first-class citizens: mentionable, assignable, with activity streams
- Workspace-level prompt guidance fed to all agents

Two approaches:

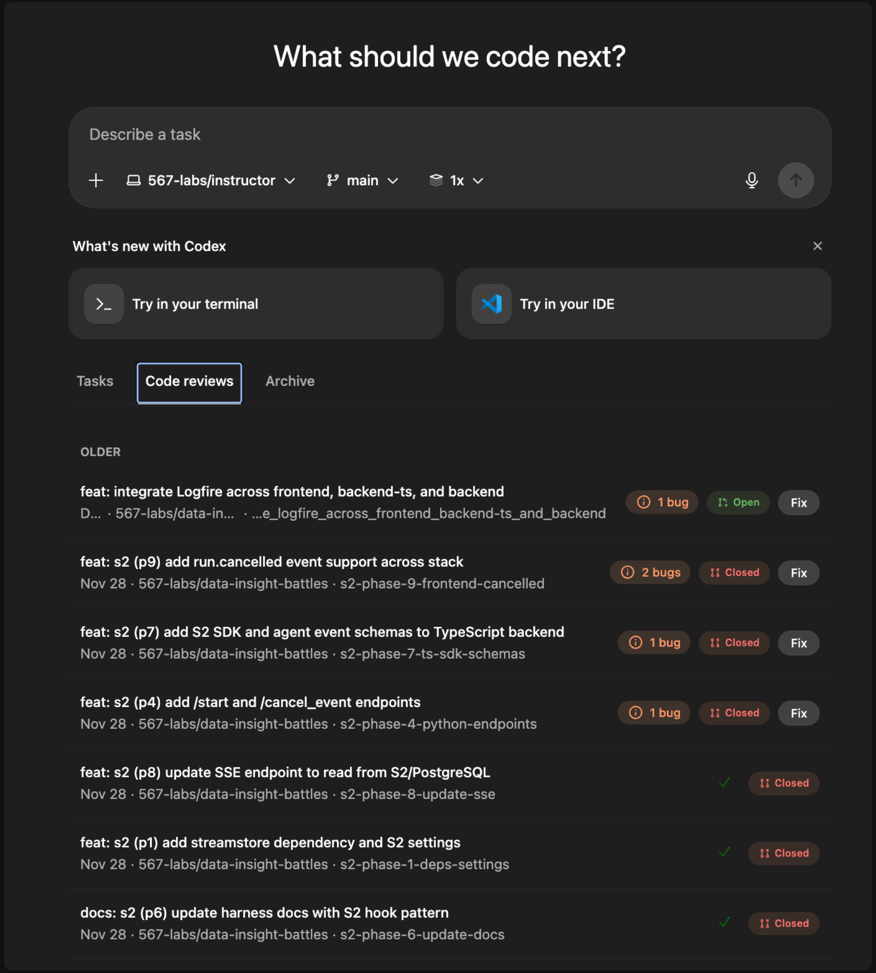
1. Linear as orchestrator - managing everything
2. Linear as source of truth - pull context into Cursor as needed

Codex: Multi-Interface Access

One agent, many interfaces:

- CLI / VS Code / Cursor extensions
- Browser (chatgpt.com/codex)
- GitHub mentions
- Slack integration
- **iOS app** - kick off from your phone

Same agent everywhere. Start on phone, check on laptop, review via Slack.


What should we code next?

