



Skill Navora

TRACK 10

AI & No-Code Tools Lab

Complete tutorials for the design and automation tools used across the programs: Figma (design systems, prototypes, Dev Mode, Figma Make) and n8n (workflows, APIs, AI agents, MCP).

2 lessons

Version 2026.10

Updated October 2026

Created by **Varun Kumar**
www.skillnavora.in

Contents

Every entry is a clickable link. The PDF also has bookmarks for each lesson and section.

[01 Figma Complete Tutorial: Interface, Auto Layout, Components, Variables, Prototypes, Dev Mode and Figma AI](#)

[02 n8n Complete Tutorial: Workflow Automation, Data Pipelines and AI Agents](#)

Figma Complete Tutorial: Interface, Auto Layout, Components, Variables, Prototypes, Dev Mode and Figma AI

Learn Figma from zero to a handoff-ready design. Tour the interface, master frames, shapes, text and the 8-point grid, build responsive layouts with auto layout and constraints, create components with variants and properties, use styles and variables (light and dark modes), prototype flows with Smart Animate and overlays, hand off in Dev Mode, run workshops in FigJam, and generate working prototypes with Figma AI and Figma Make. Includes a dashboard-design project for data roles.

✦ Figma 2026 (Figma Design, FigJam, Dev Mode, Figma Make)

Beginner to Advanced

🕒 180 min

By the end of this lesson you can

- Navigate Figma Design and know when to use Figma Design, FigJam, Dev Mode, Slides, Sites and Make
- Build screens with frames, auto layout, constraints and an 8-point spacing system
- Create reusable components with variants, component properties and a small design system using variables
- Prototype a multi-screen flow with interactions, Smart Animate and overlays, and share it for review
- Hand off designs in Dev Mode and speed up work with Figma AI, Figma Make and plugins

What Figma is and why every product and data team uses it

Figma is a browser-based design tool where many people work on the same file at the same time. Product managers sketch flows, designers build interfaces and design systems, developers inspect measurements and code, and data analysts design dashboards before building them in Power BI or Tableau.

Figma products

Figma Design

Screens, components, design systems and prototypes

FigJam

Online whiteboard: workshops, user flows, retros

Dev Mode

Developers inspect sizes, colours and code snippets

Figma Slides

Presentations built with Figma's tools

Figma Make

Describe an app in words; get a working prototype

Figma Sites

Design and publish a website from Figma

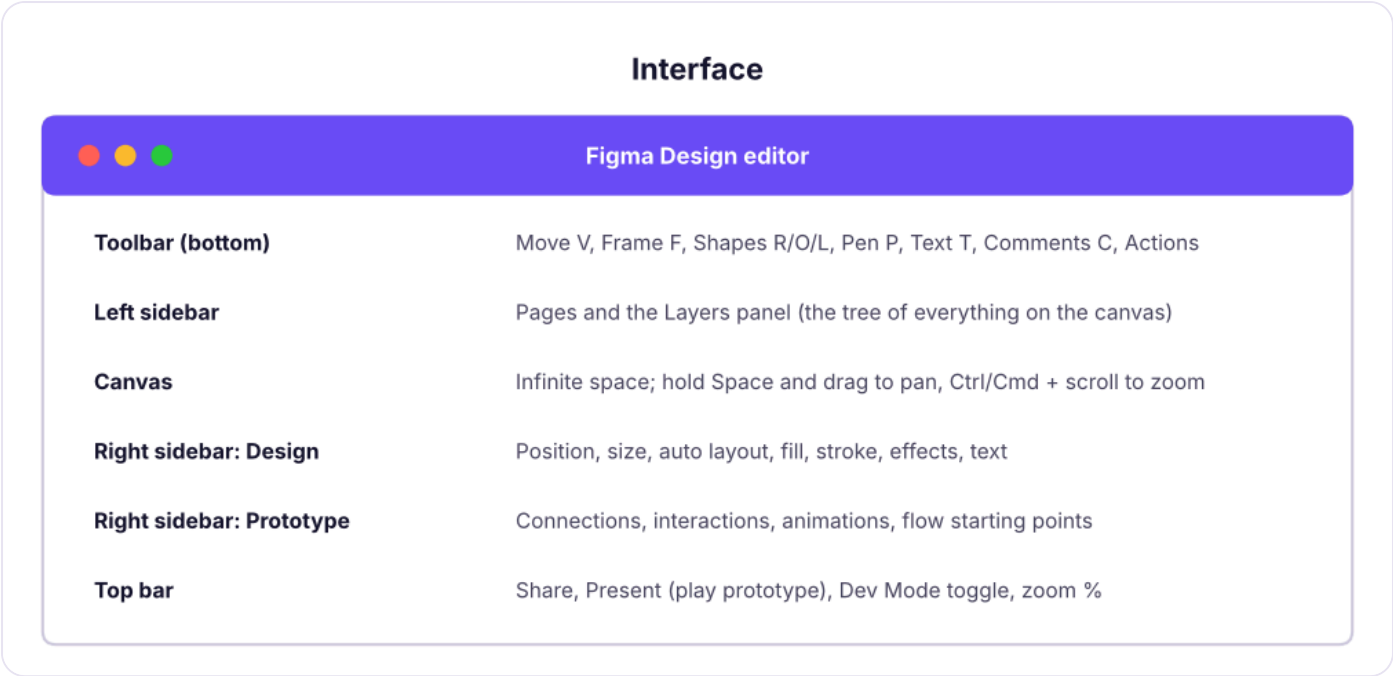
The Figma family. One account, one file format, different products for different jobs.

