BUILD IT YOURSELF · 2026

The Operator's Field Notes.

How the work gets made, and how to build it yourself. One connected pipeline I built with Claude, written up so you can replicate the method.

Conner Crowe

Fractional marketing lead.
Home, furniture, and decor brands on Shopify.

What is in here.

This is the method behind a connected pipeline I built with Claude, and the instructions to build your own. Not a sales deck. The kitchen, with the door open and the recipes on the counter.

PART ONE

The method: how to think about it

01 **Why a pipeline, not automation.** What the word means, and the trade that makes it worth doing.

02 **The anatomy of a stage.** A defined task has three parts. The upfront cost, and who pays it.

03 **The reusable core and the standard.** How one component serves every stage, and why each stage is scored.

PART TWO

Build it yourself: the useful half

04 **The setup and the skill.** The tools you need, and the anatomy of a skill folder.

05 **The file structure.** How to lay a project out, with the knowledge layer.

06 **The prompts.** The real, generalized prompts for the brand constitution, the image schema, and the QA check.

07 **A worked build.** The four-stage pipeline, built end to end, step by step.

PART THREE

Closing: what still needs you

HOW TO READ IT

Part one is the thinking, about fifteen minutes. Part two is the build. If you want to replicate the approach, that is the half to keep open while you work. No email required to have read any of it.

Why a pipeline, not automation.

A home brand needed its whole catalog shot. Styled rooms, every SKU. The usual route is a photographer, a studio, and weeks of scheduling. I sent the catalog back styled and shot. There was no shoot day.

That is one pipeline, built with Claude over about two weeks. It runs four connected stages: it generates the styled image, builds the ad from it, pushes the result to the store, and rewrites the SEO product fields to match. Before I show you how to build your own, I want to be precise about what the word means, because the marketing world has worn it out.

A pipeline is a sequence of defined stages that produces one outcome. Here the outcome is a new product taken from a plain photo to a finished page and a running ad. Each stage is built once, then run when the work comes in. The build is slow. The run is fast. That gap is the whole idea.

AUTOMATION IS NOT THE SAME THING

When a founder hears "I built systems," the picture is usually a tool that runs untouched while everyone sleeps. That is automation, and it is not what this is. Automation removes the person. A pipeline keeps the person and removes the *busywork around the person*.

The difference matters because it tells you what gets faster and what does not. Setup, sequencing, formatting, the handoff between one step and the next, the third round of a thing I have already done forty times. That is what a pipeline absorbs. The judgment call about whether the output is any good still sits with me. Part three is about that line, because it is the line that decides what a senior operator is for.

BUILDING IN THE OPEN

I built this pipeline with Claude over two weeks, and I have been writing it up as I go. The method is the interesting part, so I am not keeping it behind a pitch. Part two of this document hands you the file structure and the prompts. Read it, then build your own.

Most marketing process stays behind a closed door. The pitch is the deck and the case study. How the work gets made does not get shown. This document is the opposite. You can read exactly how the food gets made, and the back of the book is the recipe card.

The anatomy of a stage.

A pipeline is only as good as the stages inside it. A stage is a defined task, and a defined task is not "make it look nice." A real one has three parts, and if any is missing the pipeline drifts.

PART ONE

The input

What goes in. A product photo. A brand voice doc. A keyword. The pipeline states what it needs, so nobody is guessing.

PART TWO

The standard

What "done right" looks like, written down before the work starts. Not a vibe. A rule the output is checked against.

PART THREE

The failure condition

What "wrong" looks like, named in advance. The drawer pull is the wrong shape. The copy uses a banned word. Caught, not shipped.

A stage defined this way can be handed to anything. A junior person. A script. A model. The handoff is clean because the stage does not depend on someone remembering how it is supposed to go. That memory now lives in the pipeline. In part two you will see where it lives literally: a folder, a file, a written instruction.

THE UPFRONT TEACHING COST

Writing a stage down at that level of detail is slow, and it never feels like progress the first time. It feels like describing a job instead of doing it. Most of the time this cost stays hidden inside one person's head, paid quietly on every project. The cost is real either way. The difference is whether you pay it once or pay it forever.

I pay it once. The image-generation stage took days to specify. The brand constitution, the prompt schema, the fidelity rule, the failure list. Then it ran across a full catalog in an afternoon. The second brand it retargets to costs hours, not days, because the structure already exists and only the brand facts change.

THE TRADE

A pipeline front-loads the thinking. The work is in defining each stage so well that running it is almost boring. Boring is the goal. Boring is repeatable, and repeatable is what you are actually buying.

This is also why a pipeline is not fragile in the way a clever one-off is. The standard and the failure condition are written down. When something slips, I do not re-derive the fix from memory. I read the rule, see which part the output broke, and tighten it. The pipeline gets harder to break every time it is run.

