Ceruleo Training Manual - Company Admin Edition

This edition is for the people who run the company workspace: owners, studio leads, operations - anyone who decides how Ceruleo is set up rather than only working inside it. It assumes you have read (or will read) the **General User edition**; everything there applies to you too. What this edition adds is the machinery: the settings, templates, and invitations that shape what everyone else experiences.

Every configuration decision in this edition serves one idea: **the work drives the shape.** Ceruleo is built for creative, non-linear, iterative projects - the kind that start as a spark, not a scope document - and your job as an admin is to provide scaffolding that supports that evolution, never a cage that prevents it. Templates are starting sketches. Permission grids are guardrails. When this edition shows you a knob, it will also tell you what kind of team each setting fits - look for the ⚙ **Make it yours** notes.

01 The one distinction that governs everything

Ceruleo has exactly two kinds of people, and the difference is set at the moment you invite them:

- A **company invitation** makes someone a company user: they join your workspace, see company tools (Sparks, contacts, programs, statuses), and can be granted company-level capabilities.
- A **project invitation** makes someone a collaborator on that one project - nothing more. Clients, vendors, and partners belong here. A project invitation **never** grows into company membership on its own; there is no promotion path except an explicit company invite.

Cost of choosing wrong, in both directions: invite a vendor to the company and they can see your company contacts, programs, and idea pipeline - things you almost certainly did not mean

to share. Invite an employee to projects only and they will work for months missing the company half of the platform, usually without knowing it exists. When someone says "I don't see Sparks," this distinction is the answer far more often than permissions are.

Each person belongs to at most one company - a second company invitation will be refused, not merged.

02 Getting around

Every page inside a project carries the **project bar** at the top: the project name (click it for recent items, other projects, and the dashboard) plus one-click links to every tool the viewer can open. Worth knowing as an admin: the bar is permission-filtered per member by the same per-project grids you configure - a member's bar shows exactly the tools their grid allows, and the **Organizer** entry appears only for people who can reorganize.

03 How this edition is organized

The chapters follow the order in which a company actually gets set up: workspace and branding → members, invitations, and per-member capabilities → permission templates → programs and project statuses → template families (structure, logs, notes, flags, spark categories, asset types) → the company library → contacts administration. The template chapters cross-link into the General User parts they power - structure templates into Part 1, log templates into Part 4 - and the shared material appears in both editions so neither reads with holes in it.

Part 1 - The Project Organizer

Most project management tools are only useful once a project has already been defined - someone hands them a finished scope and they track it. Ceruleo starts earlier, at the spark of an idea, which means the Organizer cannot work the way a normal tool's structure works. **You do not design the tree and then do the project. You sketch the tree in broad strokes, and the creative work itself carves it into shape.**

The Organizer is still the skeleton everything hangs from - flags, RFIs, schedule items, published drawings, and rallies all point at its nodes, and the payoff is still the **element dashboard**, where one click shows everything the team has ever attached to a piece of the build. But the skeleton is alive. It starts vague because the idea starts vague, and it earns its precision the same way the idea does: through iteration.

04 The mental model

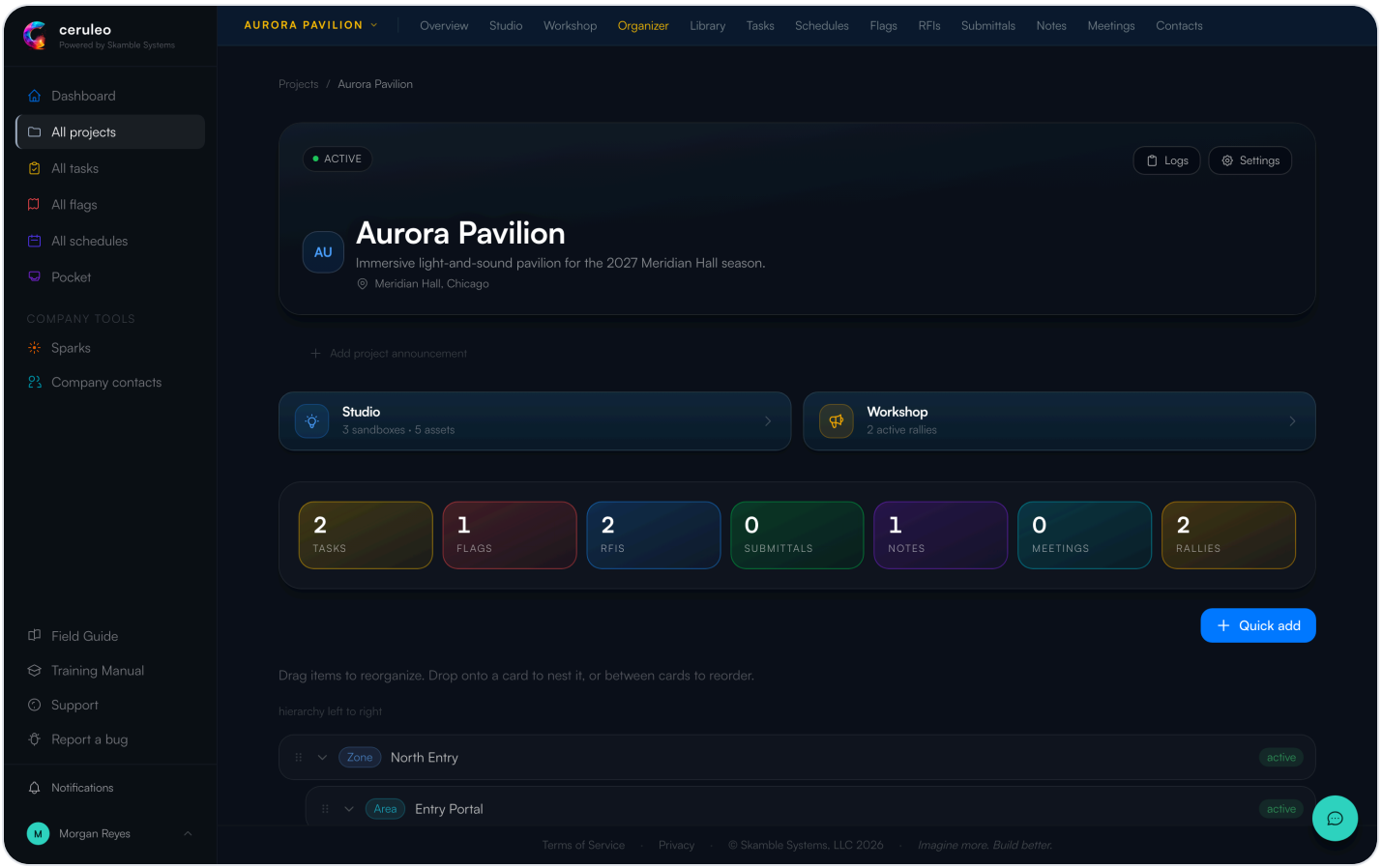
Three levels - **Zones → Areas → Elements** (the labels are configurable, the shape is not) - that mature through the life of the project:

- **Day one, the tree is a sketch.** A handful of broad zones taken from the creative brief, named honestly at the resolution you actually have. "Entry experience" is a perfectly good zone name in week one. "Portal Arch" would be a lie - nobody has designed an arch yet.
- **During creative development, the tree lags the studio slightly - on purpose.** Ideas live in Sparks and take shape in Sandboxes (Part 2). While something is still three competing concepts, it does not deserve a node; the moment one concept wins and gets a name, it does.
- **The publish moment is the forcing function.** A studio asset cannot be published without an element to land on. When the Entry Portal package is approved and ready to publish, its elements must exist - so approval reviews naturally become the rhythm where the tree gains its next ring of detail.

- **By fabrication, the tree is site-walkable.** Every build node names a place or a thing someone can point at during a site walk, and the element dashboards have become the project's memory.

The tree's job at every stage is the same: to be an honest map of the idea *as currently understood* - never more detailed than the thinking, never lagging so far behind that work has nowhere to land.

Not every branch has to be something you can touch. The tree reads most naturally as a map of the physical build, but a node is really just a home for a body of work that belongs together - and work that matters to the project but has no physical home deserves a branch of its own. A *Marketing* zone can carry elements for the launch campaign and press materials; an *Operations* area can hold staffing plans and run-of-show; a *Sponsor Deliverables* element can collect everything owed to a partner. These earn dashboards exactly the way physical nodes do: one click gathers every flag, task, note, and asset attached to that effort. The same maturity discipline applies - give a workstream a node when there is real work to land on it, not before.



Aurora Pavilion's Organizer in month three: what began as three broad strokes has been carved into the real zones, areas, and elements by the creative work itself.

05 The scenario

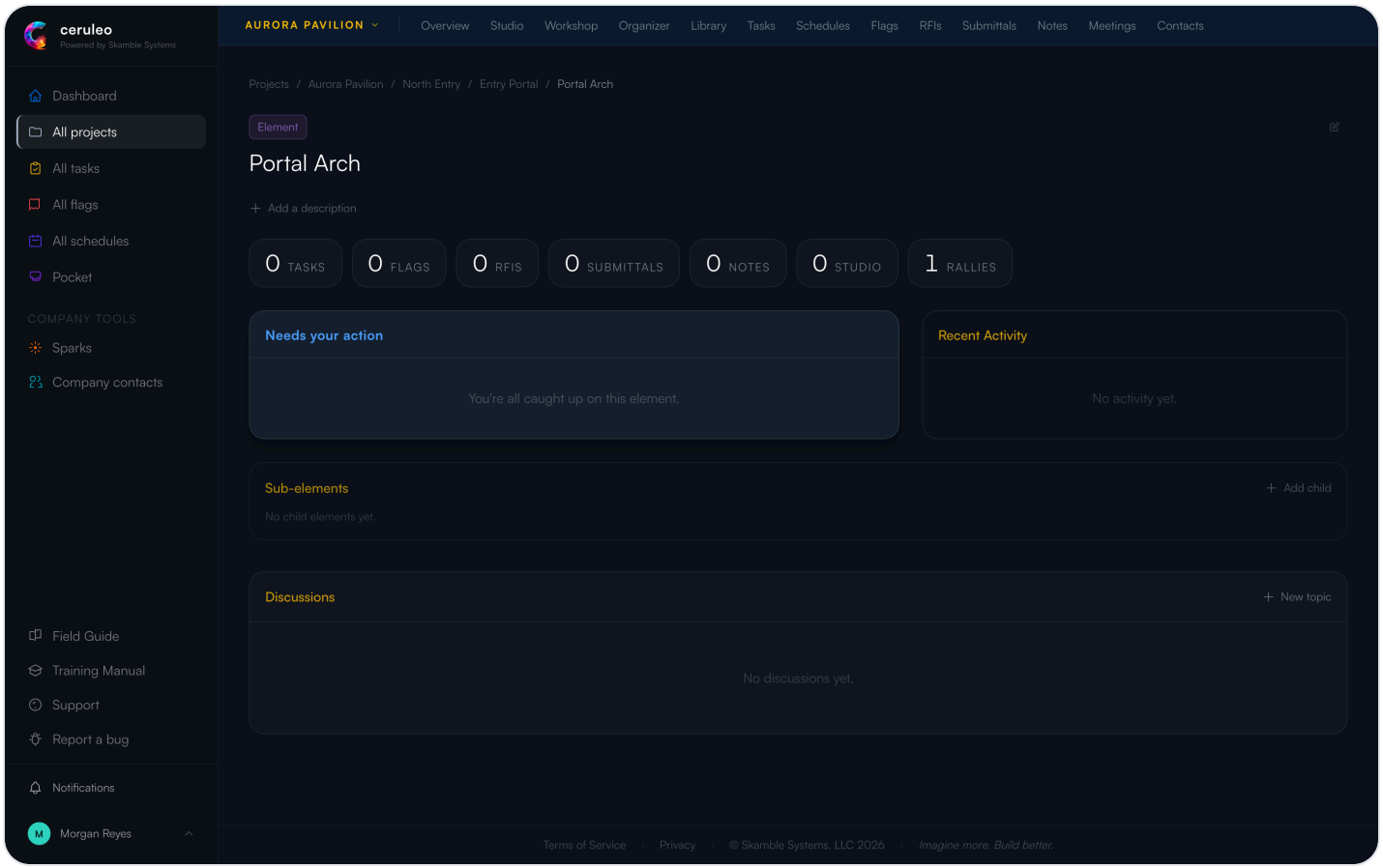
Aurora Pavilion did not begin as a structure. It began as a company Spark - **"Aurora Light Curtain,"** a fiber-optic threshold visitors part with their hands - that survived enough scrutiny to become a project (Part 2 tells that half of the story).

