# Make.com: Scenarios, Routers and AI Agents

Make is a visual automation tool. You connect apps as boxes on a canvas and draw the flow between them. This guide takes you from your first scenario to the Resume Analysis Agent from the class. It uses fake data and a test account throughout.

## 1. Words you need first

| Word | Meaning |
|---|---|
| Scenario | One automation: a chain of modules that runs in order. |
| Module | One step, such as "Watch responses" or "Add a row". |
| Trigger | The first module, which starts the scenario (a new form response, a schedule, a webhook). |
| Bundle | One item of data passing between modules. |
| Router | A split that sends each bundle down one or more paths based on filters. |
| Filter | A condition on a path, for example "score is above 70". |
| Connection | The saved login to an app (a credential). Use a test account. |
| Operation | Spend of the plan's credits per module run. Check the plan page for current credit rules. |

## 2. The Make screen

The left menu has: **Org**, **Scenarios**, **AI Agents**, **Credentials**, **Webhooks**, **MCP Toolboxes**, **Templates**, **Data stores** and **More**. The canvas in the middle is where you build. The bottom toolbar has **Run once**, a schedule toggle (for example every 15 minutes) and the undo and settings icons.

## 3. Your first scenario: form to sheet

1. Click **Scenarios**, then **Create a new scenario**.
2. Add a **Google Forms** module, choose **Watch Responses**, and connect the test account. Pick a test form with two fields.

3. Add a **Google Sheets** module and choose **Add a Row**. Select a test sheet.
4. Map each form answer to a column using the mapping panel (click a field, choose the answer).
5. Click **Run once**, then submit one test response to the form.

**You should see:** the module bubbles turn green, the sheet gets a new row, and the output panel shows the values you mapped.

## 4. Scheduling and settings

- **Schedule:** the toggle sets how often the scenario checks for new items. The class example was every 15 minutes. Shorter intervals use more operations.
- **Sequential running:** keep it on when order matters, so two runs cannot write at the same time.
- **Error handling:** right-click a module to add an error route. Route failures to a sheet row or a message so they are visible.
- **Notifications:** decide when Make emails you about failed runs. Turn on notifications for failures, not for every success.

> **Before you enable a scenario,** check three things: its operations per month against your plan, its notification setting, and what happens on failure. A scenario left running on a schedule keeps spending operations.

## 5. Routers and filters

A router is a fork in the road. The class example, the Resume Analysis Agent, uses one.

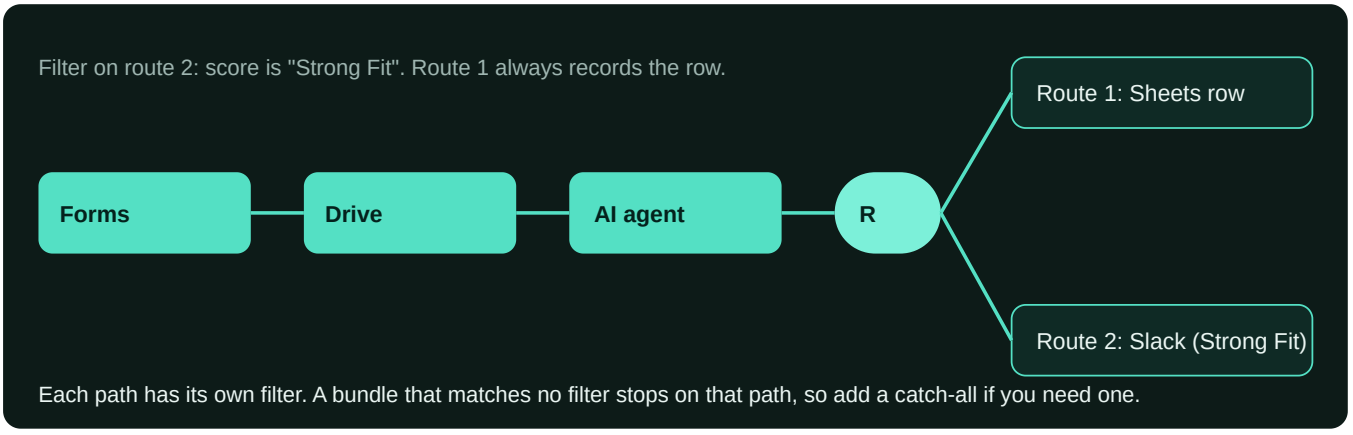



Figure 1. A router splits bundles into paths. Each path has a filter.