Plan (2026)	Who it is for
Starter (free)	Learning: a limited number of Figma Design and FigJam files, unlimited personal drafts, free viewers
Professional	Freelancers and small teams: unlimited files, team libraries, Dev Mode
Organization / Enterprise	Companies: design-system analytics, branching, SSO, admin controls
Education (free)	Verified students and teachers get Professional features

TIP Start free

Sign up at figma.com with Google, then install the desktop app (optional; the browser works the same). Students should apply for the free Education plan.

The interface



The Figma Design editor.

Essential shortcuts:

Action	Windows	Mac
Frame / Rectangle / Ellipse / Text	F / R / O / T	F / R / O / T
Duplicate	Ctrl + D	Cmd + D
Group / Ungroup	Ctrl + G / Ctrl + Shift + G	Cmd + G / Cmd + Shift + G
Add auto layout	Shift + A	Shift + A
Create component	Ctrl + Alt + K	Cmd + Option + K
Show measurements	hold Alt and hover	hold Option and hover
Zoom to fit / to selection	Shift + 1 / Shift + 2	Shift + 1 / Shift + 2
Quick actions (search any command)	Ctrl + /	Cmd + /

Frames, shapes and text

- A **frame** (F) is a screen or a container. Pick a preset in the right panel (iPhone 16, Android Compact, Desktop 1440). Frames can be nested and can clip content.
- A **group** (Ctrl+G) only bundles layers; it has no layout of its own. Prefer frames with auto layout.

- **Shapes** (R, O, L, P) take fills (solid, gradient, image), strokes, corner radius and effects (drop shadow, blur).
- **Text** (T): set font, size, line height and letter spacing in the Design panel; use *Auto width* for labels and *Fixed width* for paragraphs.

The 8-point grid

Use spacing and sizes in multiples of 8 (4 for small gaps): 8, 16, 24, 32, 48, 64. Add a layout grid to a frame (Design panel → Layout grid → 8 px grid or columns: 4 columns for mobile, 12 for desktop) so everything lines up.

Python

```
# A type scale (ratio 1.25 from 16 px) and an 8-point spacing scale: the numbers you put
into text styles and auto layout gaps
base, ratio = 16, 1.25
type_scale = {name: round(base * ratio ** i) for i, name in enumerate(["Body", "H6", "H5",
"H4", "H3", "H2", "H1"])}
spacing = [4, 8, 12, 16, 24, 32, 48, 64]
print("Type scale (px):", type_scale)
print("Spacing tokens (px):", spacing)
```

OUTPUT

```
Type scale (px): {'Body': 16, 'H6': 20, 'H5': 25, 'H4': 31, 'H3': 39, 'H2': 49, 'H1':
61}
Spacing tokens (px): [4, 8, 12, 16, 24, 32, 48, 64]
```

Colour contrast (accessibility)

Text must be readable: WCAG asks for a contrast ratio of at least **4.5:1** for normal text and **3:1** for large text. Figma shows contrast in plugins such as *Stark* or *A11y – Color Contrast Checker*; this is the formula they use:

Python

```
def lum(hex_):
    r, g, b = (int(hex_[i:i + 2], 16) / 255 for i in (1, 3, 5))
    f = lambda c: c / 12.92 if c ≤ 0.03928 else ((c + 0.055) / 1.055) ** 2.4
    return 0.2126 * f(r) + 0.7152 * f(g) + 0.0722 * f(b)
def contrast(fg, bg):
    a, b = sorted([lum(fg), lum(bg)], reverse=True); return (a + 0.05) / (b + 0.05)
for fg, bg in [("FFFFFF", "#6C4CF5"), ("9CA3AF", "FFFFFF"), ("17142E", "F3F0FF"),
("FFFFFF", "#12B5A6")]:
    r = contrast(fg, bg); print(f"{fg} on {bg}: {r:4.2f}:1 →", "PASS normal text" if r ≥
4.5 else "PASS large text only" if r ≥ 3 else "FAIL")
```

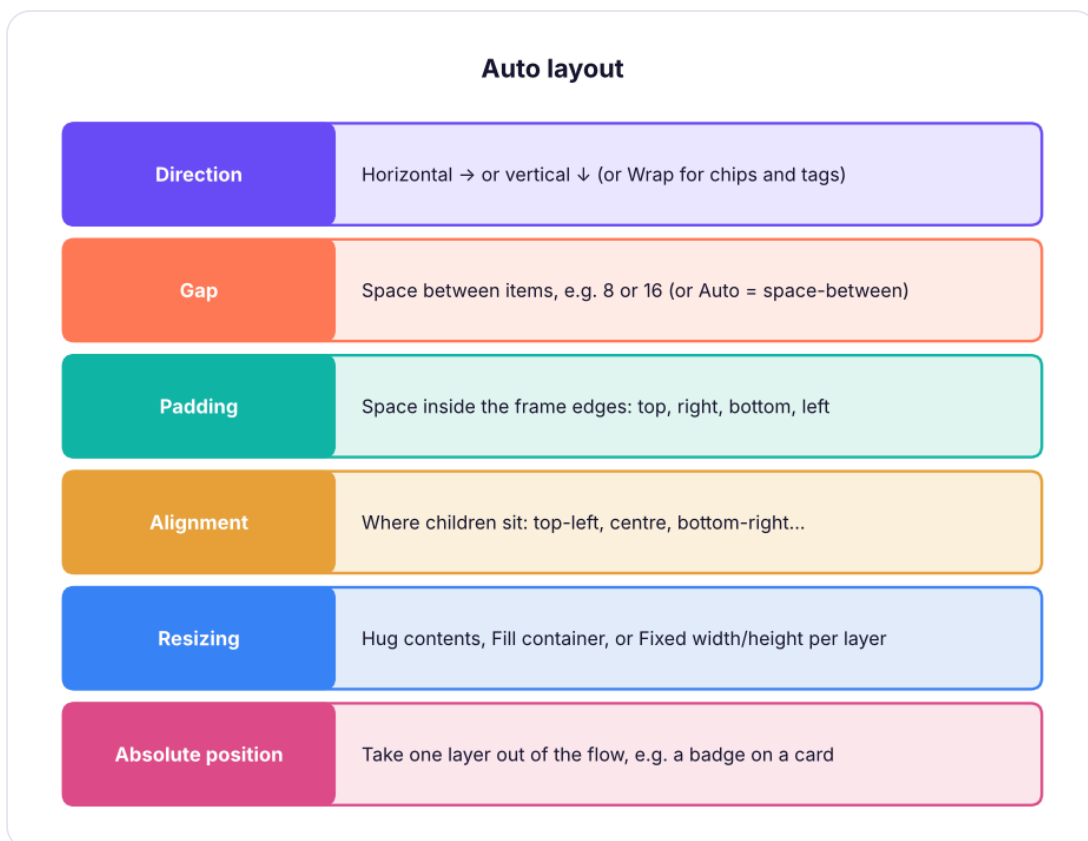
OUTPUT

```
#FFFFFF on #6C4CF5: 5.26:1 → PASS normal text
#9CA3AF on #FFFFFF: 2.54:1 → FAIL
#17142E on #F3F0FF: 15.92:1 → PASS normal text
#FFFFFF on #12B5A6: 2.57:1 → FAIL
```