Week one. Morgan, leading the project, opens the Organizer and sketches exactly what the pitch knows and nothing more: three zones - *North Entry*, *Main Stage*, *Concourse* - because the venue and the show format imply that much. Under North Entry, one area: *Entry Portal*, since the light curtain idea clearly lives at the threshold. No elements anywhere. Ten minutes of work, and every early flag, note, and RFI now has somewhere broad-but-true to land.

Development. Jules explores the entry in the *Entry Portal Concepts* sandbox - arch studies, curtain mockups, queue treatments. None of these get nodes yet; they are still arguments, not decisions. When the direction settles into a portal arch flanked by the curtain with rails managing the queue, *those names* become the elements: **Portal Arch**, **Light Curtain**, **Queue Rails**. The tree just learned what the studio decided.

Firming. The wayfinding package matures fastest. When Morgan is ready to publish the approved sign family, publishing demands a destination - so *Wayfinding* gains **Monolith Signs** and **Directional Blades**, and the approved assets land on them. From here on, anyone standing in the concourse asking "what's the current sign spec?" opens the element and reads the answer.

Fabrication. Riley - the outside fabricator, not a Halcyon employee - files a flag about a cracked acrylic panel. It lands on **Light Curtain**, an element that did not exist when the project started and now carries the whole history of that piece: the concept lineage, the published spec, the flag, the tasks.



The Portal Arch element dashboard: the whole story of one piece of the build - a story that started as a spark, not a spreadsheet row.

06 Decision rules

Rule: sketch broad strokes on day one - never a finished tree, never nothing. Ten minutes, a few zones from the brief, an area where the founding idea obviously lives. Project creation defaults to a **blank** structure and nothing ever forces you back, so the sketch is a discipline, not a prompt.

Cost of skipping it: the **empty-organizer anti-pattern** - the most common failure we see. Work happens, but every record lands "somewhere in the project"; nothing can be traced to a place, and publish stalls the first time an approved asset has no element to land on. Retrofitting structure after three months means re-placing hundreds of items by hand.

Cost of overdoing it: a fully imagined tree on day one is fiction, and teams treat it as decided. Elements named before the studio has explored them quietly pre-empt the creative work - the tool starts dictating the design. If a node names something nobody has chosen yet, delete it.

Rule: a concept earns its node when it wins, not while it competes. Three arch options in a sandbox are one future element, not three nodes. Promote the winner's name into the tree at the moment of decision - that is the handshake between the studio and the map.

Cost of choosing wrong: nodes for losing concepts litter the tree and collect misplaced work; waiting too long past the decision leaves finished thinking with nowhere to land, and people fall back to unplaced items.

Rule: name nodes at the resolution you honestly have, and rename freely as resolution improves. "Entry experience" → "North Entry" → with *Portal Arch* under it is healthy evolution, and renaming is free - every placed item follows automatically. Name after places, things, or standing workstreams - never after time, trades, or people: phases belong to the schedule (Part 3), disciplines to how you assign work. (Workstream branches like *Marketing* are the exception that proves the trade rule: their work has no home anywhere on the physical tree. A *Lighting* zone fails precisely because its work does - on every element it touches.)

Cost of choosing wrong: a "Phase 2" zone goes stale the moment the schedule shifts; a "Lighting" area forces every multi-trade element into two homes or none. Either way people stop trusting the tree, and the dashboards go dark.

Rule: place work at creation, not "later." Every create dialog that can carry an element offers a placement picker, and creating *from* an element page pre-fills it. Placement is optional almost everywhere by design (an unplaced task legitimately means

"whole project"), so the habit has to come from you - even in week one, when "placed" just means "on the right broad zone."

Cost of choosing wrong: unplaced items are invisible from every element dashboard. They still exist in the tool tabs, but nobody browsing the *thing* will ever meet them. See also the **orphaned-rally anti-pattern** in Part 2.

07 Who can shape the tree

Adding elements is deliberately open: any project member can add nodes by default (an admin can withhold this per member). **Reorganizing** - moving and re-nesting what exists - is deliberately narrow: project admins, org admins, and members explicitly granted reorganize rights. In a tree that is *supposed* to keep changing, that asymmetry is the point: growing the map as ideas win is everyone's job; rewriting the map is a decision, made by whoever holds the whole picture. Treat reorganization as a rhythm - a quick pass at each phase gate, when the studio's decisions have outrun the sketch - not as an emergency.

If the Organizer tab won't let you drag nodes, that is not a bug - ask your project admin, and have a reason ready.

08 Troubleshooting

- **"I can't find where someone put something."** Check the tool tab and look at the item's placement line. If it has none, it was created unplaced - place it now (open the item, add the element) rather than filing a duplicate.
- **"Our tree still looks like week one and the project is half done."** The studio outran the map. Don't boil the ocean: promote the *decided* concepts to elements first (start where publishing is imminent), re-place only the open items, and leave closed history where it lies.
- **"Two elements are really the same thing."** Move the work to the survivor, then delete the duplicate. Deleting a node does not delete the work placed on it - those items simply

become unplaced, so re-place them immediately. But deleting a node **does** delete everything *nested under it*: removing an area removes its elements too. Empty the branch before you prune it.