1. Click the wrench icon between two modules and choose **Add router**.
2. Add a filter to each route (for example the AI score field equals "Strong Fit").

3. Test with one bundle that matches and one that does not. Confirm only the right route runs.

## 6. The Resume Analysis Agent (class example)

The class scenario was built to read a candidate's resume and sort the candidate. The full chain is:

1. **Google Forms (Watch Responses):** the candidate submits a form with a resume file link.
2. **Google Drive (Download a File):** the resume file is fetched from Drive.
3. **AI agent:** reads the resume, uses a knowledge source (the job criteria) and returns a fit rating.
4. **Router:** route 1 adds a row to Google Sheets for every candidate; route 2 sends a Slack message only for "Strong Fit" candidates.

The agent in the screenshot also had an **Add tool** option, an **Add MCP** option, a **Chat** option and a knowledge attachment. The AI model module was an OpenAI connection in the class.

> **Privacy.** A resume contains personal data. For practice, use a fake resume with a made-up name and email. Do not send real candidates' files to a test scenario, and do not post Slack messages to a shared channel during testing.

### Build checklist

| Step | You should see |
|---|---|
| Forms trigger runs on a test response | One bundle with the form fields |
| Drive download returns a file | A file name and size in the output |
| AI step returns a fixed-format rating | One word from your allowed list (for example Strong Fit, Possible, No Fit) |
| Router sends only matching bundles to Slack | Slack message only for Strong Fit |
| Failed AI step | Error route writes a row marked "needs attention" |

## 7. Maia: building with a description

Maia is Make's AI assistant (labelled Beta in the class screenshots). You describe what you want in plain English, and Maia proposes modules. The panel offers "Continue where we left off", "Get inspiration" and "Ask a question about Make".

1. Open a scenario and click **Build with Maia**.

2. Describe the goal in one sentence: "When a form is submitted, download the attached file, ask an AI model to rate it, and add a row to a sheet."
3. Review every module Maia proposes. Check each connection, field mapping and filter yourself.
4. Never trust a generated scenario without running it on fake data first.

Maia is a drafting aid. The person who runs the scenario owns its results, so review every step.

## 8. MCP Toolboxes and AI Agents in Make

**MCP** (Model Context Protocol) is a standard way for an AI app to use outside tools. Make's MCP Toolboxes group tools that an agent can call. Give an agent only the tools it needs, and read-only tools where possible. The same least-permission rule from the skills and agents guide applies here.

## 9. Failure, retries and cost

| Issue | What to do |
|---|---|
| Module fails on a bad file | Route the error to a "needs attention" row; fix the input; re-run that bundle. |
| Scenario runs too often | Lengthen the schedule or use a webhook trigger instead of polling. |
| Operations run out | Check the plan usage page and reduce the schedule or the number of modules per run. |
| Duplicate rows | Store the form response ID and skip responses already in the sheet. |

Check Make's current plan and credit page before you estimate cost. Plans and credit rules change.

## 10. Quick check

1. What starts a scenario?
2. What does a filter do on a router path?
3. Why should you use a test connection?
4. What should you do before enabling a scenario that Maia built?

**Answers:** 1. The trigger module (for example a new form response). 2. Decides whether a bundle goes down that path. 3. So a mistake cannot touch real accounts or data. 4. Run it on fake data and check every module and filter.

