Master 95% of Claude Code (as a beginner)

This guide summarizes the key concepts, the WAT framework, and the steps for building, testing, and deploying AI automations using Claude Code.

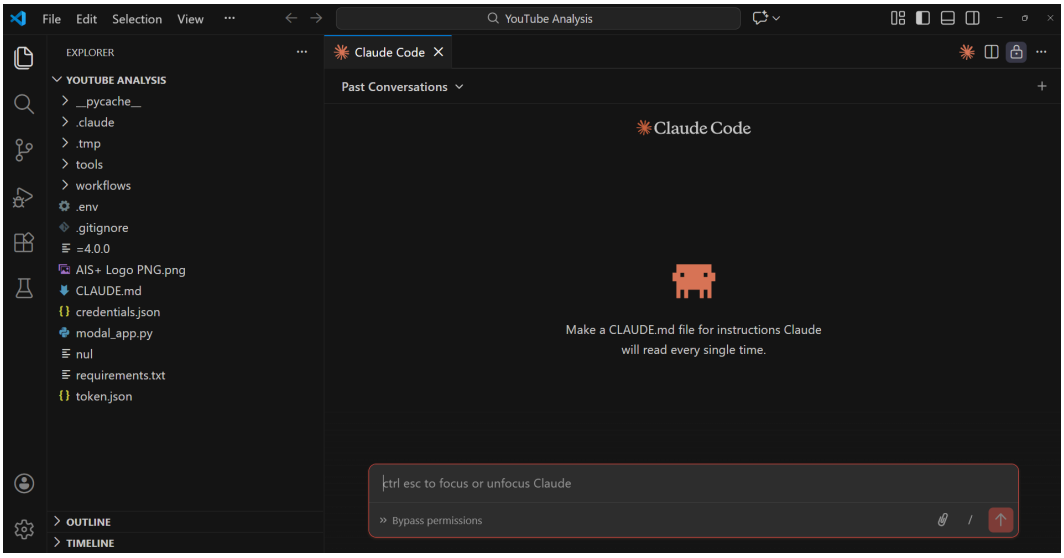
By: Nate Herk

---

1. Interface & Setup

Claude Code allows you to build complex coding projects and automations inside your local development environment (IDE).

- **The Environment:** The tutorial uses **Visual Studio Code (VS Code)**.
- **The Extension:** Search for and install the "Claude Code" extension in the VS Code marketplace.
- **Access Requirements:** - Requires a paid Anthropic plan (Claude Pro or Team).
    - You must sign in with your Anthropic account within the extension.
- **Layout:**
    - **Left Side (Explorer):** This is where your file structure lives (Workflows, Tools, Prompts).
    - **Right Side (Agent):** The chat interface where you interact with Claude Code to plan and execute tasks.
- **Bypass Permissions:** For faster builds, you can enable "Bypass Permissions" in the extension settings. This allows the agent to edit files without asking for approval on every single step.



2. The WAT Framework

To build reliable AI automations, Nate uses a three-layer structure called the **WAT Framework**. This separates probabilistic reasoning (AI) from deterministic execution (Code).

Layer 1: Workflows (/workflows)

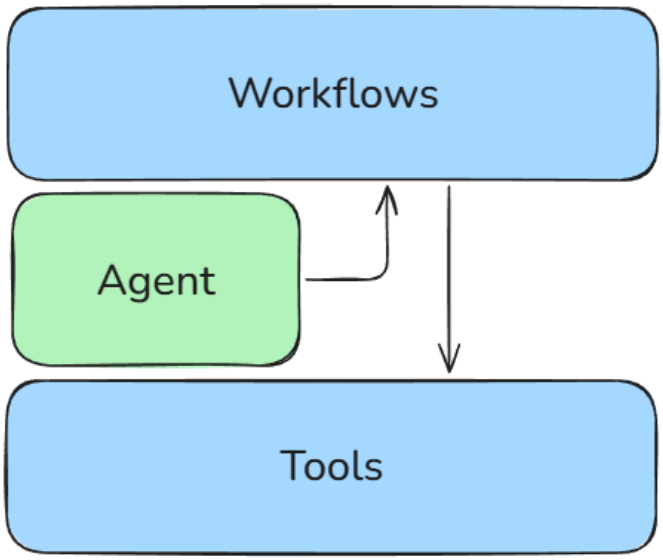
- **Format:** .md (Markdown) files.
- **Purpose:** These are the SOPs (Standard Operating Procedures). They define the objective, required inputs, tool sequences, and how to handle edge cases in plain English.
- **Analogy:** This is the "manager" telling the worker exactly what steps to take.

