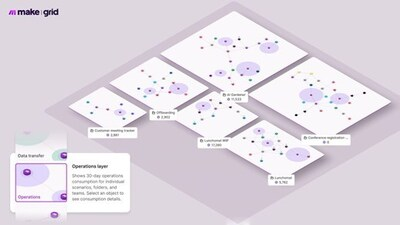
# **C4 and How We’re Implementing It**

This note explains the **C4 model** and how our product adapts it for multi-tool, no-code/low-code stacks (Airtable, Make, n8n, Bubble, Softr, Twilio, etc.). It’s written for both the internal team and external users who want to understand the reasoning behind our diagrams and what each view means.

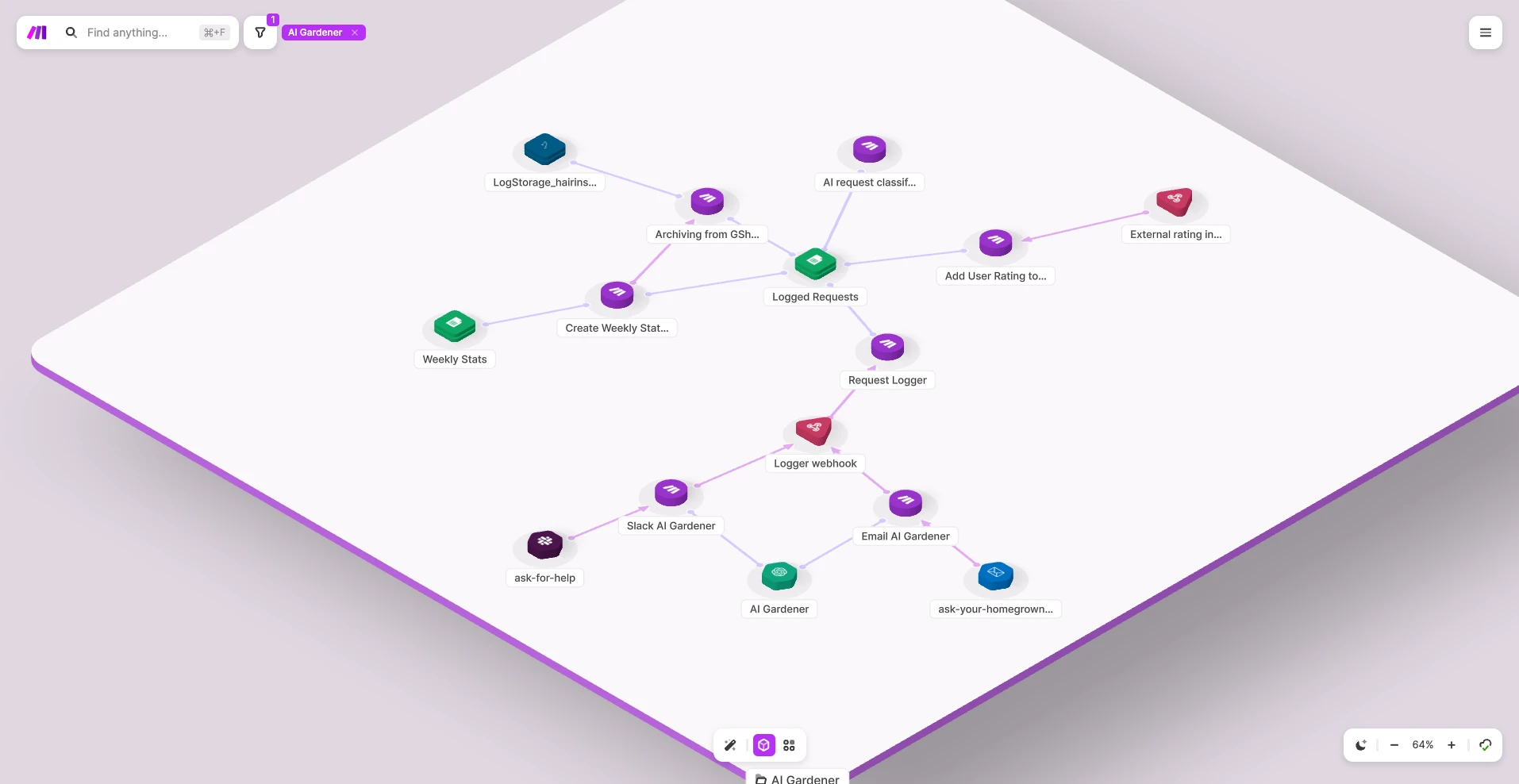
## **1) What is C4 (in one page)**

**C4** is a simple, hierarchical way to document software systems at multiple zoom levels:

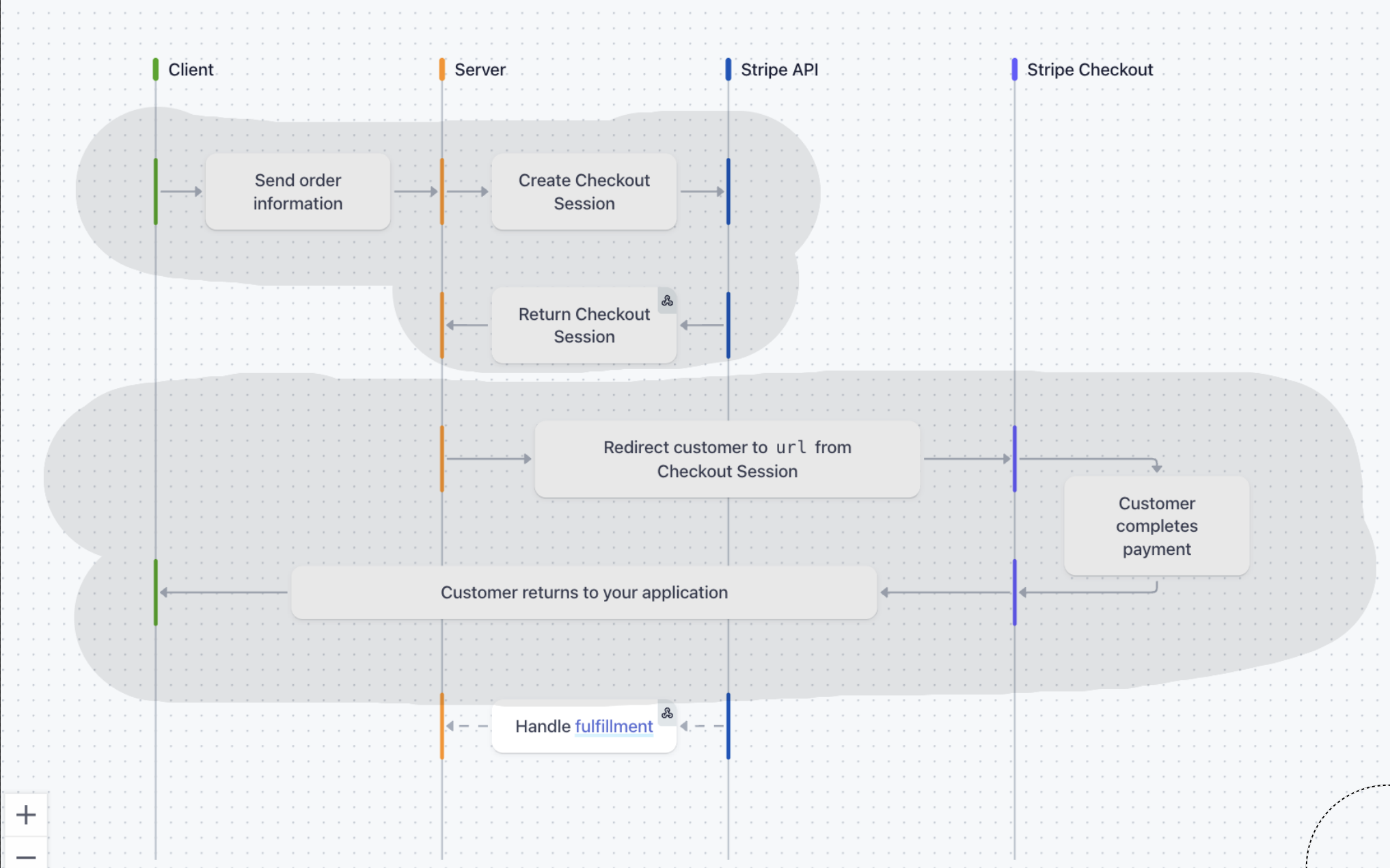
1. **Contexts** – the big picture: people (actors) and systems, and how they relate to “our system.” (user Journeys and Containers inside them)



1. **Containers** – the deployable/runtime building blocks inside the system (web apps, databases, services), and how they talk. (hub and spoke diagram, Feature)



1. **Components** – the internal parts of a container (modules, controllers, etc.) and their relationships. (Workflow Sequence Diagram, Some components are personas)



1. **Code** – the implementation details (classes, functions). (Link to Teable Components inside Component vertical lines. Vertical )

There’s also an optional **Dynamic** view for “what happens over time” (e.g., a sequence of calls in a user flow).