## 11. Glossary and further reading

- Make help centre: help.make.com (scenarios, routers, Maia and MCP pages).
- Make pricing and credit page: check the live page before you estimate operations.

## Part 2. Every Make option you need, explained

Make's labels can change between releases. The terms below are the concepts you will meet; if a menu name differs, look for the closest match.

### 2.1 Words on the canvas

| Word | Meaning | Example |
|---|---|---|
| Scenario | The whole automation | Resume Analysis Agent |
| Module | One step (a bubble on the canvas) | Google Sheets: Add a Row |
| Bundle | One item of data passing between modules | One form response |
| Route (connection) | The line that carries bundles from one module to the next | Watch Responses → Download a File |
| Mapping | Choosing which earlier field fills a later field | Email field → Email column |
| Connection | Saved login for an app | Google account (test) |
| Data store | Make's built-in table for keeping records between runs | Processed response IDs |
| Operation | One module action; plans include a set number. Check the live plan page. | Each module run counts |

### 2.2 Trigger modules: how a scenario starts

| Trigger type | Starts when | Use it for | Watch out |
|---|---|---|---|
| Schedule | At the interval you set (the class used every 15 minutes) | Checking a sheet or inbox regularly | Each check uses operations even when nothing is new |
| Watch responses (Google Forms) | A new form response arrives | Intake forms | File uploads come as links; you need a Drive download step |
| Custom webhook | Another app posts to the Make URL | Real-time events from your own app | Use a test run while building; activate the scenario for live use |

| | | | |
|---|---|---|---|
| Watch new rows (Sheets) | A new row appears | Reacting to sheet data | Sheet changes can be batched; test with a few rows |
| Manual run | You press Run once | Building and debugging | Not automatic |

### 2.3 Module types you will use

| Module type | Examples | What to set |
|---|---|---|
| App action | Google Drive: Download a File; Google Sheets: Add a Row; Slack: Send a Message; Gmail: Send an Email | Connection, the action, and the field mapping |
| Flow control | Router, Filter, Repeater, Iterator, Aggregator, Sleep | Routes, filter conditions, the batch logic |
| Data tools | Set variable, Text parser, JSON parse, Tools | Use these to clean data before an AI step |
| Data store | Data store: Add record, Search records, Get record | Store, key field, record fields |
| AI module | An AI model call (OpenAI in the class example; check the apps list for other providers) | Connection, model, prompt, and output format |
| AI agent | The agent builder: tools, knowledge, chat | Goal, tools with least permission, stop rule |
| Error handler | Break, Resume, Commit, Rollback, Ignore directives on a module | Choose per module (section 4) |

### 2.4 Filters and routers in detail

| Concept | Plain meaning | Example |
|---|---|---|
| Router | Splits one route into several; each path has a filter | Path 1: all candidates to Sheets; Path 2: Strong Fit to Slack |
| Filter | A condition on a path; only matching bundles pass | AI rating equals "Strong Fit" |
| Fallback path | A path with no filter that catches everything else | Route anything unclassified to a review sheet |
| Text operators | Contains, equals, starts with | Rating text contains "Strong" |
| Number and date operators | Greater than, less than, after | Start date after today |

> **Always add a fallback path** on a router. Without it, a bundle that matches no filter stops quietly and you will not know it happened.