The reusable core, and the standard.

When I built this pipeline I did not write four stages from scratch. I built one stage well, noticed a part of it would be useful in the others, pulled that part out, and wired it back in. That move is how four stages share one core instead of repeating it.

THE REUSABLE CORE, IN FOUR MOVES

01 **Build one stage.** Solve a single real problem for the brand. Do not generalize yet. Generalizing too early produces a vague tool that fits nothing well.

02 **Find the reusable part.** After it works, look again. One piece is almost always doing a job the rest of the pipeline also needs. The image generator inside the ad stage does not care that it is an ad.

03 **Pull it out.** Lift that piece into its own component with its own clean input and standard. Now it is a shared core, not a buried step.

04 **Wire it back.** Point the original stage at the extracted core, then point the other stages that need it at the same core. One improvement lands across the pipeline at once.

A concrete pass through the move. The image-generation stage and the ad stage both need a render. So the image core lives once, as its own component, and both stages call it. Fix the core and both stages get the fix. One extraction, the whole pipeline sharper.

THE STANDARD, OR IT IS JUST FAST

Speed without a standard is a way to produce bad work quickly. Every stage I run is scored against something specific. The standard is the part of the stage that says what "done right" means, concrete enough that an output can pass or fail it without my opinion in the room.

WEAK STANDARD	REAL STANDARD
"Make it look on-brand."	**"Scored against the real product photo."**
Nothing to check against. Every output passes, because passing is a matter of taste. Drift is invisible until a customer sees it.	The render is compared to the actual product. Wrong material, wrong proportion, wrong hardware. It fails before it ships.

For the image stage the standard is product fidelity, checked against a reference photo. For the SEO stage it is a fixed title and meta-description pattern, checked field by field. The check is a gate, not a final polish. Nothing leaves a stage until it passes. In part two you will write one of these checks yourself.

The setup, and the skill.

This is where the document turns from method to manual. Everything from here is what you need to build a pipeline of your own. Start with the tools, then the unit a pipeline is made of.

WHAT YOU NEED TO START

01

Claude

The operator that runs the steps, holds the standard, and writes the prompts. I use it in a coding environment so it can read and write files in a project folder.

02

The skill system

A way to save a defined task as a folder Claude reads on demand. This is how a pipeline stops being a chat and becomes a thing you reuse.

03

An image model

For imagery pipelines, a current image model with an API key. Claude directs it from a prompt. Any capable model with a key works.

That is the whole kit. No servers, no orchestration platform, no subscriptions stacked on subscriptions. A project folder, Claude with access to it, and a key for the image model. The leverage is not in exotic tooling. It is in how carefully the task is written down.

WHAT A SKILL IS

A skill is a folder that holds one defined task. It carries a **SKILL.md** file, and any supporting files the task needs, scripts, fonts, templates. Claude reads the folder when the task comes up, follows the instructions inside, and uses the supporting files. The skill is the pipeline, saved. Build the task once, drop it in a skill folder, and every future run starts from the same written standard instead of from memory.

The key behavior: Claude does not load every skill at once. It reads the short description of each, and pulls the full instructions only for the one that matches the job in front of it. That is why the description line matters as much as the instructions. The next page shows the anatomy of the file itself.

WHY A FOLDER, NOT A PROMPT

A prompt lives in one chat and dies with it. A skill folder is permanent, versionable, and improvable. When a run reveals a weakness, you edit the file. The pipeline gets sharper, and it stays sharp.

The anatomy of a SKILL.md.

A SKILL.md has two halves. A short frontmatter block at the top, and the instructions below it. Here is a complete, generalized one for an image-generation skill, the kind of component the extraction loop produces.

SKILLS/IMAGE-GENERATOR/SKILL.MD — EXAMPLE SKILL

```
---
name: image-generator
description: Generate a product image from a
  structured prompt. Use when the deliverable is
  a rendered image for a catalog, ad, or page.
  Triggers on "render the product", "make the
  image", "generate the scene".
---

# Image generator

Calls the image model and saves the result.

## When to use
Use whenever a step needs a rendered image.
Do NOT use for diagrams or vector logos.

## How to invoke
Run the bundled script with an absolute path:
  python skills/image-generator/render.py \
    --schema scene.json --out images/out.png

## The standard
The render is scored against the reference
photo. A wrong material, proportion, or piece
of hardware fails the render. Regenerate.
```

The frontmatter is the top block between the `---` lines. `name` is the folder name. `description` is the line Claude reads to decide whether this skill matches the job. Write it so the match is obvious. List the trigger phrases plainly. **The instructions** are everything below: when to use it, when not to, how to invoke it, and the standard it is held to. **Supporting files**, the `render.py` script and anything else, live in the same folder.