- **"The design changed and half the tree is obsolete."** That is the system working - the tree is supposed to trail the idea. Rename what evolved, prune what died (see above), and resist the urge to keep dead structure "for the record": the record lives on the items, not the scaffolding.

> **Mechanics** - creating elements, the drag-to-reorganize gestures, and the structure section of project settings are covered click-by-click in the in-app **Field Guide → Organizer**.

09 The admin's half: structure templates

If your company starts the same *kind* of project repeatedly - pavilions, retail rollouts, touring sets - a structure template codifies the day-one sketch so no project lead rebuilds it from memory. Company settings → structure templates lets you define the level labels and the broad-strokes starting skeleton once; project creators then pick the template instead of the blank default.

> **Rule: the template is the sketch, not the finished tree.** Ship the zones a project of this kind always has, the naming convention, and nothing that a studio has not yet designed. Elements are the creative work's output - a template that pre-names them dictates design decisions before anyone has had the idea.
>
> **Cost of choosing wrong:** over-stuffed templates get skipped ("blank was faster"), and you are back to the empty-organizer anti-pattern - now with a template nobody uses. Worse, teams that *do* use an over-detailed template inherit a fictional tree and spend the concept phase working around someone else's guesses.

Two operational notes:

- Templates are copied at project creation. Editing a template later does not touch existing projects, so template improvements only compound on *future* work - another reason to invest in them early.
- Watch new projects for the blank default. A project created blank three weeks ago with real activity in it is the cheapest possible intervention point: one conversation with its lead now, or hundreds of unplaced items later.

Part 2 - Sandboxes and Rallies

Part 1 said the creative work carves the Organizer into shape. This part is about where that work actually happens. Ceruleo gives you three containers for an idea, and choosing between them is the most consequential judgment call in the platform - not because any choice is fatal, but because each container gives an idea a different kind of life:

- A **Spark** keeps an idea *alive without commitment*. It has an owner and a discussion, and it costs nothing to keep.
- A **Sandbox** gives an idea *room to become something* - explorations, versioned assets, reviews, and eventually approved work that publishes onto the Organizer.
- A **Rally** gives a question *a burst of collective attention* - a short, focused push where contributions pile up fast and end in decisions and action items.

The one-line test: **Spark to keep it, Sandbox to make it, Rally to decide it.**

10 The pipeline (the spine of the whole platform)

The journey from nothing to built runs: **spark → sandbox → asset → approved → published onto an element.** Every stage is a deliberate gate, and each gate exists to protect the stage before it.

1. Sparks - the front porch. A company Spark lives in the company-wide pool, visible to company users: it is where "what if the entry was a curtain of light you part with your hands?" waits for its moment. A *project* Spark lives inside a sandbox on a specific project - same idea-shape, but already attached to context.

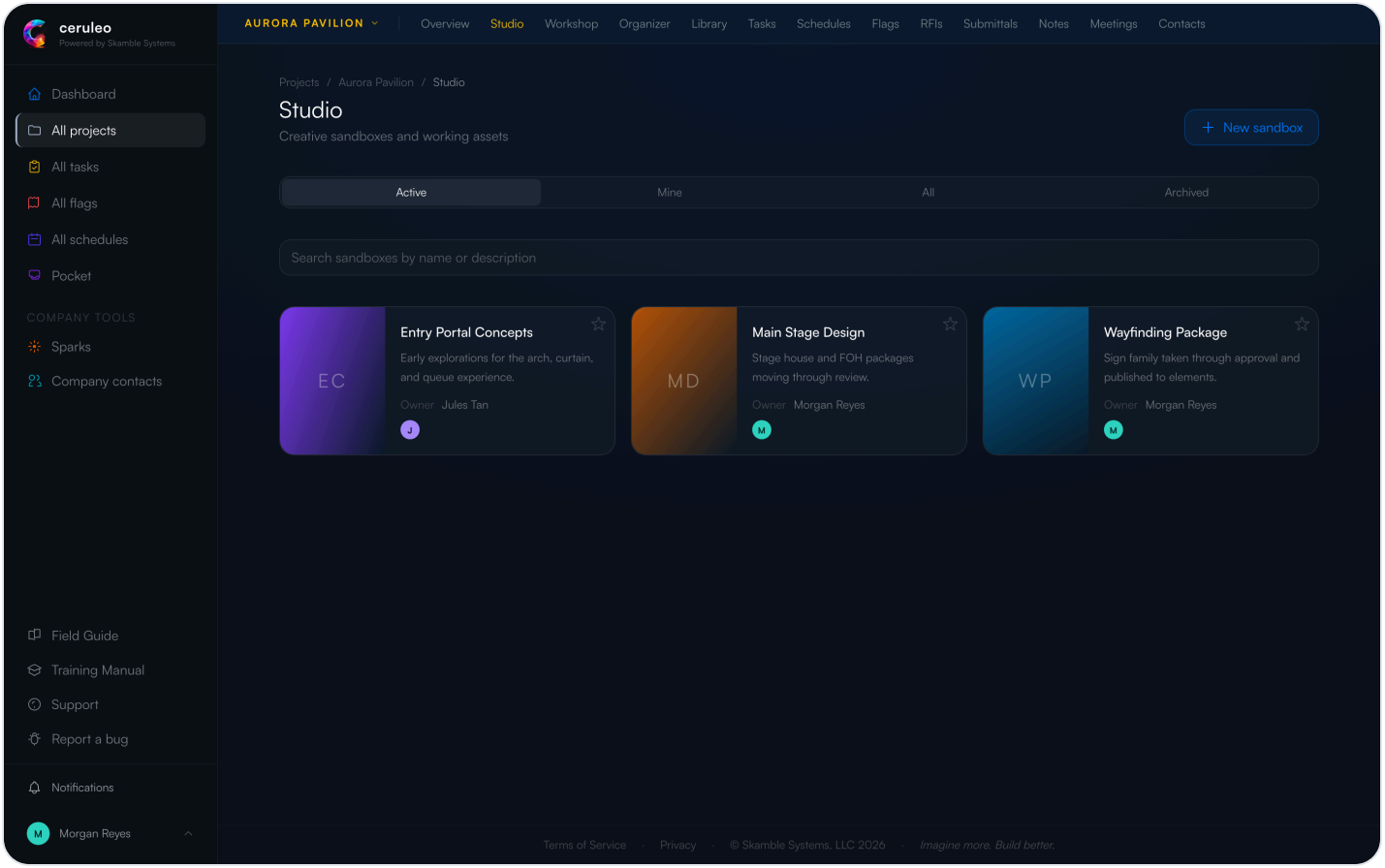
2. Conversion - the commitment. Converting a spark moves it into a sandbox - into an existing one, or into a brand-new sandbox created for it. Understand the mechanics: conversion is a **move, not a copy**. The spark's discussion and lineage travel with it; there is no residual "idea copy" left on the porch. That is deliberate - the pipeline never forks an idea into two lives that drift apart.