## Part 3. Build your own Make scenario: a 12-step recipe

1. **Write the goal in one sentence.** Example: "When a candidate submits a fake resume, rate it and log it."
   *Check:* you can state the output.
2. **List the inputs and fields**, with example values.
   *Check:* every field has a name and a sample.
3. **Choose the trigger**: manual to build, a form or webhook for real events, a schedule for polling.
   *Check:* Run once returns one bundle.
4. **Add the first action** and map a field. Run once.
   *Check:* the target row or message shows the mapped values.
5. **Add validation**: a filter that stops empty or malformed bundles, routed to a review path.
   *Check:* a bad input goes to review, not to the output.
6. **Add the AI step last among the data steps.** Ask for one fixed word from a list, not free text.
   *Check:* the output matches the list every time in five tests.
7. **Add the router** with a filter per path and a fallback.
   *Check:* each test bundle reaches exactly one path.
8. **Add the external action** (Slack, Gmail) only on the right path and only with a test channel or address.
   *Check:* the test channel receives one message per matching bundle.
9. **Set error handling** on each risky module (section 4).
   *Check:* force a failure; the error route writes "needs attention".
10. **Prevent duplicates** with a data store: search for the response ID before writing.
    *Check:* running the same bundle twice writes one row.
11. **Test five inputs**: normal, empty field, duplicate, unexpected AI output, and a tool failure.
    *Check:* each result matches the test plan (section 6).
12. **Activate and watch the first runs**, then review history daily while learning.
    *Check:* the run history shows green runs and any red runs are explained.

## Part 4. Error handling, explained simply

| Directive | What it does | Use it when |
|---|---|---|
| Resume | Continues the scenario using a substitute value | A non-critical field is missing |

| | | |
|---|---|---|
| Ignore | Skips the failed bundle and continues the run | Losing this one item is acceptable and logged |
| Break | Stores the failed bundle to retry later | A temporary outage (for example an app is down) |
| Commit | Keeps the work done so far and stops the run | Partial results are still valid |
| Rollback | Undoes changes where the app supports it | An all-or-nothing change |

For practice, use **Break** for external steps (so you can retry them) and **Resume** for optional fields. Write the choice in a note on the scenario, so the next person knows why it is set that way.

## Part 5. The Resume Analysis Agent, built step by step (fake data only)

> **Fake data only.** Create a test Google Form with a made-up candidate (for example "Test Candidate", candidate.test@example.com) and a sample file. Do not upload real resumes. This tool must not make hiring decisions; a person reviews every result.

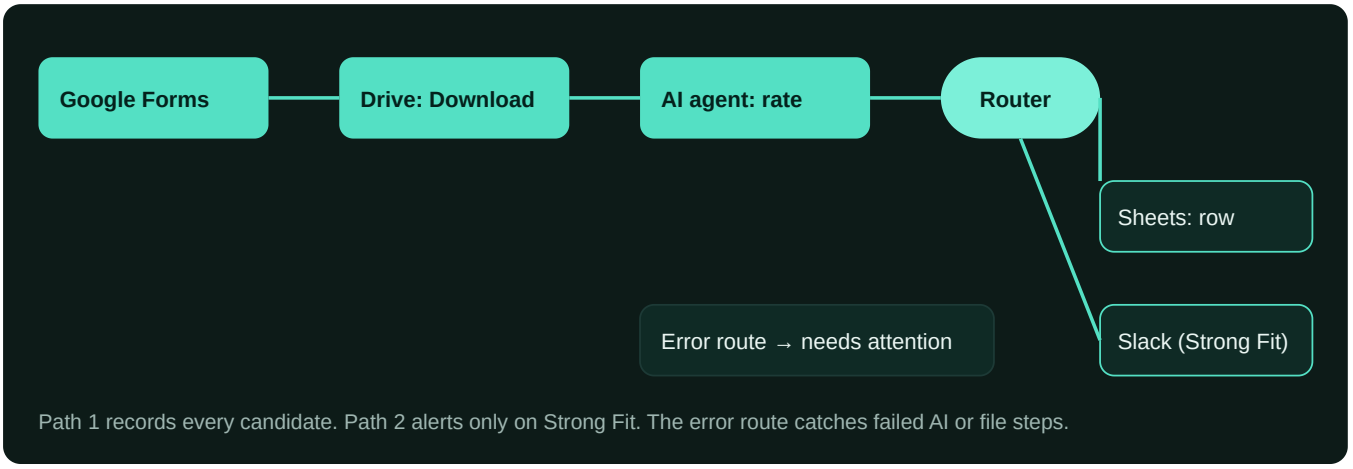



Figure 3. The Resume Analysis Agent layout, redrawn for SkillNavora.