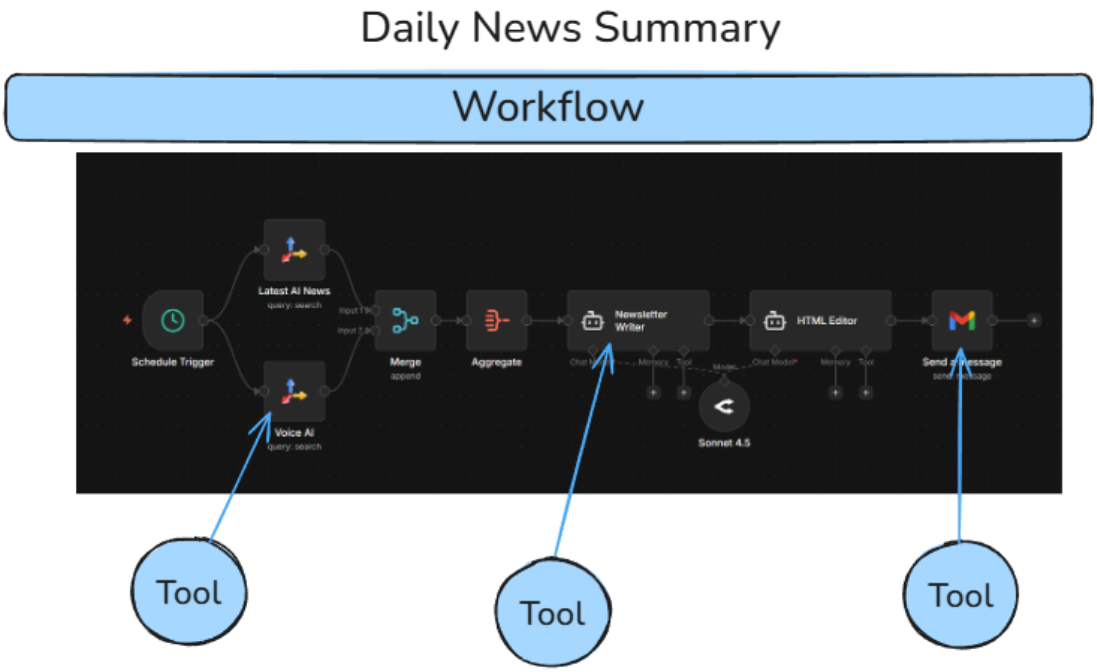
Layer 2: Agent (claude.md)

- **Format:** .md (Markdown) system prompt.
- **Purpose:** This is the core instruction set for Claude Code. It tells the agent how to navigate the folders, which tools to use, and how to follow the WAT framework.
- **Self-Healing:** The agent is instructed to read errors, refactor tools, and update workflows if a process fails.

Layer 3: Tools (/tools)

- **Format:** .py (Python) files.
- **Purpose:** Actual code that executes actions (e.g., scraping a site, sending an email, querying a database).
- **Security:** API keys and secrets are **never** stored in these files. They are kept in a .env file to prevent exposure.





3. Planning & Building the Automation

Before writing a single line of code, you must move into **Plan Mode** in the Claude Code interface.