Describe a task

What's new with Codex

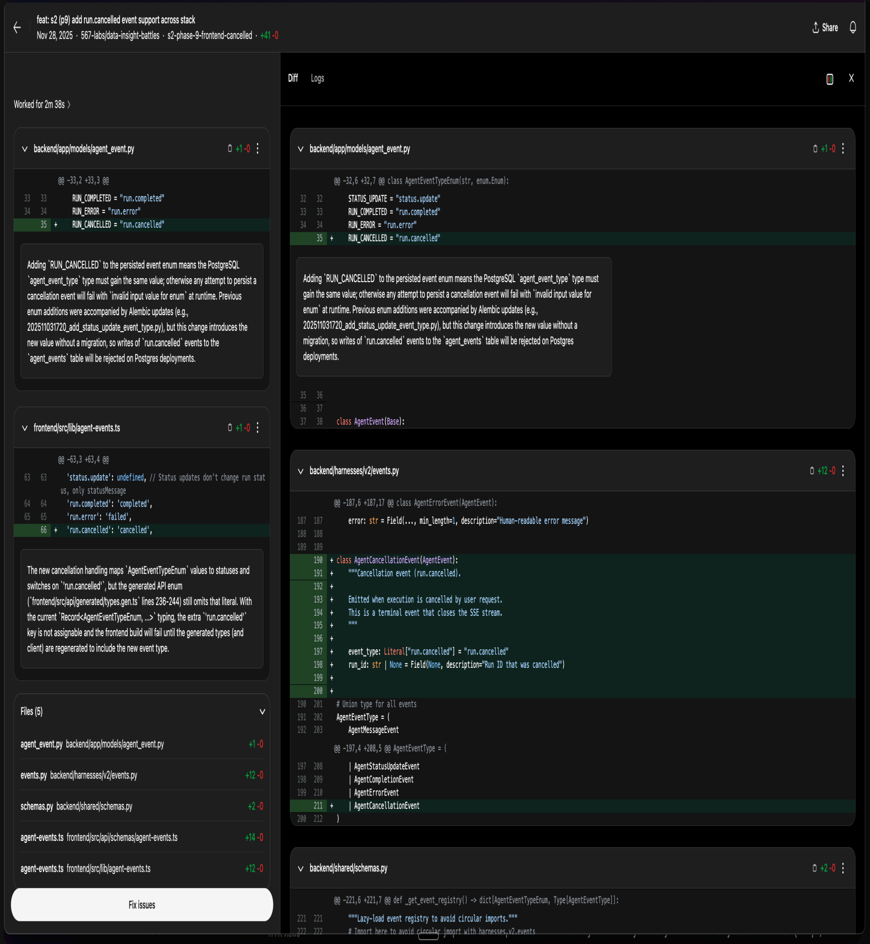
OLDER

feat: integrate Logfire across frontend, backend-ts, and backend D... · 567-labs/data-in... · ...e_logfire_across_frontend_backend-ts_and_backend	1 bug	Open	Fix
feat: s2 (p9) add run.cancelled event support across stack Nov 28 · 567-labs/data-insight-battles · s2-phase-9-frontend-cancelled	2 bugs	Closed	Fix
feat: s2 (p7) add S2 SDK and agent event schemas to TypeScript backend Nov 28 · 567-labs/data-insight-battles · s2-phase-7-ts-sdk-schemas	1 bug	Closed	Fix
feat: s2 (p4) add /start and /cancel_event endpoints Nov 28 · 567-labs/data-insight-battles · s2-phase-4-python-endpoints	1 bug	Closed	Fix
feat: s2 (p8) update SSE endpoint to read from S2/PostgreSQL Nov 28 · 567-labs/data-insight-battles · s2-phase-8-update-sse	✓	Closed	
feat: s2 (p1) add streamstore dependency and S2 settings Nov 28 · 567-labs/data-insight-battles · s2-phase-1-deps-settings	✓	Closed	
docs: s2 (p6) update harness docs with S2 hook pattern Nov 28 · 567-labs/data-insight-battles · s2-phase-6-update-docs	✓	Closed	

Codex: Code Reviews

One-click fixes from the code review tab:

- Codex analyzes PRs and identifies potential bugs
- Click "Fix issues" to launch an agent that addresses them
- Agent reads the diff, understands context, and creates fixes



feat: s2 (p9) add run.cancelled event support across stack
Nov 28, 2025 · 567-labs/data-insight-battles · s2-phase-9-frontend-cancelled · +41 -0