Notice the shape. This skill has the three parts of a task from chapter two. The input is a structured prompt. The standard is the reference photo. The failure condition is a visible mismatch. A good SKILL.md is a task definition with a folder around it. If you can write the task down properly, you can write the skill.

The project file structure.


```
# Skills live where Claude can always reach them.
skills/
  brand-constitution/   # SKILL.md: builds the brand facts file
  image-generator/      # SKILL.md + render.py: the shared core
  image-stage/           # SKILL.md: stage one, the styled render
  ad-stage/              # SKILL.md: stage two, builds the ad
  shopify-push/          # SKILL.md: stage three, sends it live
  seo-fields/            # SKILL.md: stage four, rewrites titles
  fidelity-check/       # SKILL.md: the QA gate

# Each client is one project folder.
clients/<brand>/
  brand-constitution.json   # the brand facts: every step reads this
  reference/               # real product photos, the fidelity source
  output/                  # finished renders, dated
  _intel/                  # the knowledge layer: see below
    profile.md              # who the brand is, scope, status
    playbook.md             # the repeatable process for this brand
    decisions.md            # every non-obvious choice, and why
    outcomes.md             # what shipped, what worked
```

THE SPLIT

Skills vs. projects

- Skills are the verbs. They hold the process and do not change from one brand to the next.
- Projects are the nouns. One folder per brand, holding that brand's facts, references, and output.
- A skill reads a project's files. The same stage skills run for any brand by pointing at a different project folder.

THE KNOWLEDGE LAYER

Why _intel/ exists

- A chat ends and its context evaporates. _intel/ is the part that stays.
- It holds the decisions and the outcomes, so the next run does not relearn last month's lesson.
- Keep client-private data walled off in its own subfolder, separate from the transferable process.

The rule is simple. Process goes in skills. Brand facts go in the project. Keep that line clean and the one pipeline serves every brand you take on. Blur it, and you rebuild the pipeline for each client.

The prompts: brand and schema.

A pipeline runs on prompts. Here are two of the three that matter, written to be real and usable. Adapt the bracketed parts and the rest works as is.

PROMPT ONE: THE BRAND CONSTITUTION

PROMPT · BUILD BRAND-CONSTITUTION.JSON STEP 1

```
Read every page of [BRAND]'s live store and the
product photos in reference/, and write
brand-constitution.json with these fields:
  palette      exact hex values, named
  materials    every real material, by product
  finish       matte, gloss, grain, sheen
  proportion   how the product sits and scales
  never        what it must NEVER look like
Rules: confirm every fact against the live
store. Do not guess a material or finish. If
a fact is not verifiable, mark it UNKNOWN.
```

The constitution is the file every later step reads, so the brand does not drift between SKUs. The **never** list is the failure condition from chapter two, written down. A guessed fact is how an image pipeline produces something confident and wrong.

PROMPT TWO: THE STRUCTURED IMAGE SCHEMA

SCENE.JSON · THE IMAGE PROMPT STEP 2

```
{
  "product": {
    "name": "[SKU name]",
    "material": "[from constitution]",
    "keep_exact": "silhouette, hardware, proportion"
  },
  "room": {
    "setting": "[styled room type]",
    "palette": "[from constitution]"
  },
  "light": "[direction, time of day, softness]",
  "camera": "[angle, lens, distance]",
  "negative_space": "[where to leave room]"
}
```

The instruction to the image model is a **schema, not a paragraph**. Fixed fields give the model precise control and a shape it cannot drift out of. A paragraph leaves gaps the model fills on its own. Every required field populated is the standard; an empty field is the failure condition.

The prompt: the fidelity check.

The third prompt is the gate. It runs Claude as the reviewer, scoring the finished image against the real product photo to decide whether it ships.

PROMPT · THE FIDELITY / QA CHECK — STEP 5

```
# Pass Claude the render and the reference.
Compare the render against the reference photo
for [SKU]. Check, one at a time:

  material      does it match the real product
  hardware      pulls, legs, seams, stitching
  proportion    shape and scale unchanged
  color         within the palette, no drift
  never-list    none of the banned traits present

Return PASS only if every check is clean.
On any mismatch, return FAIL, name the
exact check that failed, and say what is
wrong in one line. Do not soften a FAIL.
```

This is the standard from chapter three, written as a prompt you can run. The instruction "do not soften a FAIL" matters. A reviewer that wants to please passes weak work. The gate has to be allowed to fail things, or it is decoration.

WHY THESE THREE ARE ENOUGH

A constitution that fixes the facts. A schema that fixes the instruction. A check that fixes the standard. Those three cover the input, the work, and the gate, and every other prompt in an imagery pipeline is a variation on one of them. They are generalized on purpose: the structure is the transferable part, the bracketed fields are yours to fill. Fill them honestly and you have a working pipeline.