1. **Create the scenario** "Resume analysis (test)" and add **Google Forms: Watch Responses**. Choose the test form and run once after one fake submission.
2. **Add Google Drive: Download a File**. Map the file ID from the form's upload field. Run once.
   *Check:* the bundle now includes the file name and content.
3. **Add the AI step.** Use one prompt with a fixed output: "Reply with exactly one of: Strong Fit, Possible, No Fit. Base the answer only on the job criteria provided." Provide the criteria as fixed text, not a real job list.
   *Check:* five fake resumes return only those three words.

4. **Add the router.** Path 1: Google Sheets Add a Row for every bundle (fallback-free). Path 2: a filter "rating equals Strong Fit", then Slack Send a Message to a test channel.
   *Check:* a Strong Fit test reaches both paths; a No Fit test reaches only Sheets.
5. **Add error handling.** On the Drive and AI modules, set Break; add an error route that writes "needs attention" and the error text to a sheet.
   *Check:* disconnect the Drive link for one run; the error row appears and no Slack message is sent.
6. **Add a duplicate check** with a data store keyed by form response ID.
   *Check:* the same response submitted twice writes one row.
7. **Review every output** before any real use. A person decides what happens next.

## Part 6. Maia, the AI scenario builder

Maia (labelled Beta in the class screenshots) drafts a scenario from a plain-language description. The panel offers "Continue where we left off", "Get inspiration" and "Ask a question about Make".

| Maia can help with | You must still check |
|---|---|
| Drafting the module order from a description | Each connection and which account it uses |
| Suggesting modules you have not used before | Every field mapping and filter |
| Explaining an existing scenario | Whether the scenario is safe to run with real data |
| Answering "how do I" questions about Make | That the answer matches the current product (check the help centre) |

Rule: never run a Maia-built scenario on real data before you have run it on fake data and read each module's output.

## Part 7. MCP and Make

**MCP** (Model Context Protocol) is a standard way for AI applications to use outside tools. Make's MCP toolboxes group tools an agent can call. A tool is a single action, such as reading a sheet or posting a message.

| Concept | Plain meaning | Safe default |
|---|---|---|
| Input | What the agent sends to the tool | Only the fields the tool needs |
| Output | What the tool returns to the agent | Read-only results where possible |

| | | |
|---|---|---|
| Scope | Which account data the tool can reach | A test account with fake data |
| Permission | Whether the tool can read, write or send | Read first; add write only when needed |
| Approval boundary | The point where a person must approve | Before any send, delete or access change |
| Audit log | A record of each tool call | Keep it, without personal data |

## Part 8. Full test plan

| Test | Input | Pass when |
|---|---|---|
| Happy path | Fake strong candidate | Sheets row and Slack alert |
| Weak candidate | Fake no-fit candidate | Sheets row only |
| Unexpected AI output | Forced odd answer | Error route, no Slack message |
| Missing file | Form with no upload | Error route with a clear reason |
| Duplicate response | Same response twice | One row only |
| Slack down | Disconnect Slack | Break directive stores the bundle for retry |

## Part 9. Common errors

| Error | Cause | Fix |
|---|---|---|
| Empty output field | Mapping points to the wrong module | Re-map from the latest module output |
| Connection expired | Login was revoked | Reconnect the test account |
| Scenario runs but nothing happens | Router has no matching path and no fallback | Add a fallback path |
| Operations used up quickly | Short schedule on a large scenario | Lengthen the schedule; reduce modules |
| Duplicate rows | No duplicate check | Add the data store check (Part 3, step 10) |

## Part 10. Final checklist before you go live

| Item | Done |
|---|---|

| | |
|---|---|
| Test account and fake data only | [ ] |
| Every router has a fallback path | [ ] |
| Error routes on Break or Resume for every risky module | [ ] |
| Duplicate check tested | [ ] |
| Human review before any hiring-related action | [ ] |
| Operations and schedule checked against the live plan page | [ ] |
| Run history reviewed after the first ten runs | [ ] |

## Part 11. Labs: five practice exercises with fake data

### Lab 1: form to sheet (25 minutes)

1. Create a test Google Form with two fields. Build the scenario: Watch Responses, then Add a Row.
2. Submit one fake response and run once.
3. **Pass when:** one row appears with each answer in the right column.