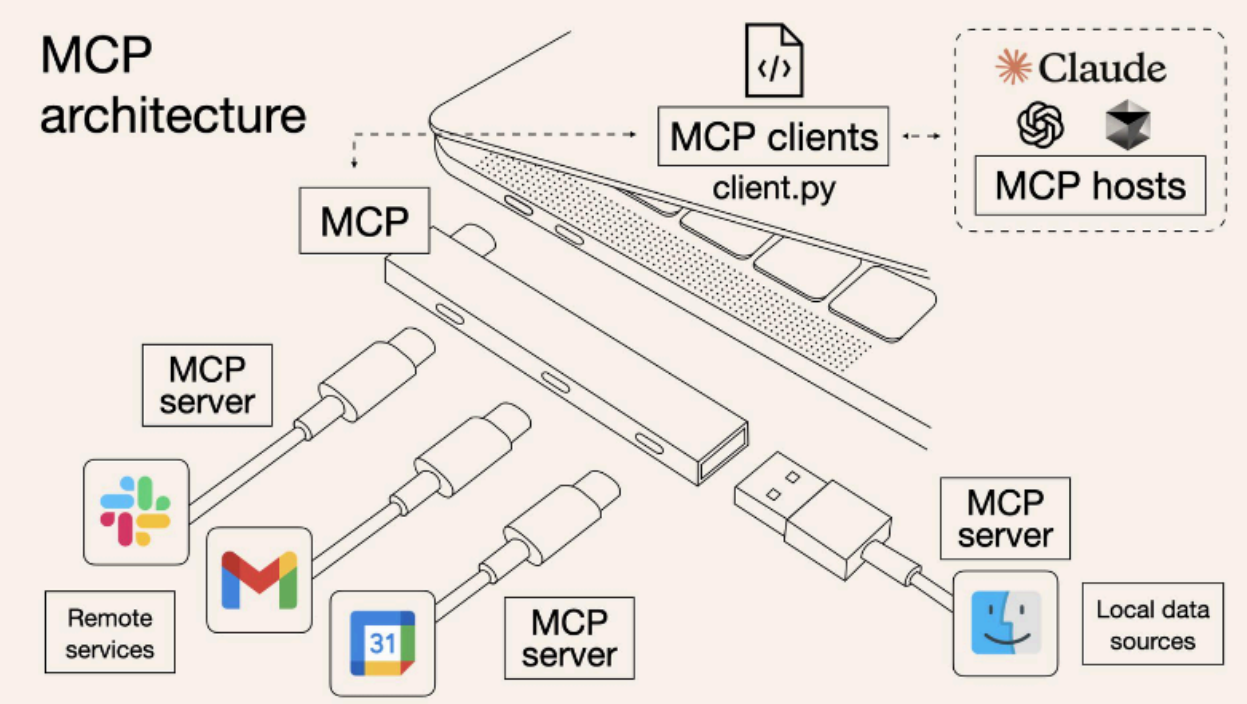
1. **The Brain Dump:** Describe your goal clearly (e.g., "Scrape YouTube channels in the AI niche and create a branded slide deck").
2. **Iterative Questioning:** In Plan Mode, the agent will ask clarifying questions about frequency, data points, and delivery methods.
3. **To-Do List:** Once the plan is accepted, the agent creates a checklist and executes it step-by-step (creating folders, writing Python scripts, and setting up workflows).

4. Superpowers: MCPs & Skills

Claude Code can be extended with external capabilities to handle tasks it can't do natively.

- **MCP (Model Context Protocol) Servers:** Think of this as an "App Store" for AI. It provides a universal port to connect to services like Gmail, Google Calendar, or Slack without writing individual API integrations for every tiny task.
- **Skills:** These are dynamic, reusable instructions or custom prompts (e.g., a "Canvas Design" skill) that Claude loads only when needed.
    - **Local Skills:** Installed for a specific project.