A NOTE ON ADAPTING THESE

Run each prompt once by hand in a chat before you put it in a skill. Tighten the wording, then save the working version into the SKILL.md. The next two pages walk all three through the four-stage pipeline, step by step. The chat is the draft. The skill is the final.

A worked build: the four-stage pipeline.

Here is the pipeline built in full. A home brand makes products faster than anyone can photograph them, so the pipeline takes a new SKU from a plain photo to a finished page and a running ad. Four connected stages, the prompt or config at each, so you can build your own beside it.

The conventional version of this job is a photographer, a studio rental, an ad designer, and someone updating the store by hand. The pipeline version is four stages that run in sequence, each one a skill, each fed by the stage before it. Set up the folder structure from chapter five first, then build each stage.

01 Image generation

Claude fills scene.json from the brand constitution, then the image-generator core renders the plain product photo into a styled room. The fidelity check scores it against the reference before it moves on. This is the studio and the photographer, collapsed into one stage.

Build: image-stage + image-generator skill **Config:** scene.json, page 08 **Failure:** silhouette altered

02 Ad creation

The approved render becomes Meta ads. The same image core sizes it to each placement, then a short script composites the headline and logo by rule, not by eye. Fixed positions, fixed type, fixed margins.

Build: ad-stage skill + composite script **Config:** fixed placement rules **Standard:** matched sizes, no manual nudging

03 Shopify push

The finished render and assets go to the brand's live Shopify store through the store API. The product image is updated in place. Claude only writes the fields the pipeline owns, so nothing else on the page is touched.

Build: shopify-push skill **Config:** store API, scoped fields **Standard:** only owned fields written

04 SEO product fields

Claude rewrites the product title and meta description to one consistent, search-ready pattern, drawing the product facts from the constitution so they stay accurate. Same pattern across every SKU.

Build: seo-fields skill **Config:** the title pattern **Failure:** a field off-pattern or off-fact

What each step is scored against.

Once the four stages are built, this is the table I check a batch against before I call it done. Build the same table for your pipeline. It is the part a founder can actually inspect.

STAGE	SCORED AGAINST	FAILS WHEN
Image generation	The real product photo. Material, hardware, and proportion compared directly against the reference.	The model reshapes the product, or any visible mismatch with the reference.
Ad creation	The placement rules. Brand mark and copy land in fixed positions, sizes matched across placements.	Anything is nudged by eye, or a placement size is off.
Shopify push	The scoped field list. Only the fields the pipeline owns are written to the live store.	A field outside the pipeline's scope is changed on the page.
SEO product fields	The title pattern and the constitution. Every title follows the pattern and every fact is accurate.	A title drifts off-pattern, or states a fact the constitution does not confirm.

HOW THE PIPELINE GETS BETTER

When a render slips past the gate and I catch it in review, I do not file a mental note. I add the failure to the right row of this table, and to the never-list in the constitution. The next run is checked against the stricter version. This is also where the knowledge layer earns its place: the decision and the lesson go into `_intel/decisions.md`, so the fix is permanent and the next brand inherits it.

A year of runs makes the scorecard sharp. That is the compounding from chapter three, happening on a real pipeline. The build was slow. The run is an afternoon. The second brand is hours, because the skills already exist and only the project folder changes.

THE POINT OF THE WORKED BUILD

This is one pipeline: four stages, three prompts, a handful of skills, and a project folder. A new product goes in as a plain photo and comes out as a styled render, a finished ad, an updated store page, and a search-ready title. Build it once. Every product after that is one run.

PART THREE / CLOSING

What still needs you.

You have the method and the build. Here is the part the build cannot do for you.

A pipeline produces an output. It does not decide whether that output was the right thing to make. That decision is judgment, and judgment does not run on a schedule. There are three calls a pipeline cannot make. What to build at all: pointing the pipeline at the right job is a read on the business and the season. Whether the output is good in context: a render can pass the fidelity check and still be wrong for the campaign. When to break the standard: every rule has a case it should not apply to, and knowing which case is the senior part.

This is the review, and it sits on every output before it ships. The pipeline does the production. I do the looking. On this build, a handful of renders passed the fidelity check and I sent them back anyway. The render was accurate and the room was wrong for the brand. The pipeline gave me a clean, fast, consistent output. I gave the brand a decision about whether the output was right.

So build the pipeline. Write the constitution, the schema, the check. Save them as skills, lay the project out with its knowledge layer, and run it. Claude does the production, fast and consistent and tireless on the fortieth pass. Then you do the review. That is not a limitation of the method. That is the method. The whole point of removing the busywork is to be left holding only the part that was always worth your attention.

See the pipeline in depth.

The four stages, the before-and-after imagery, and how the whole thing fits together. Walked through properly. Free, no email required.

connercrowe.com/lab

Conner Crowe · Fractional marketing lead
Home, furniture, and decor brands on Shopify · hi@connercrowe.com