

SKILLNAVORA · AI PRODUCT MANAGEMENT & GENAI
LAUNCHPAD · SKILL 7 OF 7

Agent Orchestration & Monitoring

Run longer AI workflows reliably: workflow patterns, routing and model selection, parallel work, state and checkpoints, human-in-the-loop approvals, error recovery, multi-agent systems, scheduling and triggers, cost control, and production monitoring.

Edition 2026 · skillnavora.in

How to use this guide

Each class has goals, concepts with Indian examples, key takeaways and a practice exercise, from basic to advanced. Finish with the final project to build something you can show in interviews.

	Class	Level
Class 1	Workflows vs agents: start simple	Basic
Class 2	Core patterns: chaining, routing and parallelisation	Basic
Class 3	Orchestrator-workers and evaluator-optimizer	Intermediate
Class 4	State, checkpoints and long-running work	Intermediate
Class 5	Human-in-the-loop, approvals and error recovery	Intermediate
Class 6	Multi-agent systems	Advanced
Class 7	Triggers, scheduling and cost control	Advanced
Class 8	Monitoring, observability and the final project	Advanced
Toolkit: templates, patterns cheat sheet, glossary, 8 interview questions, further reading		

Workflows vs agents: start simple

Know when a fixed workflow is enough and when an agent is worth it.

	Workflow	Agent
Steps	Defined in advance by you	Chosen by the model as it goes
Predictability	High	Lower; needs guardrails
Cost / speed	Lower / faster	Higher / slower
Best for	Known, repeatable processes	Open-ended tasks with unknown steps

Rule: start with one model and a few tools in a clear workflow. Add agent autonomy or multiple agents only when simpler options fail your evals.

Building blocks

An LLM call + retrieval + tools + memory, connected by an orchestrator (code, n8n, Make, Zapier or an agent framework).

Key takeaways

- Workflows are predictable; agents are flexible.
- Start simple; add autonomy only when needed.
- Measure with evals before adding complexity.

Practice exercise: For 3 processes in your work, decide workflow or agent and justify it.

Core patterns: chaining, routing and parallelisation

Use the three most common orchestration patterns.

Pattern	How it works	Example
Prompt chaining	Step 1 output → step 2 input, with checks between	Extract facts → draft PRD → review
Routing	Classify the request, send it to the right workflow or model	Support: refund → refund flow; bug → engineering; simple FAQ → cheap model
Parallelisation	Run independent parts at once, then combine; or run several attempts and vote	Analyse 5 competitors in parallel; 3 judges vote on a risky answer

Model selection inside routing

Use small, fast models for classification and simple answers; larger models for reasoning, writing and agents. Measure quality and cost per successful task, not per call.

Key takeaways

- Chain for sequences, route for variety, parallelise for speed.
- Routing also picks the right model for cost.
- Check outputs between steps.

Practice exercise: Design a router for a customer-support bot with 5 intents, the workflow and model size for each.

Orchestrator-workers and evaluator-optimizer

Use the two most powerful agentic patterns safely.

Orchestrator-workers

A lead model plans the work, sends sub-tasks to workers (other model calls or agents), then combines results. Good for research and analysis where sub-tasks aren't known in advance.

Evaluator-optimizer

One model drafts, another evaluates against a rubric and gives feedback, loop until the bar is met or a step limit is reached. Good for writing, code and PRDs with clear quality criteria.

```
loop max 3 times:
  draft = writer(brief, feedback)
  review = evaluator(draft, rubric)      # returns pass + issues
  if review.pass: break
  feedback = review.issues
return draft, review
```

Always set limits (steps, time, cost) so loops end.

Key takeaways

- Orchestrator plans; workers execute; orchestrator combines.
- Evaluator-optimizer loops on a rubric with a step cap.
- Limits on steps, time and cost are mandatory.

Practice exercise: Sketch an orchestrator-workers design for 'market research report on EV two-wheelers in India'.

State, checkpoints and long-running work

Make workflows resumable and traceable.

- **State:** a record of what's done, what's pending, inputs, outputs and decisions (a table, JSON file or database row per run).
- **Checkpoints:** save after each major step so a failure resumes from there instead of starting over.
- **Idempotency:** re-running a step doesn't duplicate actions (e.g. doesn't create two Jira tickets).
- **Caching:** reuse valid results (search results, document summaries) to save time and money.

```
run_id | step | status (pending/running/done/failed) | input_ref | output_ref | started |  
finished | cost
```

Key takeaways

- Track state per run.
- Checkpoint after major steps.
- Make actions idempotent; cache reusable results.

Practice exercise: Design a state table for a weekly competitor-report workflow with 5 steps.

Human-in-the-loop, approvals and error recovery

Put people at the right points and recover from failures safely.

Where humans add value

- Before consequential actions: sending, publishing, paying, deleting.
- When confidence is low or rules conflict.
- Reviewing samples for quality.

Error recovery

Failure	Response
Timeout / rate limit	Retry with backoff; switch provider
Bad output (fails checks)	Retry with feedback; then escalate to human
Tool error	Log, retry once, fall back to manual path
Unexpected state	Stop, alert owner, keep checkpoint

Key takeaways

- Approvals before consequential actions.
- Retry with limits; fall back; escalate.
- Never fail silently: alert an owner.

Practice exercise: Write the approval points and error-recovery rules for an AI that drafts and sends customer emails.