Worked for 2m 38s

backend/app/models/agent_event.py +1 -0

```
@@ -33,2 +33,3 @@
    RUN_COMPLETED = "run.completed"
    RUN_ERROR = "run.error"
+   RUN_CANCELLED = "run.cancelled"
```

Adding `RUN_CANCELLED` to the persisted event enum means the PostgreSQL `agent_event_type` type must gain the same value; otherwise any attempt to persist a cancellation event will fail with `invalid input value for enum` at runtime. Previous enum additions were accompanied by Alembic updates (e.g., 202511031720_add_status_update_event_type.py), but this change introduces the new value without a migration, so writes of `run.cancelled` events to the `agent_events` table will be rejected on Postgres deployments.

frontend/src/lib/agent-events.ts +1 -0

```
@@ -63,3 +63,4 @@
    'status.update': undefined, // Status updates don't change run stat
us, only statusMessage
    'run.completed': 'completed',
    'run.error': 'failed',
+   'run.cancelled': 'cancelled',
```

The new cancellation handling maps `AgentEventTypeEnum` values to statuses and switches on `'run.cancelled'`, but the generated API enum (`frontend/src/api/generated/types.gen.ts` lines 236-244) still omits that literal. With the current `Record<AgentEventTypeEnum, ...>` typing, the extra `'run.cancelled'` key is not assignable and the frontend build will fail until the generated types (and client) are regenerated to include the new event type.

Files (5)

agent_event.py	backend/app/models/agent_event.py	+1 -0
events.py	backend/harnesses/v2/events.py	+12 -0
schemas.py	backend/shared/schemas.py	+2 -0
agent-events.ts	frontend/src/api/schemas/agent-events.ts	+14 -0
agent-events.ts	frontend/src/lib/agent-events.ts	+12 -0

backend/app/models/agent_event.py +1 -0

```
@@ -32,6 +32,7 @@ class AgentEventTypeEnum(str, enum.Enum):
    STATUS_UPDATE = "status.update"
    RUN_COMPLETED = "run.completed"
    RUN_ERROR = "run.error"
+   RUN_CANCELLED = "run.cancelled"
```

Adding `RUN_CANCELLED` to the persisted event enum means the PostgreSQL `agent_event_type` type must gain the same value; otherwise any attempt to persist a cancellation event will fail with `invalid input value for enum` at runtime. Previous enum additions were accompanied by Alembic updates (e.g., 202511031720_add_status_update_event_type.py), but this change introduces the new value without a migration, so writes of `run.cancelled` events to the `agent_events` table will be rejected on Postgres deployments.

```


class AgentEvent(Base):
```

backend/harnesses/v2/events.py +12 -0

```
@@ -187,6 +187,17 @@ class AgentErrorEvent(AgentEvent):
    error: str = Field(..., min_length=1, description="Human-readable error message")


+ class AgentCancellationEvent(AgentEvent):
+     """Cancellation event (run.cancelled).
+
+     Emitted when execution is cancelled by user request.
+     This is a terminal event that closes the SSE stream.
+     """
+
+     event_type: Literal["run.cancelled"] = "run.cancelled"
+     run_id: str | None = Field(None, description="Run ID that was cancelled")
+
+
  # Union type for all events
  AgentEventType = (
      AgentMessageEvent
@@ -197,4 +208,5 @@ AgentEventType = (
      | AgentStatusUpdateEvent
      | AgentCompletionEvent
      | AgentErrorEvent
+     | AgentCancellationEvent
  )
```

backend/shared/schemas.py +2 -0

```
@@ -221,6 +221,7 @@ def _get_event_registry() -> dict[AgentEventTypeEnum, Type[AgentEventType]]:
    """Lazy-load event registry to avoid circular imports."""
    # Import here to avoid circular import with harnesses.v2.events
```

Case Study: Datadog’s Incident Prevention

Datadog uses Codex for system-level code review across 1,000+ engineers.

Their incident replay test: they ran Codex against historical PRs that had contributed to incidents.