- **Global Skills:** Installed across your entire Claude Code instance for use in any project.



Skills vs...?

Skills vs. Projects

Projects provide static background knowledge that's always loaded when you start chats within them. Skills provide specialized procedures that activate dynamically when needed and work everywhere across Claude.

Skills vs. MCP (Model Context Protocol)

MCP connects Claude to external services and data sources. Skills provide procedural knowledge—instructions for how to complete specific tasks or workflows. You can use both together: MCP connections give Claude access to tools, while Skills teach Claude how to use those tools effectively.

5. Testing & Optimization

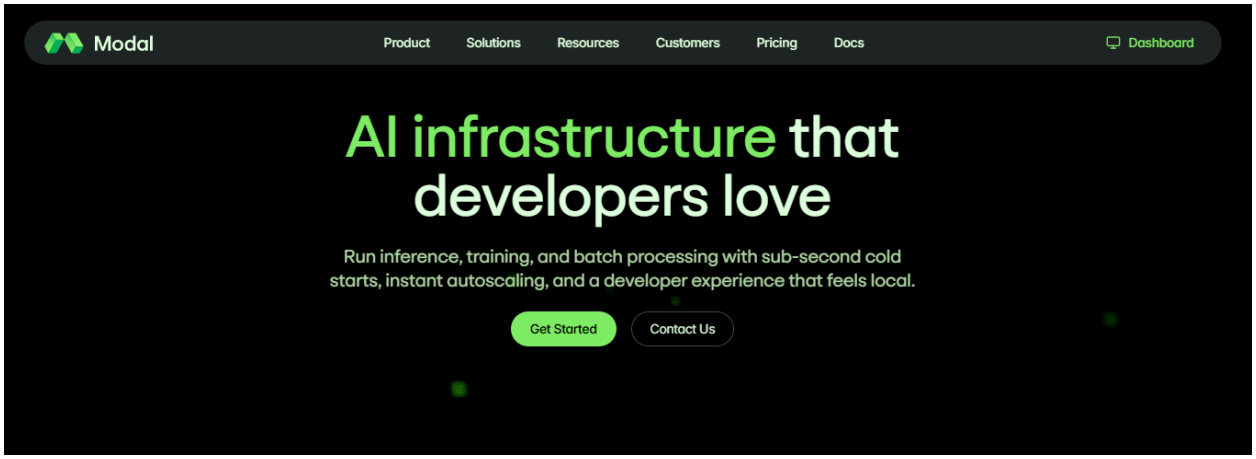
Building the automation is only half the battle. You must test and refine the logic.

- **Initial Run:** Execute the workflow in a test environment to identify missing dependencies or API errors.
- **Error Resolution:** If a tool fails, copy the terminal error and paste it back into the Claude Code chat. The agent will analyze the log, fix the Python script, and re-run the test.
- **Branding & Assets:** You can drag and drop assets (like logos or images) into your project folder and instruct the agent to incorporate them into the final deliverables (PDFs, Slide Decks, etc.).

6. Deploying to Production (Modal)

Once the automation works locally, you need to host it in the cloud so it can run automatically on a schedule (Cron) or via a trigger (Webhook).

- **Hosting Platform:** The tutorial uses **Modal**, a serverless infrastructure for Python.
- **Benefits:** You only pay for the seconds the code is actually running. It handles the "computers in the cloud" for you.
- **Deployment Steps:**
    1. Install the Modal client via Claude Code.
    2. Instruct the agent: "Push this workflow to Modal to run every Monday at 6 AM."
    3. **Security Review:** Always ask the agent to perform a security review before deploying to ensure no API keys or vulnerabilities are exposed.
- **Monitoring:** Use the Modal dashboard to check logs and track execution history. If a cloud run fails, paste the Modal logs back into Claude Code to fix the deployment.



Want to connect with others building and monetizing AI automation?

Become an AIS Plus Member