3. Sandboxes - the making. A sandbox is a workroom, not a filing cabinet: concept studies, dead ends, competing options, and eventually **assets** - the documents, files, and models that accumulate versions as thinking hardens. Assets carry a status ladder (draft → concept → development → not-for-construction → approved) that tells everyone how much weight a document can bear. Nothing in a sandbox is placed on the Organizer, and that is a feature: the map only learns about winners (Part 1).

4. Review - the pressure test. When an asset version needs judgment rather than chatter, request a **review**: named reviewers, a due date, real tasks on each reviewer's plate, and recorded decisions.

5. Approval and publish - the graduation. Approving an asset freezes its review cycle - approved means *done arguing*. Publishing then pushes it out of the workroom onto the Organizer, and publishing has two preconditions: the asset must be **approved**, and it must have an **element to land on** - the forcing function from Part 1 that keeps the map growing. Publish to everyone, or to a specific audience when the work is sensitive.

From that moment, the element dashboard - not the sandbox - is where the project reads the current truth.



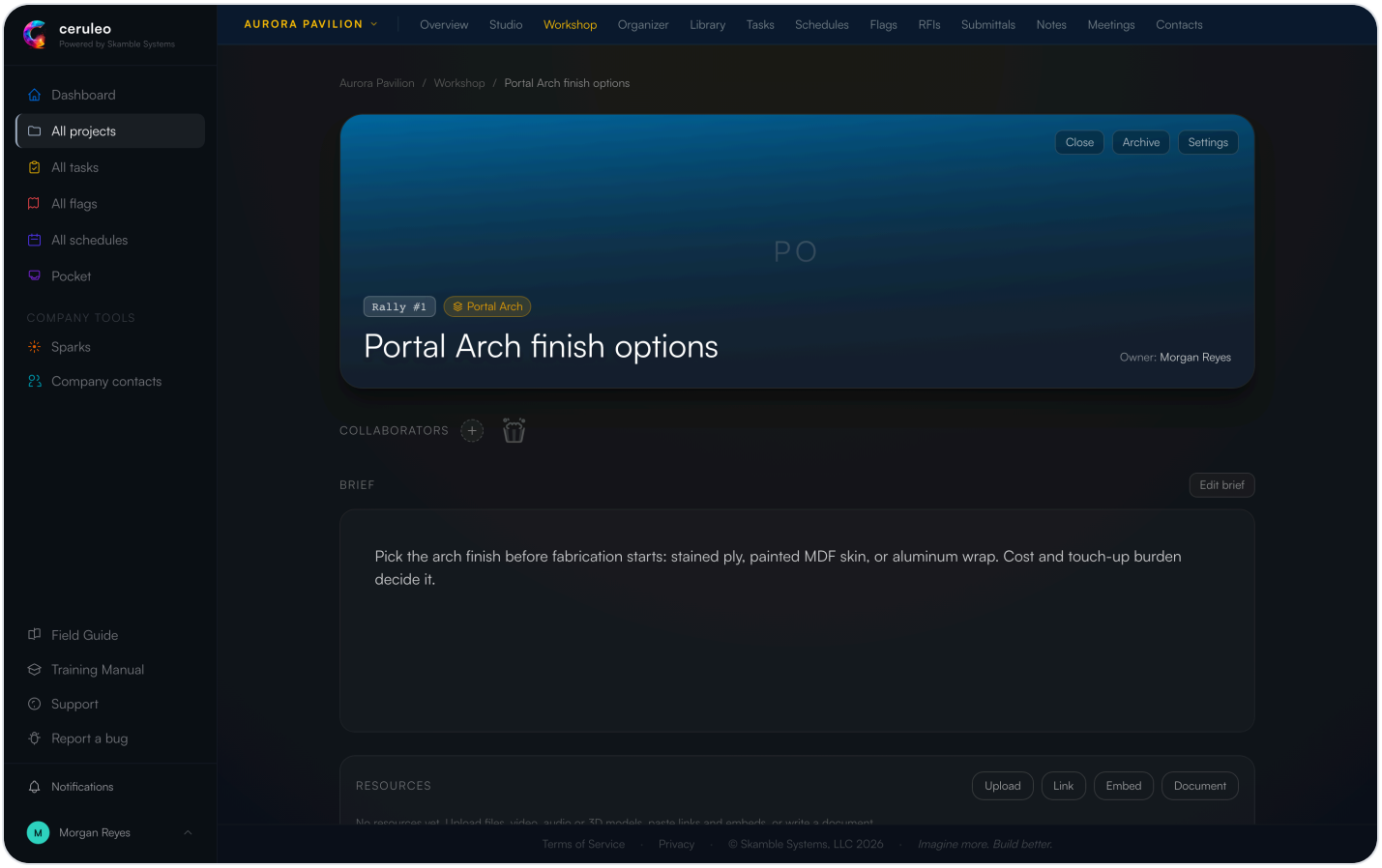
Aurora Pavilion's Studio: three sandboxes at three maturities - early explorations, work in review, and a package already published onto its elements.