Results:

- 22% of incidents examined could have been prevented by Codex feedback
- These PRs had already passed human code review - Codex found additional risks
- Engineers shifted from ignoring "bot noise" to treating Codex comments as real feedback

> "A Codex comment feels like the smartest engineer I’ve worked with and who has infinite time to find bugs. It sees connections my brain doesn’t hold all at once."
>
> - Brad Carter, Engineering Manager at Datadog

Part 3: Foundations for Trust

Building Confidence in Async Agents

Why Morning’s Foundation Matters

We covered AGENTS.md, testing infrastructure, and repo setup in sync this morning. Reiterating because these are critical for async workflows.

AGENTS.md improvements compound over time:

- Each time an agent misunderstands something, add a clarifying rule
- Over weeks and months, your agent context becomes rich enough to trust background agents
- Start small, review carefully, update AGENTS.md, gradually increase complexity

Testing infrastructure enables verification:

- Unit tests catch obvious regressions
- API tests against real endpoints catch integration issues
- Preview deployments let agents see what users will see
- CI that agents can read and respond to

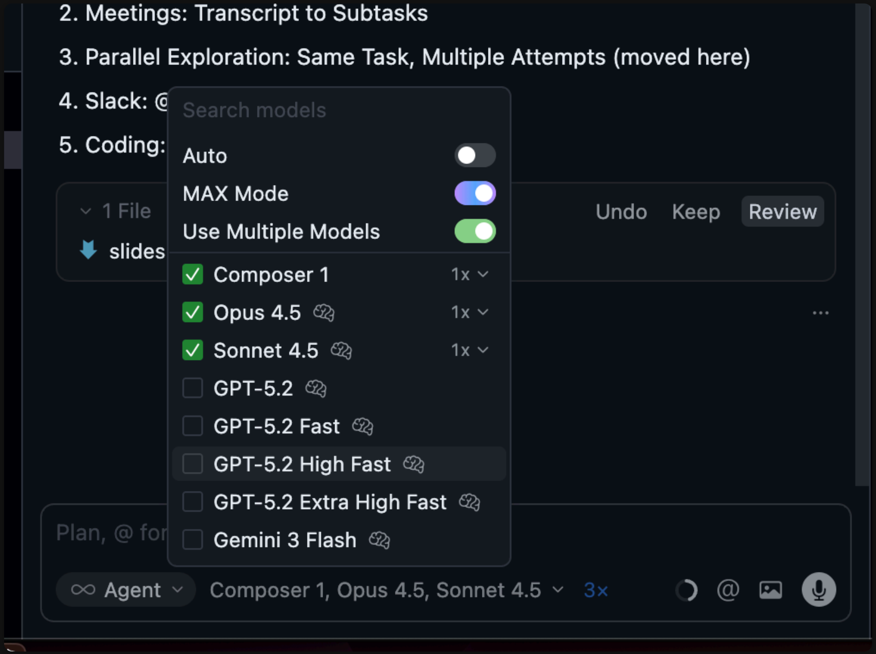
Infrastructure setup enables autonomy:

Same Task, Multiple Models

Cursor is best for this - access to many models in one place:

1. Enable "Use Multiple Models" toggle
2. Select multiple models (Composer, Opus, Sonnet, etc.)
3. Let them run the same task in parallel
4. Compare the implementations
5. Pick the best one, discard the rest

My heuristic: I usually pick the one with the fewest lines of code.



2. Meetings: Transcript to Subtasks
3. Parallel Exploration: Same Task, Multiple Attempts (moved here)
4. Slack: @
5. Coding:

Search models

Auto: off
MAX Mode: on
Use Multiple Models: on

- [✓] Composer 1 — 1x
- [✓] Opus 4.5 — 1x
- [✓] Sonnet 4.5 — 1x
- [ ] GPT-5.2
- [ ] GPT-5.2 Fast
- [ ] GPT-5.2 High Fast
- [ ] GPT-5.2 Extra High Fast
- [ ] Gemini 3 Flash

Part 5: Team Adoption

The AI Champion Role

Reiterating from sync - the AI champion role is essential for team adoption.