Multi-agent systems

Know when splitting work across agents helps, and how to coordinate them.

Multiple specialised agents (researcher, analyst, writer, reviewer) can help when work is large, parallel and needs different tools or context. They also add cost, latency and coordination problems.

- Give each agent a clear role, tools and output format.
- Share state through a common store, not long chat histories.
- Use a lead agent or a fixed workflow to coordinate and to decide when work is done.
- Evaluate the final result and each agent's contribution.

Many tasks work best with one capable agent and good tools; prove the multi-agent version is better on your evals before adopting it.

Key takeaways

- Multi-agent helps for large parallel work.
- Clear roles, shared state, a coordinator.
- Prove the benefit with evals.

Practice exercise: Design a 3-agent system for producing a weekly market newsletter and compare it with a single-agent design.

Triggers, scheduling and cost control

Run workflows automatically and within budget.

Triggers

- **Scheduled:** every morning, every Friday (cron, n8n schedule).
- **Event-driven:** new form submission, new support ticket, file uploaded, price change detected.
- **On demand:** a button, Slack command or chat request.

Cost management

- Budget per run and per month; alert at 80%.
- Route easy work to smaller models; cache; trim context.
- Cap steps, retries and tool calls.
- Track cost per successful task as a key metric.

Key takeaways

- Choose scheduled, event-driven or on-demand triggers.
- Set budgets and alerts.
- Measure cost per successful task.

Practice exercise: Define triggers and a monthly budget with alerts for 2 workflows you'd automate.

Monitoring, observability and the final project

Know when a workflow breaks or degrades, and fix it fast.

What to monitor

Signal	Example alert
Success rate	Runs succeeding fell below 95%
Quality	Sampled judge score dropped 5 points
Latency	p95 above 30 s
Cost	Daily spend above budget
Safety	Any unapproved action or injection flag
Drift	New topics or languages in inputs

Tools

Traces and dashboards in Langfuse, LangSmith, Arize Phoenix or Braintrust; n8n execution logs; simple Google Sheet logs and Slack alerts for no-code workflows.

Final project

Build one end-to-end workflow (e.g. "new customer feedback → classify → summarise weekly → Notion report → Slack"), with routing, a checkpointed state table, one approval step, retries, a budget, a monitoring sheet and a 10-case eval. Demo it in 5 minutes.

Key takeaways

- Monitor success, quality, latency, cost, safety, drift.
- Trace every run for debugging.
- Ship with alerts and an owner.

Practice exercise: Complete the final project.

Templates

A. Workflow design sheet

Field	Fill in
Goal and trigger	
Steps (pattern: chain / route / parallel / orchestrator)	
Models per step	
Tools	
State and checkpoints	
Approvals	
Error recovery	
Budget and limits	
Monitoring and owner	
Eval set	

B. Run log

```
run_id | trigger | steps done | status | cost | latency | quality score | approvals | errors
```

Patterns cheat sheet

Term / tool	Meaning / use
Prompt chaining	Sequential steps with checks
Routing	Classify then send to the right path or model
Parallelisation	Independent parts at once; or voting
Orchestrator-workers	Lead plans and delegates; combines results
Evaluator-optimizer	Draft → evaluate → improve loop with a cap
Checkpoint	Saved progress to resume after failure
Idempotency	Re-running doesn't duplicate actions
Human-in-the-loop	Approval or review by a person
Observability	Traces, logs and metrics to see what happened

Glossary

Orchestrator	Component that runs and connects steps
Trigger	Event or schedule that starts a workflow
Backoff	Waiting longer between retries
Trace	Record of every step in a run
Drift	Inputs or quality changing over time

8 interview questions with answers

Q1. Workflow or agent?

Workflow when steps are known and repeatable; agent when steps can't be predicted. Start simple and prove the agent is better with evals.

Q2. Explain routing.

Classify each request and send it to the appropriate workflow or model, e.g. cheap model for FAQs, strong model for complex cases.

Q3. What is orchestrator-workers?

A lead model breaks a task into sub-tasks, delegates to workers and synthesises their results.

Q4. How do you make long workflows reliable?

State tracking, checkpoints, idempotent actions, retries with limits, approvals, alerts and monitoring.

Q5. When do you need multiple agents?

Large parallel work needing different tools or context, and only when evals show it beats a single agent.

Q6. How do you control AI workflow cost?

Budgets and alerts, routing to smaller models, caching, trimmed context, step and retry caps, and tracking cost per successful task.

Q7. What would you monitor in production?

Success rate, sampled quality scores, latency, cost, safety events and input drift, with owners and alerts.

Q8. Where do humans fit?

Before consequential actions, on low confidence or conflicts, and in sampled quality review.

Further reading and tools

- Anthropic: Building effective agents: <https://www.anthropic.com/research/building-effective-agents>
- Anthropic: Demystifying evals for AI agents: <https://www.anthropic.com/research/demystifying-evals-for-ai-agents>
- n8n docs: <https://docs.n8n.io/>
- LangGraph docs: <https://langchain-ai.github.io/langgraph/>
- Langfuse docs: <https://langfuse.com/docs>
- OpenAI Agents SDK: <https://openai.github.io/openai-agents-python/>

© SkillNavora · skillnavora.in · Part of the AI Product Management and GenAI Launchpad programs. Practise with the free AI Tutor, AI PM Toolkit and Practice Lab on skillnavora.in.