### Lab 2: filter and fallback (30 minutes)

1. Add a router after the form. Path 1 with a filter "answer equals Yes" goes to Sheets. Path 2 is a fallback that goes to a review sheet.
2. Submit one "Yes" and one "No" response.
3. **Pass when:** each response reaches exactly one path, and nothing disappears.

### Lab 3: data store duplicate check (40 minutes)

1. Create a data store with a key field for the response ID.
2. Before the Sheets module, search the store for the response ID; continue only if nothing is found, then add the record.
3. **Pass when:** submitting the same response twice writes one row.

### Lab 4: AI output lock (40 minutes)

1. Add an AI module with a prompt that asks for one word from a list (Strong Fit, Possible, No Fit).
2. Add a filter that accepts only those three words; send anything else to the review path.
3. Run five fake inputs, including one that tempts a long answer.
4. **Pass when:** every bundle reaches a path, and no free-text answer reaches the sheet.

### Lab 5: break and retry (30 minutes)

1. Set Break on the Slack module. Disconnect the test Slack connection for one run.
2. Reconnect and run the incomplete execution from the scenario's history.
3. **Pass when:** the bundle is retried and the message arrives once, not twice.

## Part 12. Troubleshooting table

| Symptom | Most likely cause | What to check first |
|---|---|---|
| Module shows "no data" | Upstream module produced no bundle | Run the previous module once and read its output |
| Mapping shows a stale value | Old output cached | Re-run from the module that changed |
| Router sends nothing | No matching filter and no fallback | Add the fallback path |
| Scenario is on but not running | Schedule off, or trigger not set | Check the schedule toggle and the trigger settings |
| Connection needs re-authorisation | Login expired or permissions changed | Reconnect the test account |
| Operations run out early | Frequent schedule or many modules | Lengthen the schedule; check the usage page |
| AI output varies | Free-text answer | Ask for one word from a list; lock with a filter |

## Part 13. Choosing between Make, n8n and a custom app

| Situation | Good fit | Why |
|---|---|---|
| Visual, fast automation across many business apps | Make | Visual routers and filters; quick to test |
| Self-hosting, custom code and data control | n8n | Can run on your own server with code nodes |
| Core product logic with users and a database | A custom app (for example on Replit) | Full control of the user experience and data |

The three can work together. For example, a form in an app, an automation in Make, and an agent in n8n. Keep each boundary clear and document it.

## Part 14. Full glossary

| Term | Plain meaning |
|---|---|
| Scenario | One automation made of connected modules. |
| Module | One step, such as an app action or a flow control. |
| Bundle | One item of data moving through a scenario. |
| Router | A split into paths, each with a filter. |
| Filter | A condition that decides whether a bundle continues. |
| Fallback path | A path that catches anything no filter matched. |
| Data store | A built-in table that keeps records between runs. |
| Directive | An error rule on a module: Resume, Ignore, Break, Commit or Rollback. |
| MCP | A standard way for AI apps to use outside tools. |

## Part 15. Quiz (answers follow)

1. What starts a scenario?
2. Why add a fallback path to a router?
3. How do you stop duplicate rows?
4. Why lock AI output to a list of words?
5. What should a Break directive do with a failed bundle?

**Answers:** 1. The trigger module, such as a new form response. 2. So unmatched bundles are not lost. 3. Search a data store for the response ID before writing. 4. So the filter can route predictably and no free text reaches the sheet. 5. Store it so it can be retried.