11 Rallies - the other container

A rally is not a slower sandbox; it is a different physics. Sandboxes hold *work products* that converge over weeks. A rally holds *a question* - "which arch finish?", "how do we sequence load-in?" - and burns hot for days: typed contributions (updates, questions, blockers, ideas, feedback), a brief that frames the problem, action items that become real tasks on real plates, and a close when the question is answered.

Link rallies to the elements they concern. A rally about the Portal Arch finish belongs on **Portal Arch**, where its outcome will still be findable in month six. The platform will never force the link - some rallies genuinely concern the whole project - but an unlinked rally about a specific thing is the **orphaned-rally anti-pattern**: the discussion happens, the decision gets made, and the element's dashboard never heard about it. (The create dialog offers element linking, unlinked

rallies wear a quiet "No element" tag, and element dashboards list their linked rallies - the tooling nudges; the judgment is yours.)



"Portal Arch finish options": a brief, typed contributions from three people including an outside fabricator, and an action item already on someone's plate - all linked to the element it will change.

12 The scenario

The **Aurora Light Curtain** spark sat on Halcyon's porch until the Meridian Hall opportunity made it real. Morgan converted it into a fresh sandbox on the new project - *Entry Portal Concepts* - where Jules's arch studies and curtain mockups took over. When the direction won, the elements got their names (Part 1), and the concept sketch that started it all remains in the sandbox as version history: the idea's paper trail from porch to pavilion.

Meanwhile the *Wayfinding Package* sandbox matured fastest: sign family approved, published onto **Monolith Signs** and **Directional Blades**. The *Main Stage Design* sandbox shows the

middle of the road - the LED wall layout at Rev B, an open review with Dana and Jules on the hook, procurement waiting on their verdict.

Two cautionary exhibits live in the same project. The **Kinetic Petal Canopy** spark has sat on the porch for months, owned but untouched - the **spark-that-never-transitions anti-pattern**. Sparks are cheap to keep, and that is exactly the danger: a porch full of ideas nobody will ever convert *or archive* stops being a pipeline and becomes a graveyard that buries the good ones. Review the porch on a rhythm; convert what's ready, archive what isn't, and let owners feel ownership. And the **Load-in logistics** rally is running with no element link - defensible as a whole-project question, but if it settles anything about the dock or the crates' route through the *North Entry*, that decision will be invisible from every element it touched.

13 Decision rules - quick reference

- **Spark vs Sandbox vs Rally:** keep it / make it / decide it.
- **Company spark vs project spark:** "should we ever?" vs "should we, here?" - and project sparks are visible to sandbox collaborators, including externals.
- **Convert into an existing sandbox vs a new one:** existing when the idea *feeds* work already moving; new when the idea *is* the work. A sandbox with two unrelated ideas in it will version them into each other's way.
- **Comments vs review request:** shaping vs verdict. Reviews create tasks and records; comments create conversation.
- **Approve, then publish; element first:** approval ends the argument, publish needs a landing spot, audience defaults to everyone - narrow it only when the work is genuinely sensitive, because private publishes fragment the "current truth" the element dashboard exists to provide.

> ⚙ **Make it yours** - the pipeline's vocabulary is configurable at the company level: spark categories, asset categories and types (company settings → template families) shape the create dialogs your team sees. Per-project, the studio permission grid supports a dedicated **reviewer** level - collaborators who can judge work without editing it - and each sandbox can allow or withhold collaborator versioning. Privacy toggles exist on sparks, sandboxes, and rallies for work

> that needs a smaller room. The gates are fixed - approve before publish, element before publish, conversion moves - because they are the scaffolding; everything else bends to how your team works.

14 Troubleshooting

- **"I converted a spark and now I can't find it in Sparks."** Working as designed - conversion moves. Its whole history is inside the destination sandbox.
- **"I need to publish but the button won't let me."** Check the two preconditions in order: is the asset approved? Does it have an element? If the element doesn't exist yet, that is Part 1's signal - the tree just fell behind a decision. Add the element, then publish.
- **"A review is stuck on someone who never responds."** Reviews are tasks on their plate - chase the task, or replace the reviewer. Do not route around a stuck review with an informal "looks fine" comment; the record will say the version was never judged.
- **"We keep debating in a sandbox's comments and going in circles."** That is a rally trying to happen. Frame the question in a brief, set the room, and burn it down in a week - then bring the answer back to the sandbox as the next version.

> **Mechanics** - creating sparks and sandboxes, the convert flow, versioning, review requests, publish and audiences, rally contributions and closing are covered click-by-click in the in-app **Field Guide → Studio / Workshop / Sparks**.

15 The admin's half: keeping the pipeline honest

The pipeline runs on vocabulary and rhythm, and both are yours to set.

Vocabulary *(company settings → template families)*: spark categories sort the porch; asset categories and types shape what the studio's create dialogs offer. Keep the lists short enough to be chosen correctly without thought - a category nobody can place an idea into gets everything filed under the first option.

> ⚙ **Make it yours** - per-member, `sparks` participation can be made explicit rather than default, and the studio permission grid's **reviewer** level is your tool for clients who should judge milestones without wandering the workroom. Decide these per relationship, not per policy.

Rhythm: the porch review. Put a recurring pass on the calendar - monthly is plenty - where spark owners answer one question per idea: convert, keep, or archive? The spark-that-never-transitions anti-pattern is not a user failure; it is an *unscheduled decision*. The moment the review exists, the graveyard empties itself.

Watch two signals in the Studio across projects: sandboxes with many assets and no reviews (work converging without pressure-testing - nudge a review before "approved by momentum" happens), and rallies open past their useful life (a rally that burns for a month was a sandbox wearing the wrong container - help the team migrate it).