Light grey text on white and white text on bright teal fail even the 3:1 large-text rule, so darken those colours (for example teal #0E7C72) until the checker passes.

Auto layout: responsive design

Auto layout (Shift + A) turns a frame into a flexbox: children stack horizontally or vertically with a fixed gap and padding, and the frame grows or shrinks with its content.



Auto layout settings and what they control.

STEP BY STEP**Build a responsive button**

1. Type "Get started" (T). Select it and press **Shift + A**: a frame wraps the text.
2. Set padding 12 (vertical) × 24 (horizontal), corner radius 8, fill your brand colour, text white.
3. Set the frame to **Hug** both ways: change the text to "Start your free trial" and the button grows.
4. Put three buttons in a vertical auto layout with gap 12 and set each to **Fill container**: they all stretch to the same width.

Constraints (for frames without auto layout) decide how a layer moves when its parent resizes: Left, Right, Left & Right (stretch), Centre, Scale. Use them for headers and floating buttons.

Components, variants and properties

A **component** (Ctrl + Alt + K) is a master; **instances** are copies that update when the master changes.

Feature	What it does	Example
Variants	Group related components into one set with properties	Button: Type = Primary/Secondary, State = Default/Hover/Disabled, Size = S/M/L
Boolean property	Show/hide a layer	"Has icon" on/off
Text property	Edit the label from the instance panel	Button label
Instance swap property	Swap a nested component	Choose the icon
Overrides	Change text, colour or images on one instance without detaching it	One red "Delete" button
Libraries	Publish components to the whole team (Assets panel)	Company design system

Python

```
from itertools import product
props = {"Type": ["Primary", "Secondary", "Ghost"], "State": ["Default", "Hover", "Disabled"], "Size": ["S", "M", "L"]}
variants = list(product(*props.values()))
print(len(variants), "variants in the Button set, e.g.", ", ".join("/".join(v) for v in variants[:3]))
print("Without variant properties you would manage", len(variants), "separate components.")
```

OUTPUT

```
27 variants in the Button set, e.g. Primary/Default/S, Primary/Default/M,
Primary/Default/L
Without variant properties you would manage 27 separate components.
```

Styles and variables (design tokens)

- **Styles:** saved colour, text, effect and grid styles you apply with one click.
- **Variables** (collections and modes): named values (colour, number, string, boolean) such as `color/bg/surface` or `space/16`. A collection can have **modes** like *Light* and *Dark* or *Brand A* and *Brand B*: switch a frame's mode and every bound colour changes.
- Bind spacing, radius and colours to variables so a redesign is one edit, and developers get the same token names in Dev Mode.

Prototyping

Switch to the **Prototype** tab:

1. Select a button, drag the blue handle to the target frame: this creates a **connection**.
2. Set the **trigger** (On click, On drag, While hovering, After delay, Key/Gamepad) and the **action** (Navigate to, Open overlay, Swap overlay, Back, Scroll to, Open link, Set variable).
3. Choose the **animation**: Instant, Dissolve, Move in/out, Push, Slide, or **Smart Animate** (matches layers with the same name across frames and animates the difference).
4. Mark a **flow starting point** and press **Present** (Ctrl + Alt + Enter) or share the prototype link.

Advanced: **interactive components** (a toggle that switches its own variant on click), **variables and conditionals** in prototypes (a counter, a cart total, "if logged in"), and **scroll behaviour** (fixed headers, horizontal carousels).

Collaboration, review and Dev Mode

- **Comments** (C) pinned to the canvas; **@mention** teammates; resolve when done.
- **Version history** (File → Show version history); **branching** on Organization plans to explore without touching the main file.
- **Dev Mode** (toggle in the top bar): developers see measurements, colours as variables, assets to export (SVG, PNG, PDF), CSS/iOS/Android snippets, "Ready for dev" statuses, and **Code Connect** links to real code components. Through the **Figma MCP server**, AI coding tools such as Claude Code can read selected frames and generate code that uses your variables.