Why it’s popular: it’s **consistent, lightweight, and scalable**—stakeholders can start at the big picture and drill down only as needed.

## **2) Our adaptation of C4 for multi-SaaS/no-code estates**

Traditional C4 assumed one codebase and known “components.” We operate across many SaaS tools and automations. We therefore:

* Keep **Context** and **Container** almost as-is.
* Replace the classic “Component boxes” with a **Dynamic (Sequence) view** that is more honest and useful for no-code data movement.
* Link to “real things” (tables, scenarios, endpoints) as **Components** via deep links, not as a static box-and-arrow component diagram.

### **The three levels you’ll see in our UI**

| **Our Level** | **C4 Equivalent** | **What it shows** | **Who uses it** |
| --- | --- | --- | --- |
| **L1 – Journeys & Features** | **Context** | Journeys (columns), Features (cards), actors and external systems (as chips/notes) | PMs, clients, leadership |
| **L2 – Container Interaction (auto)** | **Container** | Hub-and-spoke map: which **containers** read/write/sync with which, summarized counts | Tech leads, ops |
| **L3 – Swimlane Sequence (per workflow)** | **Dynamic** | Time-ordered steps across **actors** and **containers** (READ/WRITE/PROCESS/TRIGGER/etc.) | Builders, reviewers, onboarding |

**L4 (drill-down)** is not a separate canvas: it’s the click-through to actual **Components** (Airtable base/table/field, Make scenario, n8n workflow, webhook URL, page URL, etc.).

## **3) Key concepts & vocabulary**

* **Journey** – A business journey (e.g., Onboarding, Fulfillment). Groups **Features**.
* **Feature** – A business capability within a journey. A feature can span multiple tools.
* **Workflow (L3)** – A single sequence map inside a feature. A feature can have several workflows.
* **Container** (*C4 container*) – A runtime unit: Bubble app, Airtable base, Make bundle, Twilio service, micro-app, etc. Containers can be reused across workflows.
* **Actor** – A person/role (User, Admin, Agent). Actors appear in sequences but are **not** containers.
* **SequenceStep** – One ordered step: from X to Y, with an operation type and a short label.
* **Component (drill-down)** – A real object in a tool (Airtable table/field, a Make scenario, an endpoint). We link to these; we don’t draw them as boxes.
* **Env (environment)** – Prod, Staging, Dev, etc., applied to containers to prevent mixing environments by mistake.

## **4) Visual grammar (consistent across the product)**

**L3 operations (arrows/boxes):**

* **READ** – solid line with open arrowhead (pull from a container)
* **WRITE** – solid line with filled arrowhead (push to a container)
* **SYNC** – double-headed arrow (bidirectional sync)
* **TRIGGER** – bent arrow / lightning glyph (kicks off a scenario or webhook)
* **WEBHOOK** – dotted arrow from SaaS back into our system
* **SCHEDULE** – clock pill on the lane (delayed action: +48h, +240h)
* **PROCESS** – rounded box on the same lane (in-place processing)

**Lanes:**

* **Containers** have a header ring colored by **env** (Prod/Stage/Dev), with status styling:  
  + **Planned** = dashed border, **Implemented** = solid, **Deprecated** = faded.
* **Actors** have no env ring.

**Placeholders vs. implemented:**

* Steps can reference **placeholder** components (not built yet). They render as dashed chips until linked to a real component.

**Conditional jumps (optional):**

* A step can show →[§A] if clicked and the target step carries [§A] (anchor). We draw the dashed connector only on hover to avoid clutter.

## **5) How L2 is generated from L3 (automation)**

We don’t draw L2 by hand. We **compute it** from the L3 SequenceSteps of a Feature:

1. Aggregate each pair of containers (A→B) by operation type → chips like READ×3 | WRITE×1 | SYNC×2.
2. Compute a simple **hub score** (degree + ops) to place the most central container(s) in the middle.
3. Group spokes by **tool** and collapse long tails into “+N more.”

Result: a clean **hub-and-spoke** map that answers, “What touches our hub and how?” Clicking a spoke jumps to the relevant L3 steps between those containers.

