

OpenAI Agents API 2026

Build Long-Running AI Agents

Durable cloud agents, managed sessions, tool orchestration, sandboxes, subagents, recovery, and production safety.

2026 EDITION • VERIFIED 29 SEPTEMBER 2026

Contents

- 01 Cover
- 02 Contents
- 03 Why the Agents API Matters
- 04 Core Architecture
- 05 Create a Reusable Agent
- 06 Durable Sessions
- 07 Long-Running Turns
- 08 Events, Streaming & Webhooks
- 09 Choose the Right Environment
- 10 Tools, MCP & External Systems
- 11 Production System Prompt
- 12 Multi-Agent Orchestration
- 13 Function Calls as Approval Gates
- 14 Steer Active Work
- 15 Context, Memory & Compaction
- 16 Observability & Usage
- 17 Security, Sandboxing & Secrets
- 18 Recovery & Failure Patterns
- 19 Production Blueprint
- 20 Disclaimer

Why the Agents API Matters

The Agents API is the managed runtime for durable cloud agents. OpenAI runs the Codex harness while your application defines the job, tools, environment, and product experience.

- Built for multi-step work that may outlive a single HTTP request.
- Sessions preserve the agent configuration, conversation, and saved work over time.
- The managed harness handles orchestration, automatic context compaction, and recovery.
- Use the Agents SDK instead when you want the agent loop to run inside your own application.

KEY TAKEAWAY

Core idea: treat the agent as a durable worker with state, not as a one-shot completion endpoint.

Core Architecture

A production Agents API workflow has four main resources. Keeping these boundaries clear prevents state, security, and lifecycle mistakes.

- Agent - reusable model, instructions, tools, reasoning, and multi-agent settings.
- Session - durable instance that receives tasks, stores turns, and can be continued later.
- Environment - optional compute and filesystem: none, OpenAI-hosted, or self-hosted.
- Events and items - the live record of input, progress, tool calls, output, and state changes.

KEY TAKEAWAY

The harness is managed by OpenAI; your app remains responsible for product logic, authorization, and any application-run tools.

Create a Reusable Agent

Persist a reusable agent when multiple sessions should share the same core behavior. Store only configuration - never credentials.

COPY CODE

```
from openai import OpenAI

client = OpenAI()

agent = client.beta.agents.create(
    name="Research Operator",
    model="gpt-6-astra",
    instructions=(
        "Research carefully, cite evidence, use tools only "
        "when needed, and never invent missing facts."
    ),
    multi_agent={"enabled": True, "max_concurrent_subagents": 3},
)

print(agent.id)
```

IMPLEMENTATION NOTES

- Saved-agent updates affect new sessions; existing sessions keep their resolved configuration.
- Use metadata for app-side identifiers, not secrets.

Durable Sessions

A session is the long-lived unit of work. Reuse the same session for follow-up requests so the agent can continue from its existing conversation and saved work.

COPY CODE

```
session = client.beta.agents.sessions.create(
    agent_id=agent.id,
    environment={"type": "openai_hosted"},
    input=[{
        "role": "user",
        "content": [{
            "type": "input_text",
            "text": "Audit this project and produce a remediation plan."
        }]
    }],
)

session_id = session.id
```

IMPLEMENTATION NOTES

- Persist session IDs in your application database.
- One session can contain many turns; do not create a new session for every follow-up.

Long-Running Turns

Turns run asynchronously. A complex task can keep working while your client disconnects, and the session can later be inspected or continued.

- Input sent to an idle session starts a new turn.
- Input sent while a turn is active steers the current work instead of creating a separate conversation.
- Check the turn outcome explicitly: completed, failed, or cancelled.
- A completed turn still requires output inspection; individual tools may have failed or returned partial data.

KEY TAKEAWAY

Production rule: do not equate “session is idle” with “task succeeded.” Inspect the turn outcome and final artifacts.

Events, Streaming & Webhooks

Long-running agents need a control plane. Use event streams for live UX and webhooks for lifecycle automation when your process is not continuously connected.

- Streaming exposes progress and output events during a turn.
- Webhooks can react to created, in-progress, idle, failed, and action-required session states.
- The action-required state can signal a pending function result or an environment connection.
- Design webhook handlers to be idempotent so retries cannot provision duplicate compute or repeat side effects.

KEY TAKEAWAY

Use streaming for the user experience; use webhooks for durable backend reactions.

Choose the Right Environment

The environment controls where the agent runs commands and works with files. Choose the smallest capability boundary that can complete the task.

- none - best for workflows that only need model reasoning and service-connected tools.
- openai_hosted - managed Linux workspace with files, packages, setup commands, and configurable network access.
- self_hosted - your own laptop, container, VM, or remote sandbox; useful for private networks or custom software.
- Isolate environments by user or workload when data must not be shared.

KEY TAKEAWAY

Environment choice is a security decision, not just a convenience setting.

Tools, MCP & External Systems

Agents become useful when they can safely read current information and take bounded actions. Tool design should make authority explicit.

- Use web search when the job needs fresh public information.
- Use MCP servers to expose structured tools from OpenAI or your own environment.
- Use application function handlers when an action must pass through your authorization layer.
- Keep tool descriptions precise; ambiguous tools increase wrong-call risk and wasted tokens.

KEY TAKEAWAY

Prefer narrow, auditable tools over one powerful “do anything” function.

Production System Prompt

A durable agent needs explicit operating rules for evidence, tools, checkpoints, failure handling, and irreversible actions.

COPY PROMPT

You are a production task agent.

Goal:

Complete the assigned task accurately and leave a clear audit trail.