FigJam for workshops

Sticky notes, shapes and connectors, stamps and voting, timers, templates (journey maps, retros, brainstorming, user-story maps), and **FigJam AI** that sorts and summarises stickies and generates diagrams from a prompt. Use it for discovery workshops, user flows and sprint retros.

Figma AI and Figma Make

Feature	What it does
First draft / Make designs	Generate a starting UI from a prompt using your design system
Rename layers, replace content, search by image	Clean-up and realistic placeholder content
Figma Make	Prompt-to-app: describe a product ("a habit tracker with streaks and a weekly chart") and get a working, clickable prototype with real code you can refine by chatting, publish, or bring into Figma Design
Plugins	Unsplash (photos), Iconify (icons), Content Reel, Stark (accessibility), Autoflow (flow arrows), Charts / Figma Chart for data visualisation mock-ups

2026 UPDATE

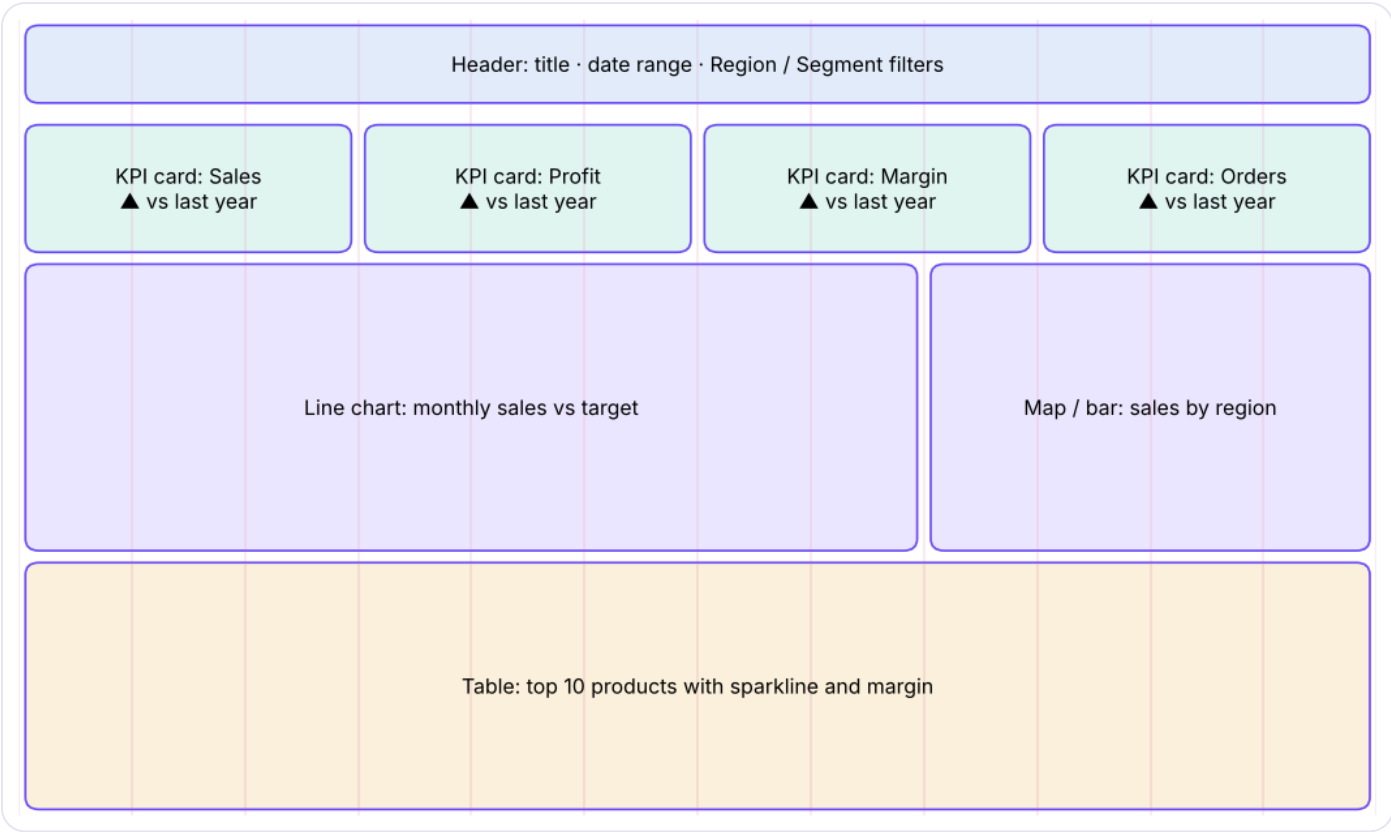
Good practice with AI in Figma

Use AI for a first draft or realistic content, then apply your components, variables and accessibility checks. Treat generated screens as sketches until a designer has reviewed spacing, hierarchy and states (empty, loading, error).

Project for data roles: design a KPI dashboard before building it

Designing the dashboard in Figma first saves hours in Power BI or Tableau:

1. Frame: Desktop 1440 × 1024, 12-column grid, 24 px gutters.
2. Header: title, date range picker, filters (Region, Segment) as components.
3. KPI cards: a component with variants (Up/Down/Neutral) and text properties for label, value and change.
4. Charts: placeholders or the *Chart* plugin for a line (trend), bar (ranking) and map; a table at the bottom.
5. Variables: Light and Dark modes; check contrast for every text colour.
6. Prototype: clicking a region filter swaps to a "South" variant of the page; add an overlay for a drill-through.
7. Share for comments with stakeholders, then rebuild in Power BI/Tableau with the agreed layout and colours (export the palette as hex codes or a Power BI theme JSON).