Part 3 - Scheduling

A schedule in most tools is a contract written before the work is understood - which is why most schedules are fiction by week three. Ceruleo's scheduling is built for the other reality: creative projects where the shape of the work is still emerging while the deadline is already real. The schedule here is scaffolding that *maintains itself* as things move, so the team spends its attention on the two things software cannot know: what depends on what, and what "done" means.

16 The mental model

A project is not one schedule. It is a set of schedules tied together by logic. This is the part most platforms cannot do, and it changes how you plan: the design studio, the fabrication shop, and the install crew each keep their own schedule - their own rhythm, their own working days, their own approver - and **dependencies flow freely across them**. The shop's "arch fabrication" can wait on the studio's "shop drawings" even though they live in different schedules owned by different leads. One team's slip re-flows the other team's dates automatically, across the boundary, without anyone maintaining a master spreadsheet that pretends all three teams are one.

Within any schedule: phases hold items; items carry dates; dependencies carry the logic. Once you tell the schedule that arch fabrication follows shop drawings with two days of lag, you never manually move arch fabrication again - its dates are *derived*. Change the drawings' finish date and everything downstream re-flows. That is the core trade the tool offers: an item with a dependency gives up manual dates and goes **auto**. Accept the trade everywhere you honestly know the logic; keep items manual only where dates are genuinely imposed from outside (a venue date, a permit window).

Milestones are promises, not work. "Load-in complete" has no duration and no assignee's hours - it exists so that dependencies have a clean thing to converge on and the team has a

clean thing to celebrate or miss. If it takes effort, it is a task item; if it marks a state of the world, it is a milestone.

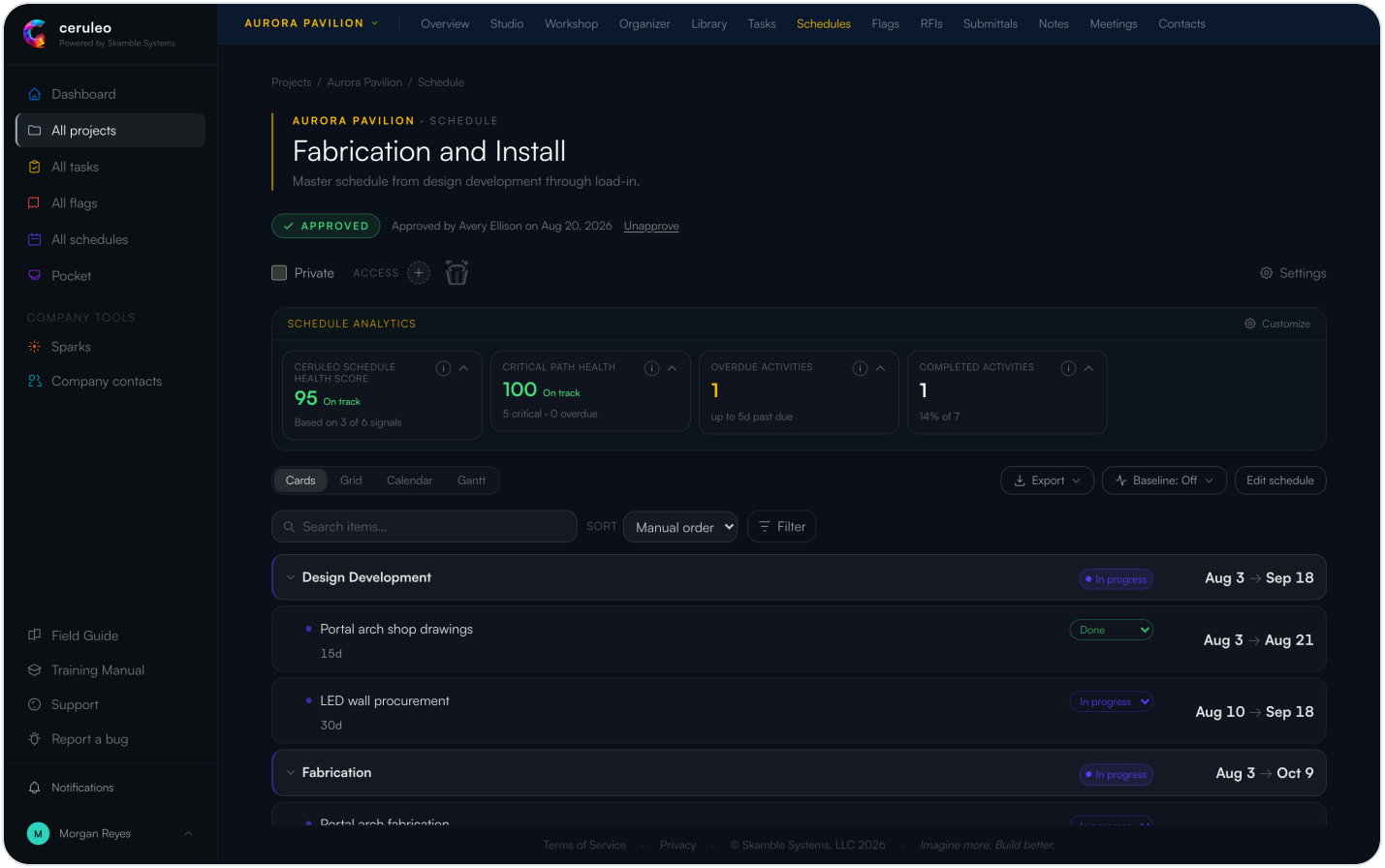
Approval is a handshake, and edits break it - on purpose. A schedule is a **draft** until an approver signs it; from then on it is the version people may rely on. The moment anyone materially changes an approved schedule's items - dates, structure, dependencies, milestones - the approval is automatically revoked and the schedule returns to draft for re-approval. That is not friction; that is the tool refusing to let "approved" and "current" drift apart silently. Plan your edits in batches, then re-approve once.