Operating rules:

1. Break complex work into explicit steps before acting.
2. Use the minimum tools and permissions required.
3. Verify external facts before relying on them.
4. Never invent missing files, credentials, results, or tool output.
5. Before irreversible actions, stop and request application approval.
6. If a tool fails, record the failure, try one safe recovery path, then surface the blocker instead of looping indefinitely.
7. At completion, return: outcome, evidence, files changed, unresolved risks, and the recommended next action.

IMPLEMENTATION NOTES

- Customize authority boundaries for your product.
- Put secrets in secure runtime systems, never in instructions.

Multi-Agent Orchestration

The managed harness can delegate independent work to subagents. Each subagent gets its own context, while the root agent coordinates and combines results.

- Enable multi-agent behavior in the agent configuration and cap concurrency.
- Use subagents for independent research, review, or investigation tasks that can run in parallel.
- Keep dependent steps in the root agent so ordering remains clear.
- Subagents can share configured MCP and web tools; coordinate carefully if they touch the same files.

KEY TAKEAWAY

Parallelism improves throughput only when tasks are genuinely independent.

Function Calls as Approval Gates

When the agent needs an application-run function, the session can enter an action-required state. Your server can validate the request before returning a result.

- Inspect the requested function, arguments, caller identity, and session policy.
- Reject or require human approval for destructive, financial, external-communication, or privilege-changing actions.
- Execute approved functions in your trusted application, not inside untrusted prompt context.
- Return only the function result the agent needs; do not leak credentials or internal policy data.

KEY TAKEAWAY

This pattern turns your application into the authorization boundary for high-impact actions.

Steer Active Work

You do not have to cancel a long task just because priorities change. Sending new input during an active turn steers the current work.

COPY STEERING MESSAGE

Priority update:

- Keep all work completed so far.
- Stop investigating the legacy integration.
- Focus only on the production API path.
- Do not delete or overwrite existing artifacts.
- Before the next external action, summarize the new plan in 5 bullets.
- Continue from the current session state; do not restart the task.

IMPLEMENTATION NOTES

- Steering is best for changing scope, priorities, or constraints.
- Use cancellation when continued work would be unsafe or wasteful.

Context, Memory & Compaction

Long-running work produces more history than you want to resend manually. The Agents API manages the session state and includes automatic context compaction in the managed harness.

- Keep durable business state in your own database even when the session preserves working context.
- Treat session history as operational context, not as your only system of record.
- Store artifact identifiers, external transaction IDs, and approval decisions outside the model conversation.
- Design prompts so the agent can reconstruct intent from durable facts after compaction or recovery.

KEY TAKEAWAY

Separate “what the agent remembers” from “what your product guarantees.”

Observability & Usage

A long-running agent is a distributed workflow. You need visibility into turns, tools, subagents, failures, latency, and token consumption.

- Inspect sessions and turns in the Agents logs on the OpenAI Platform.
- Follow the event stream or saved history for live and post-run debugging.
- Review root-agent and subagent usage separately when diagnosing cost.
- Correlate your own request ID, user ID, and job ID with the OpenAI session ID.

KEY TAKEAWAY

Ship traces and app telemetry together; neither side alone explains the full workflow.

Security, Sandboxing & Secrets

Agent-generated code can access whatever its environment exposes. Security therefore starts with isolation, least privilege, and credential boundaries.

- Keep the application API key outside the sandbox. Use restricted environment credentials only for the connection they need.
- Restrict outbound network access to approved domains whenever possible.
- Keep third-party secrets in vaults, proxies, or application-side function handlers rather than prompts or files.
- Use separate environments for users or workloads that must not share data.

KEY TAKEAWAY

Assume generated code can read files and environment values available inside its sandbox.

Recovery & Failure Patterns

Durability is not the same as infallibility. Build explicit recovery logic around environments, tool calls, partial artifacts, and session outcomes.

- Persist session and environment mappings before starting expensive work.
- Make external side effects idempotent with request keys or transaction IDs.
- On reconnect, inspect saved session state before resubmitting work.
- Use bounded retries with backoff; after the recovery budget is exhausted, surface a clear blocker.
- Verify final artifacts independently before publishing, sending, or deploying them.

KEY TAKEAWAY

The safest retry is one that cannot accidentally repeat the real-world side effect.

Production Blueprint

A practical launch pattern: your application owns identity and authorization; the Agents API owns durable agent execution; the environment owns bounded compute.

- 1. Create or select a reusable agent configuration.
- 2. Create a session and persist its ID with your job record.
- 3. Attach the least-privileged environment and tool set.
- 4. Stream events to the UI and use webhooks for backend lifecycle handling.
- 5. Gate sensitive function calls through application authorization.
- 6. Validate artifacts, record outcomes, and continue the same session for follow-ups.

COPY LAUNCH CHECKLIST

Before launch, confirm:

- ☐ Session IDs are persisted
- ☐ Webhook handlers are idempotent
- ☐ Tool permissions are least-privileged
- ☐ Sandbox network access is restricted
- ☐ Secrets never appear in prompts or artifacts
- ☐ High-impact functions require approval
- ☐ Failure and cancellation paths are tested
- ☐ Final outputs are validated before side effects
- ☐ Usage and session telemetry are monitored

Disclaimer

This guide is provided for educational and informational purposes only. It is not official OpenAI documentation and does not create any warranty, security guarantee, or service commitment.

- API names, beta headers, model availability, pricing, permissions, limits, and behavior can change after publication.
- Always verify implementation details against the current official OpenAI documentation before deploying production systems.
- Code and prompts in this guide are illustrative starting points and must be adapted to your application, security model, legal obligations, and data-handling requirements.
- Do not place production credentials, personal data, or confidential information into prompts or test environments unless your architecture and policies explicitly permit it.

KEY TAKEAWAY

Verification date: 29 September 2026. Primary reference: developers.openai.com