Wireframe of the KPI dashboard layout on a 12-column grid.



Summary

KEY TAKEAWAYS

- Frames + auto layout + an 8-point grid give clean, responsive screens.
- Components with variants and properties, plus variables with modes, form a design system.
- Prototype with connections, Smart Animate, overlays and variables; share links for review.
- Dev Mode, Code Connect and the MCP server connect design to code; FigJam runs workshops; Figma Make turns prompts into working prototypes.

Quick quiz

6 questions · tap an answer to check it

1. What does auto layout do?

- A** Stacks children with a set direction, gap and padding so the frame resizes with its content
- B** Exports code
- C** Adds a layout grid only
- D** Groups layers without any layout

Answer: A. It works like CSS flexbox.

2. You need Primary/Secondary buttons in three sizes and three states. What should you build?

A One component set with variant properties Type, Size and State

B 27 unrelated components

C Groups

D Separate files

Answer: A. Variants keep related components in one set.

3. Which feature lets one frame switch between Light and Dark colours?

A Variables with modes

B Constraints

C Smart Animate

D Comments

Answer: A. Bind colours to variables and switch the mode.

4. What is the minimum WCAG contrast for normal-size text?

A 4.5:1

B 2:1

C 3:1

D 7:1 only

Answer: A. 3:1 applies to large text.

5. Which animation matches layers with the same name across two frames?

A Smart Animate

B Dissolve

C Instant

D Push

Answer: A. Name layers consistently for it to work.

6. Where do developers inspect sizes, tokens and code snippets?

A Dev Mode

B FigJam

C Figma Slides

D The Pages panel

Answer: A. Dev Mode is built for handoff.

EXERCISE Practice project

Design a 3-screen mobile app for a food-delivery order flow (menu → cart → order placed) with: a Button component set (Type × State), a product-card component with an image, title, price and a "+ Add" button using auto layout, Light/Dark variables, a prototype with Smart Animate from cart to confirmation, and an overlay for "Apply coupon". Share the prototype link.

Show solution (worked answer)

Use 390 × 844 frames, a 4-column grid with 16 px margins, auto layout for cards (vertical, gap 8, padding 12), a component set with 6 variants, variables collection "Color" with Light/Dark modes bound to fills, connections On click → Navigate to (Smart Animate 300 ms ease-out), and an Open overlay action centred with a dark background. Check every text colour with a contrast plugin.

INTERVIEW CORNER Questions you may be asked

Q: How do you keep designs consistent across a large product? A shared library of components with variants and properties, styles and variables (tokens) with modes, documentation pages, and Dev Mode / Code Connect so code uses the same tokens.

Q: Auto layout vs constraints? Auto layout arranges children in a flow (direction, gap, padding, hug/fill), like flexbox; constraints pin a layer to its parent's edges when the parent is resized, for frames without auto layout.

Q: How do you use Figma as a PM or analyst? Wireframe flows and dashboards, run FigJam workshops, review prototypes with users before building, comment on designs, and prototype quickly with Figma Make.

Learn more (official and free)

- [Figma Help Center and courses](#)
- [Figma YouTube channel \(Figma for beginners series\)](#)
- [Figma Community files and templates](#)
- [Figma Make](#)
- [Figma Learn: Dev Mode and Code Connect](#)

n8n Complete Tutorial: Workflow Automation, Data Pipelines and AI Agents

Learn n8n, the open, self-hostable workflow-automation tool. Set it up on n8n Cloud or Docker, understand triggers, nodes, credentials, items and expressions, transform data with Edit Fields, IF, Switch, Merge, Loop and Code nodes, call any API with HTTP Request, handle errors and schedules, build AI agents with chat models, memory, tools and vector stores, and connect n8n to MCP. Four complete projects: lead capture, a daily sales report, an email-triage AI agent and a document Q&A bot.

✦ n8n 1.x (2026)

Beginner to Advanced

🕒 180 min

By the end of this lesson you can

- Install or sign up for n8n and build, test and activate a workflow
- Explain triggers, nodes, items and expressions, and transform data between apps
- Call REST APIs with HTTP Request and write JavaScript or Python in the Code node
- Add error handling, retries, schedules and webhooks for reliable automations
- Build AI-agent workflows with chat models, memory, tools, vector stores and MCP

What n8n is

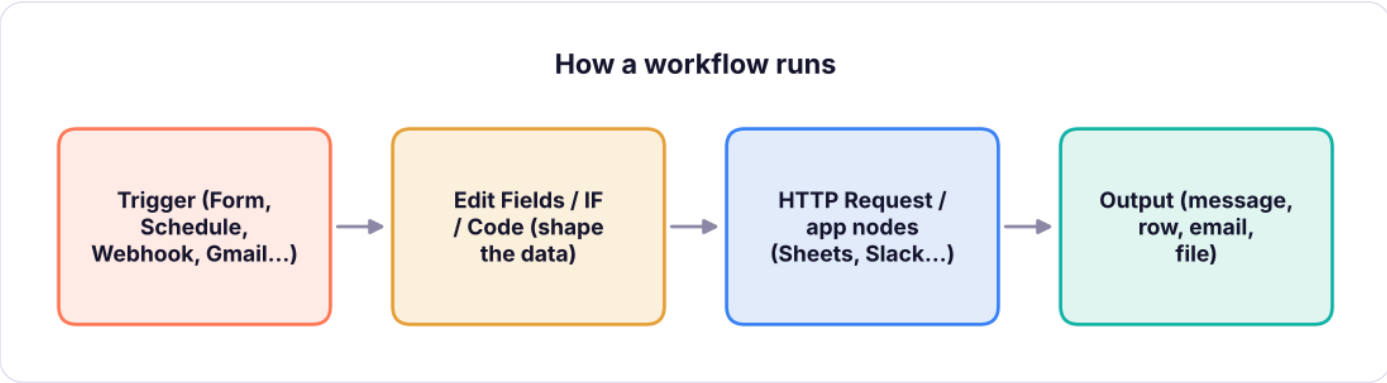
n8n ("n-eight-n", from *nodemation*) connects apps and data with visual workflows. Each workflow starts with a **trigger** (a form is submitted, a schedule fires, a webhook is called) and passes data through **nodes** that read, transform and send it. Compared with Zapier and Make, n8n can be **self-hosted** (your data stays on your server), lets you write **code** in any step, and has strong **AI-agent** nodes.