If you’re the AI champion, prioritize setting up integrations:

- Work with IT to approve GitHub apps (Cursor, Codex)
- Get Slack workspace permissions for agent bots
- Configure Linear/Jira integrations for your repos
- Set up the secrets and environment variables agents need
- Document which repos have which integrations enabled

These integrations require admin permissions. Someone needs to own this.

Team Adoption

AI Champion: Designate someone to lead adoption and share wins

Track Metrics: Use enterprise features to track token spend per person and team to understand adoption

What’s the first thing you’ll try when you get back? (one task or integration)

Things to Try: Post-Setup Async Tasks

Once async agents are set up, kick off these audits and improvements:

These prompts can be sent via GitHub mentions, Slack, or Cursor Plan Mode. Copy-paste and delegate.

Agent Context & Infrastructure

Improve agent context and foundation:

```
@cursor Install and audit pre-commit hooks.
Check if pre-commit is installed. If not, install it.
Set up pre-commit with ruff format and ruff check (or our project's equivalent).
Do we have both pre-commit (fast checks) and pre-push (slower checks) configured?
Audit what's currently configured and document what runs at each stage.
```

```
@cursor Update AGENTS.md with documentation standards.
Include: how docs should be written, when to update them,
and our conventions for README files and code comments.
Agents should keep documentation up to date when making changes.
```

Testing & CI Optimization

Document and optimize verification infrastructure:

```
@cursor Document how tests should be run in this repo.
Include: local vs CI, fast vs slow tests,
how to run specific test suites, and our testing conventions.
Create or update TESTING.md.
```

```
@cursor Analyze our test suite and CI pipeline.
Which tests are slowest? How long does CI take?
How could we parallelize CI steps?
What caching strategies could we use (dependencies, build artifacts)?
How could we split them into fast (pre-push) vs slow (CI-only)?
Create an optimization plan with parallelization and caching.
```

Setup Time

Configure Your Async Tooling

Enabling Codex Code Review

Turn on automatic PR reviews from Codex:

1. Go to chatgpt.com/codex
2. Navigate to Settings > GitHub Integration
3. Install the Codex GitHub App for your repositories
4. Grant access to repos where you want automatic reviews
5. Codex will automatically review PRs and identify potential issues

What you get:

- Automatic reviews on every PR in connected repos
- Reviews prioritize significant issues over nitpicks
- Tag `@codex` in PR comments to request specific reviews or changes

Works great with `/gh-address-pr-comments` - Codex leaves review comments, then use the command to address them systematically. Complete async workflow.

Enabling Cursor Code Review (Bugbot)

Turn on automatic PR reviews from Cursor:

1. Go to cursor.com/dashboard
2. Navigate to Integrations > GitHub
3. Install the Cursor GitHub App (also called Bugbot)
4. Select repositories where you want automatic reviews
5. Reviews will appear automatically on new PRs

What you get:

- Automatic code review comments on every PR
- Identifies bugs, security issues, and code quality problems

Setting Up Cloud Agent Environments

Configure environments so agents have what they need:

Codex:

- Go to chatgpt.com/codex > Settings
- Add environment secrets (API keys, credentials) per repository
- Secrets are available to agents when they run tasks

Cursor:

- Run `Cursor: Start Cloud Agent Setup` from command palette
- Configure base environment, install commands, and secrets
- Or create `.cursor/environment.json` manually:
  - Define `install` command (e.g., `npm install`)
  - Set up terminal commands (dev servers, etc.)
  - Create snapshots for reuse across repos

Questions?

Resources:

- Codex: chatgpt.com/codex
- Codex GitHub App: github.com/apps/chatgpt-codex-connector
- Cursor Cloud Agents: cursor.com/docs/cloud-agent
- Cursor GitHub: cursor.com/docs/integrations/github

Hands-On Time

The rest of this session is for you to **set up these integrations with your team**.

Vignesh and I will be here to help, and we’ll try to **kick off some background jobs together**.