## Part 16. Build-your-own planning sheet

| Question | Your answer |
|---|---|
| What starts my scenario? | [ ] |
| Which apps do I connect, and with which test account? | [ ] |
| Where does each path end? | [ ] |
| Which module is the AI step, and what is its locked output? | [ ] |
| Which step is external, and who approves it? | [ ] |

| | |
|---|---|
| What is the error directive on each risky module? | [ ] |
| How do I prevent duplicates? | [ ] |

## Part 17. Industry scenarios for Make

| Industry | Trigger | Flow control | Record | Approval before |
|---|---|---|---|---|
| Retail | New order from a test store form | Router: standard vs. high-value orders | Orders data store | Refund or change to an order |
| Recruiting (fake candidates) | Application form response | Router with fallback: Strong Fit, Possible, No Fit, Needs review | Candidates sheet | Any message to a candidate |
| Clinics | Appointment request (fake data) | Filter on slot availability | Booking sheet | Any confirmation to a patient |
| Education | Quiz submission | Filter on score band | Results sheet | Sending feedback |
| Logistics | Tracking exception | Router by exception type | Exceptions sheet | Rerouting or refund |

> **Hiring, medical and eligibility decisions must not be automated here.** The scenario may sort and prepare, but a person decides.

### 17.1 Recruiting example, node by node (fake candidates)

1. **Google Forms: Watch Responses** (test form, fake candidate).
2. **Data store: Search records** by response ID. If found, stop (duplicate).
3. **Google Drive: Download a File** (fake file).
4. **AI module** returns one word from the fixed list, plus a short reason field of 20 words or fewer.
5. **Router**: Path 1 Strong Fit; Path 2 Possible; Path 3 No Fit; Path 4 fallback to Needs review.
6. **Each path** writes one row to the candidates sheet with the path name.
7. **Strong Fit only**: Slack message to a test channel that says "Review needed" and gives the row number, not the candidate's details.
8. **Data store: Add record** with the response ID, so the duplicate check works next time.

**Pass when:** every fake response reaches one path, the duplicate is stopped, and the Slack message contains no personal details.

## Part 18. Operations and cost, without vendor prices

Every module run uses operations, and an AI call also uses model tokens. Plan both before you enable a scenario. Use the live plan page for the numbers; this table shows how to count.

| Item | How to count | Example (fake numbers) |
|---|---|---|
| Operations per bundle | One per module that runs | 8 modules = 8 operations per bundle |
| Bundles per month | Submissions per day times 30 | 20 per day = 600 per month |
| Operations per month | Bundles times operations per bundle | 600 × 8 = 4,800 |
| Polling cost | Runs per day from the schedule that find nothing | Every 15 minutes = 96 runs per day, even when empty |
| Model tokens | Input plus output per AI call, times calls | Use the tokens-guide formula with current rates |

**Lesson:** a frequent schedule on a large scenario spends operations on empty checks. A webhook trigger avoids that cost when the event is real-time.

## Part 19. Production checklist

| Item | Done |
|---|---|
| Every router has a fallback path | [ ] |
| Error directive set on each external module | [ ] |
| Duplicate check tested with the same input twice | [ ] |
| Test connections only during build; production connection reviewed | [ ] |
| Operations per month estimated from the live plan page | [ ] |
| Notifications set to failures only | [ ] |
| Run history reviewed after the first ten runs | [ ] |
| Named owner for the scenario and its data | [ ] |

## Part 20. Exit exercise

1. Pick one industry from Part 17 and write the goal in one sentence.
2. Build the trigger, one filter, one router with a fallback, and one record step, with fake data.

3. Add the AI step last, with a locked one-word output.
4. Run five fake inputs: normal, empty, duplicate, odd AI output, and a failed module.
5. **Pass when:** every input reaches the right path, the duplicate is stopped, and the failure is visible.