	Zapier	Make	n8n
Hosting	Cloud only	Cloud only	Cloud or self-hosted (Docker, npm)
Pricing unit	Tasks (each step)	Operations	Workflow executions (cloud); free to self-host under its fair-code licence
Code	Code steps (limited)	Limited	Full JavaScript/Python Code node, npm packages when self-hosted
Best for	Quick app-to-app zaps	Visual scenarios with routers	Data pipelines, AI agents, privacy-sensitive automations

Getting n8n

Option	How
n8n Cloud	Sign up at n8n.io → free trial → you get <code>yourname.app.n8n.cloud</code>
Docker (recommended self-host)	<code>docker volume create n8n_data</code> then <code>docker run -it --rm -p 5678:5678 -v n8n_data:/home/node/.n8n docker.n8n.io/n8nio/n8n</code> and open <code>http://localhost:5678</code>
npm (quick local test)	<code>npm install n8n</code> (needs Node.js)
Production self-host	Docker Compose with PostgreSQL, a domain with HTTPS (Caddy/Nginx), <code>N8N_ENCRYPTION_KEY</code> set and backed up, queue mode with Redis for scale

The editor and the core ideas



An n8n workflow: a trigger, then nodes that pass items from left to right.

Concept	Meaning
Workflow	The canvas of connected nodes; <i>Active</i> workflows run on their trigger
Trigger node	Starts the run: Manual, Schedule, Webhook, n8n Form, Chat, app triggers (Gmail, Google Sheets, Typeform, Stripe...)
Node	One step: an app action, a core utility, or an AI node
Credentials	Saved, encrypted logins/API keys (OAuth for Google, Slack, etc.), reused by nodes
Items	Data flows as a list of JSON items; most nodes run once per item
Expressions	<code>{{ \$json.email }}</code> , <code>{{ \$now.toFormat('yyyy-MM-dd') }}</code> , <code>{{ \$('Form').item.json.name }}</code> to use data from earlier nodes
Executions	History of every run with the input/output of each node, for debugging and re-running

Items: the most important idea

Every node receives and outputs an **array of items**, each `{ "json": { ... } }`. If a Google Sheets node returns 25 rows, the next Slack node runs **25 times**, once per item, unless you aggregate first.

Python

```
import json
items = [{"json": {"name": "Asha", "email": "asha@example.com", "city": "Pune", "score": 82}},
         {"json": {"name": "Ravi", "email": "ravi@example.com", "city": "Delhi", "score": 45}},
         {"json": {"name": "Meera", "email": "meera@example.com", "city": "Pune", "score": 91}}]
# IF node: score ≥ 60 goes to the "true" branch
true_branch = [i for i in items if i["json"]["score"] ≥ 60]
# Edit Fields (Set) node: keep and add fields
shaped = [{"json": {"lead": i["json"]["name"], "email": i["json"]["email"], "priority": "hot" if i["json"]["score"] ≥ 90 else "warm"}} for i in true_branch]
# Aggregate node: many items → one item with a list
aggregated = {"json": {"leads": [s["json"]["lead"] for s in shaped], "count": len(shaped)}}
print(json.dumps(shaped, indent=1)); print(aggregated)
```

OUTPUT

```
[
  {
    "json": {
      "lead": "Asha",
      "email": "asha@example.com",
      "priority": "warm"
    }
  }
]
```

```

},
{
  "json": {
    "lead": "Meera",
    "email": "meera@example.com",
    "priority": "hot"
  }
}
]
{'json': {'leads': ['Asha', 'Meera'], 'count': 2}}

```

Core nodes you will use every day

Node	Use
Edit Fields (Set)	Rename, add, remove or compute fields
IF / Switch	Route items by conditions (two branches / many branches)
Merge	Combine two inputs: append, match by a field (like a join), or by position
Filter	Drop items that do not match
Split Out	Turn a list inside one item into separate items
Aggregate / Summarize	Many items → one (list, sum, count, group by)
Loop Over Items (Split in Batches)	Process items in batches (APIs with rate limits)
Wait	Pause for a time or until a webhook/form resumes the run
HTTP Request	Call any REST API (GET/POST, headers, auth, pagination); import a cURL command
Code	JavaScript or Python over all items or each item
Date & Time, Crypto, XML/HTML, Extract From File, Convert to File	Utilities
Execute Workflow / Execute Workflow Trigger	Call one workflow from another (sub-workflows, reusable steps)
Respond to Webhook	Return a custom HTTP response (build simple APIs)

Code node example (JavaScript, "Run once for all items")

JAVASCRIPT

```
// Clean phone numbers and drop duplicates by email
const seen = new Set();
return $input.all()
  .filter(item => { const e = item.json.email.toLowerCase().trim(); if (seen.has(e)) return false; seen.add(e); return true; })
  .map(item => ({ json: { ...item.json, phone: String(item.json.phone || '').replace(/\D/g, '').slice(-10) } }));
```

The Python version of the same idea (Code node → Language: Python) uses `_input.all()` and returns a list of dicts with a `json` key.