## **6) How L3 sequences are authored (planning-first)**

* On a **Workflow** page, authors create **Containers** (lanes) from a pick-list (existing) or as **placeholders** (to be implemented later).
* Authors add **SequenceSteps** by choosing an operation and clicking from → to.  
  + Each step has a **label** (“Create Lead”, “Send email #2”) and optional **notes**.
  + Optional **component\_ids** can be pasted now or linked later (e.g., airtable:table:appX/tblY).
* A **Feature** view shows all of its **Workflows**, each in a bordered frame (same L3 layout) for quick scanning.

This lets teams plan the flow **before** components exist and then “snap” steps to real components later.

## **7) How components (L4) are linked**

We keep a **Component service** outside Bubble as the source of truth for real objects and URLs. In a step’s drawer you can:

* Paste or search a component ID; we’ll **batch-hydrate** names/URLs when you open the drawer.
* Once linked, the step’s chips turn solid and the drawer shows **Open** links.

This keeps the canvas fast and prevents N+1 lookups.

## **8) Example: “Lead Intake & Reminders”**

**Actors/Containers:** User (Actor) | Bubble App | Airtable (Leads) | Make Automations | Email Service

**Sequence (excerpt):**

1. **User → Bubble** – *TRIGGER: Submit Lead*
2. **Bubble → Airtable** – *READ: dedupe/validate*
3. **Bubble (PROCESS)** – *normalize*
4. **Bubble → Airtable** – *WRITE: create/update Lead*
5. **Make (SCHEDULE +48h)** – *wait*
6. **Make → Airtable** – *READ: eligible leads*
7. **Make → Email** – *WRITE: send email #1 with link*
8. **User → Make** – *TRIGGER: click link (webhook)* [§A anchor]
9. **Make → Airtable** – *WRITE: mark responded*
10. **Make (SCHEDULE +48h)** – *if no click*
11. **Make → Email** – *WRITE: send reminder #2*
12. **User (decision)** – →[§A] if clicked; else continue to final reminder, then **mark stale**.

L2 (auto) will show Airtable at the hub with spokes to Bubble, Make, and Email, and chips such as READ×3 | WRITE×3 | TRIGGER×1 | WEBHOOK×1.

## **9) Do’s and Don’ts (to stay C4-consistent)**

**Do**

* Treat **Containers** as runtime boundaries (apps, bases, services).
* Use **Actors** only in L1/L3; keep L2 technical (containers only).
* Keep a **small, fixed set of operations** (READ/WRITE/SYNC/TRIGGER/WEBHOOK/SCHEDULE/PROCESS).
* Mark **env** on every container; default views show **Prod**.
* Plan with **placeholders**; link to components later.

**Don’t**

* Don’t model a **Bubble page** or “Airtable field” as a container; mention pages/fields in step labels or as component links.
* Don’t mix **Prod** and **Dev** lanes without a warning—cross-env arrows are highlighted.
* Don’t overload the sequence: if it grows past ~5 lanes or ~10 visible steps, **compress** repeated ops (READ×N) or split into another workflow.

## **10) Why we prefer Dynamic (Sequence) instead of a static Component diagram**

For no-code estates, the important questions are **order and responsibility**: “Who calls whom? When? With what delay? What happens if the user clicks?” A dynamic, Stripe-style sequence diagram answers these directly and maps naturally to real tools and links. Static component diagrams are great for codebases but become misleading in multi-SaaS workflows where timing, webhooks, and retries matter.

## **11) What each audience gets**

* **Leads/Clients (L1):** Plain cards that say what a Feature does and which tools are involved.
* **Tech Leads/Ops (L2):** A truthful map of **which containers interact** and how (read/write/sync/trigger) without hand-drawing.
* **Builders/QA (L3):** The exact run path, with delays and decision points, and **one-click links** to the real tables, scenarios, and endpoints.

## **12) Roadmap alignment**

* **Today:** L3 editor/viewer (planning-first), L2 auto from L3, L4 links on click.
* **Next:** Usage/change overlays, search-pickers for components, snapshots (“Publish v1.0”), and (optionally) a topology lens for very branchy workflows.

### **TL;DR for sharers**

We’re **C4-aligned**:

* **Context** = Journeys/Features
* **Container** = Hub-and-spoke map (auto)
* **Dynamic** = Swimlane sequence (our default detailed view)
* **Code/Components** = deep links to the real objects

This approach is **live, consistent, and actionable** for multi-tool no-code teams: plan first with placeholders, then flip to implemented as components go live—no redraws, no stale docs.