HTTP Request: calling an API

Python

```
# What an HTTP Request node sends and how you map the answer back into items
request = {"method": "GET", "url": "https://api.example.com/v1/orders", "qs": {"status": "paid", "limit": 2},
          "headers": {"Authorization": "Bearer {{ $credentials.apiKey }}"}}
response = {"data": [{"id": 101, "amount": 2499, "city": "Pune"}, {"id": 102, "amount": 899, "city": "Delhi"}], "next": None}
items = [{"json": o} for o in response["data"]] # Split Out on "data"
print(request["method"], request["url"], request["qs"]); print(items)
```

OUTPUT

```
GET https://api.example.com/v1/orders {'status': 'paid', 'limit': 2}
[{'json': {'id': 101, 'amount': 2499, 'city': 'Pune'}}, {'json': {'id': 102, 'amount': 899, 'city': 'Delhi'}}]
```

Tips: use **Authentication** → **Predefined credential type** when n8n has the app, otherwise **Generic credential** (header auth, OAuth2); turn on **Pagination** for APIs with `next` links; add **Retry on fail** (Settings) for flaky APIs.

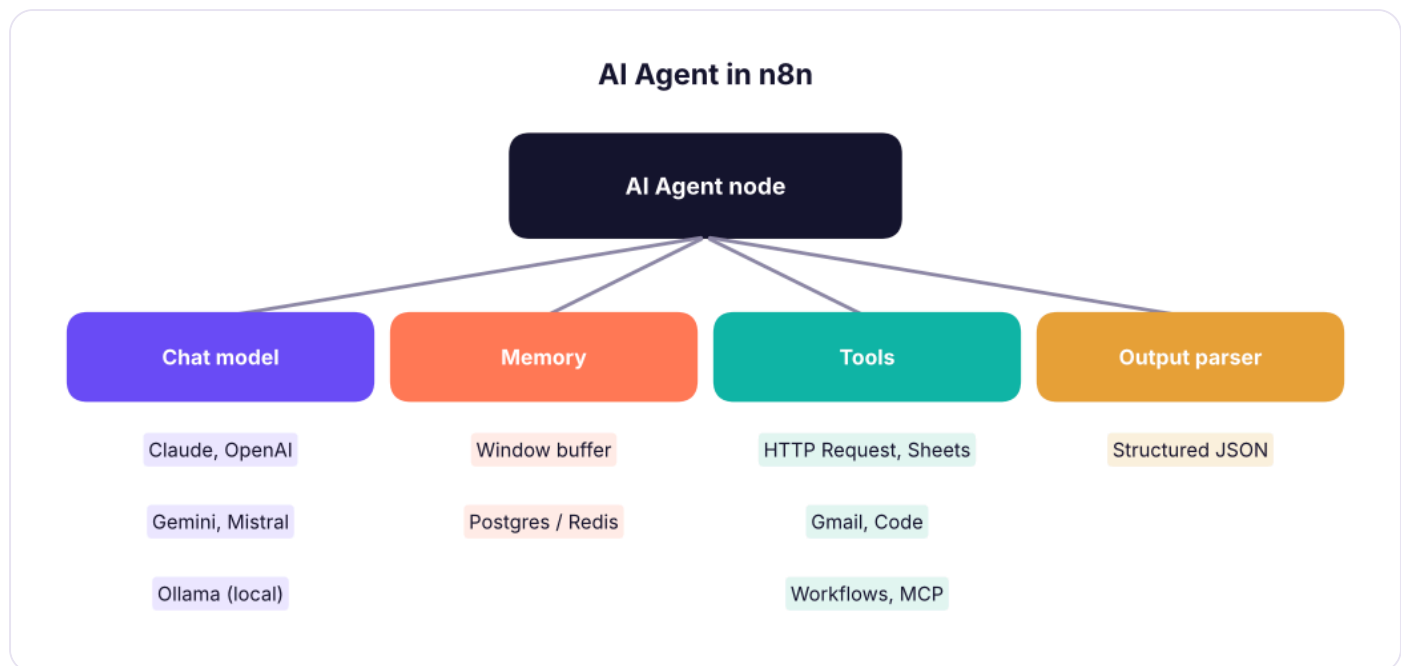
Reliability: errors, retries and scheduling

- **Settings per node:** *Retry on fail* (tries and wait), *Continue on fail* / error output branch, *Always output data*.
- **Error workflow:** Workflow settings → Error workflow → pick a workflow that starts with **Error Trigger** and sends you a Slack/WhatsApp/email alert with the failed node and message.

- **Schedule Trigger:** every X minutes, daily at 09:00 IST (set the workflow time zone to Asia/Kolkata), or a cron expression `0 9 * * 1-5`.
- **Pin data** while building: freeze a node's output so you do not call the API on every test.
- **Environment safety:** keep secrets in credentials, never in Code nodes; restrict who can edit workflows; use separate dev and prod instances for important flows.

AI agents in n8n

The **AI Agent** node runs a loop: a chat model reads the goal, decides which **tool** to call, reads the result and continues until it can answer.



An n8n AI agent and the sub-nodes plugged into it.

Other AI building blocks: **Basic LLM Chain**, **Text Classifier**, **Information Extractor** (pull fields from emails or PDFs into JSON), **Summarization Chain**, **Sentiment Analysis**, **Embeddings + Vector Store** nodes (Pinecone, Qdrant, Supabase, PGVector, in-memory) for RAG, and the **Chat Trigger** for a hosted chat window.

n8n and MCP

- **MCP Client Tool:** lets an n8n agent use tools from any MCP server.
- **MCP Server Trigger:** exposes your n8n workflows as MCP tools, so Claude or other MCP clients can call them (for example "create a lead in our CRM").

Four complete projects

1. Lead capture to Google Sheets and Slack

n8n Form Trigger (name, email, phone, course) → Edit Fields (clean phone, add timestamp) → Google Sheets: Append row → IF course = "Data Engineer" → Slack: send message to #sales (true) / Gmail: send brochure (false).

2. Daily sales report

Schedule Trigger 09:00 IST → Postgres/HTTP Request: yesterday's orders → Summarize: sum sales by region → Code: build an HTML table → Gmail: send to managers + Slack summary.

Python

```
import pandas as pd
orders = pd.DataFrame({"region": ["South", "West", "South", "North", "East", "West"],
"amount": [24990, 8999, 15500, 4300, 12000, 30500]})
summary = orders.groupby("region").amount.agg(["count", "sum"]).sort_values("sum",
ascending=False)
html = summary.to_html() # the Code node builds the same table for the email
body
print(summary); print("HTML length for the email:", len(html), "characters")
```

OUTPUT

	count	sum
region		
South	2	40490
West	2	39499
East	1	12000
North	1	4300

HTML length for the email: 571 characters

3. Email triage AI agent

Gmail Trigger (new email) → Text Classifier (Sales lead / Support / Invoice / Spam) → Switch → Sales: Information Extractor (name, company, budget) → HubSpot/Sheets + Slack ping; Support: AI Agent drafts a reply using a **Vector Store tool** over your FAQ and saves it as a **Gmail draft** for a human to send; Spam: archive.

4. Document Q&A bot (RAG)

Indexing workflow: Google Drive Trigger (new PDF) → Default Data Loader + Recursive Text Splitter → Embeddings → Vector Store: insert . Chat workflow: Chat Trigger → AI Agent with the vector store as a tool, a chat model and window memory → answer with sources. This is the RAG pattern from the GenAI track, built without code.

Summary

KEY TAKEAWAYS

- Workflows = trigger + nodes; data moves as a list of JSON items and most nodes run once per item.
- Expressions pull values from earlier nodes; Code and HTTP Request cover anything without a ready node.
- Reliability comes from retries, error branches, an error workflow, schedules with the right time zone and pinned test data.
- The AI Agent node combines a chat model, memory and tools; vector stores add RAG; MCP connects n8n to other AI tools both ways.

Quick quiz

6 questions · tap an answer to check it

1. A Google Sheets node returns 30 rows and a Slack node follows. What happens?

A The Slack node runs 30 times, once per item, unless you aggregate first

B It runs once with all rows

C It fails

D It runs twice

Answer: A. Most nodes execute per item.

2. Which expression reads the email field of the current item?

A {{ \$json.email }}

B \$email

C =email

D {email}

Answer: A. \$json is the current item's data.

3. Which node calls any REST API that has no ready-made n8n node?

A HTTP Request

B Merge

C Wait

D Filter

Answer: A. You can also import a cURL command into it.

4. How do you get alerted when a production workflow fails?

A Set an Error workflow that starts with Error Trigger and sends a message

B Check executions every morning

C Use Pin data

D Disable retries

Answer: A. The error workflow runs whenever another workflow fails.

5. What is the main reason teams self-host n8n?

A Data stays on their own servers and code/npm packages are allowed

B It has no editor

C It only works offline

D Self-hosting is required

Answer: A. Cloud is easier; self-hosting gives control.

6. Which sub-nodes plug into an AI Agent node?

A A chat model, memory, tools and an optional output parser

B Only a Schedule Trigger

C Only a Merge node

D Only credentials

Answer: A. Tools can include HTTP, Sheets, other workflows and MCP.

EXERCISE Build it

Build project 1 (form → Sheets → Slack/Gmail) and add an error workflow that messages you on failure. Then extend it: use an Information Extractor to pull "experience in years" from a free-text field and route experienced leads to a different Slack channel.

Show solution (worked answer)

Form Trigger fields: name, email, phone, course, about_you. Edit Fields: `phone = {{ $json.phone.replace(/\\D/g, '').slice(-10) }}`, `received = {{ $now.toISO() }}`. Information Extractor schema `{ "experience_years": number }` from `about_you`, then IF `experience_years ≥ 3 → #senior-leads else #leads`. Workflow settings → Error workflow → a workflow with Error Trigger → Slack "Workflow {{ \$json.workflow.name }} failed at {{ \$json.execution.lastNodeExecuted }}".

INTERVIEW CORNER Questions you may be asked

Q: n8n vs Zapier vs Make? Zapier is the simplest for app-to-app automations; Make adds visual branching and data mapping; n8n adds self-hosting, full code steps, sub-workflows and strong AI-agent features, which suits data and AI teams.

Q: How do you make an automation reliable? Retries with backoff, error branches and an error workflow with alerts, idempotent steps (check before creating), rate-limit batching, logging executions, separate test and production, and secrets in credentials.

Q: How would you build a support bot on company documents in n8n? An indexing workflow that loads documents, splits them, embeds them and stores them in a vector store; a chat workflow with Chat Trigger, an AI Agent with a chat model, memory and the vector store as a retrieval tool, with sources in answers and a human-handover path.

Learn more (official and free)

- [n8n documentation](#)
- [n8n free courses, Level 1 and Level 2](#)
- [n8n workflow templates \(thousands of ready examples\)](#)
- [n8n community forum](#)
- [Advanced AI in n8n \(agents, RAG, MCP\)](#)