17 Many schedules, one web of logic

Most tools force a choice between two bad options: one giant master schedule that no single team owns (so nobody maintains it), or separate per-team schedules that silently drift apart (so nobody trusts them). Ceruleo refuses the choice. Give every team that plans differently its **own** schedule, then wire the **hand-offs** - the few places where one team's finish is another team's start - as real cross-schedule dependencies.

The hand-off points are usually milestones: the studio's schedule ends in "Drawings released," and the shop's first item depends on it; the shop's "Crates ready" milestone feeds the install crew's "Load-in." Each team sees a schedule small enough to be honest about, each lead approves their own, and the project still behaves as one organism: when drawings slip a week, the shop's dates and the install crew's dates re-flow the same hour, with no meeting required.

A schedule that must not be moved by outsiders - a venue's fixed calendar, a contractual window - can switch off incoming dependencies entirely: other schedules can still watch it, but nothing outside it can push its dates.



Aurora Pavilion's "Fabrication and Install": approved by name and date, health metrics on top, one item honestly overdue - visible, not buried.

18 The scenario

Morgan builds the master schedule in three phases - Design Development, Fabrication, Install - and wires the logic instead of the dates: shop drawings feed arch fabrication (finish-to-start, two days of lag for the print shop); monolith fabrication can *start* ten days after arch fabrication starts (start-to-start - the shop can parallelize once the first rhythm is set); everything converges on load-in, and load-in feeds the one milestone, **Load-in complete**. Avery approves it, and that approval line - who, when - stays on the page.

As the project grows, this becomes the seed of a family: when the install crew comes aboard, they will keep their own schedule on a seven-day site week, its first item hanging from this schedule's "Load-in complete" milestone - their planning, their rhythm, wired to Morgan's dates at exactly one point.

Then reality: the powder-coat vendor slips. "Queue rails powder coat" runs past its finish date and the schedule *says so* - overdue count up, health score down from 100. Nobody re-paints the schedule to look better; the overdue item is an honest signal wired to a real conversation with a real vendor. Riley, assigned to the fabrication items, sees them on their own plate without ever opening the Gantt.

19 Baselines: the memory of the plan

The live schedule always shows the *current* truth - which means it quietly forgets what you originally promised. A **baseline** is how the schedule remembers: a named, frozen copy of every item's dates, captured at the moment you save it. Select a baseline and the views show the drift - where reality has pulled away from the plan you committed to.

Two facts make baselines safe to use without anxiety:

- **A baseline is a copy, not a reference.** The instant you save one, it is carved off from the live schedule. Editing dates afterward - including every autosaved change - touches only the live schedule. **Nothing you do later can alter a baseline you already saved.**
- **Capture is instantaneous and whole.** Whatever the schedule says at the moment you click save is exactly what the baseline holds - no more, no less.

That makes the order of operations simple. The rhythm:

1. **When a plan is agreed** (typically right at approval), capture a baseline and name it for the moment: "Contract baseline," "Approved plan v1." This is the promise, frozen.
2. **Work happens.** Dates move, items slip, the live schedule stays honest. The baseline does not move - that gap *is* the information.
3. **When you re-plan**, finish all the date and dependency changes first, re-approve the schedule, and *then* capture a new baseline named for the event: "Re-plan after venue slip." The new baseline records the new promise; every older baseline still records its own era, untouched.

One special case: if you are about to re-plan a schedule that was *never* baselined, capture one **before** you start editing - a last-chance snapshot of the old plan - then edit, then capture the

new plan when it settles. Saving a baseline is the schedule creator's or an admin's move; baselines can be renamed or deleted later, but never edited - by design, there is no way to rewrite a promise.

20 Decision rules

Rule: wire dependencies for everything the team can articulate as "X waits on Y." Four types (finish-start is 90% of life; start-start for parallelizable work; finish-finish and start-finish for the rare rest) plus lag days. Adding a dependency flips the successor to auto - that is the point.

Cost of choosing wrong: a schedule with hand-set dates and no logic must be re-fitted by hand after every slip, so it stops being re-fitted, so it stops being believed. A schedule with fake dependencies ("everything blocks everything") re-flows into nonsense. Wire what is true, only what is true.

Rule: one schedule per team-that-plans-together; hand-offs as cross-schedule dependencies. If two groups argue about working days, ownership, or approval, they want two schedules. Keep each schedule small enough for its lead to defend every line, and connect them only at the true hand-off points - usually a milestone on each side.

Cost of choosing wrong: the 200-item monolith is owned by everyone and maintained by no one; siloed schedules with no cross-links drift until the install crew discovers the slip in a hallway. Both failures look like "scheduling doesn't work here" - the fix is structural, not more discipline.

Rule: pick your view by the job, and know that dependency editing lives in Cards. Cards to build and wire; Grid to bulk-review; Calendar to talk to people who live in weeks;

Gantt to see the shape and the slack. They are lenses on one schedule, not competing schedules.

Rule: approve when people outside the room will rely on it - and treat revoked approval as information. A schedule only the builders read can live in draft happily (drafts are hidden from collaborators by default; there is a toggle when you want early visibility). The handshake matters the day a vendor, a client, or another team plans against your dates. When an edit knocks the schedule back to draft, that is the tool telling you the reliers need to hear about the change - re-approve *after* the conversation, not instead of it.

Rule: let overdue be visible. The health metrics (score, critical-path health, overdue, completed) are diagnostics, not grades. An overdue item surfaced early is the cheapest problem you will have all week; an overdue item hidden by quietly re-dating it costs you the schedule's credibility, which is the only currency a schedule has.

⚙ **Make it yours** - per schedule: working days (a five-day shop and a seven-day install crew can coexist as separate schedules), a display color, privacy for sensitive planning, draft visibility to collaborators, and a switch to block incoming dependencies from other schedules when a schedule must stay self-contained. The org-wide All Schedules view (filterable by program) reads across every schedule in the portfolio. The one thing that never bends: edits to an approved schedule revoke the approval. That is the scaffolding.

21 Troubleshooting

- **"I can't drag this item's dates."** It has a dependency - it is auto. Change the predecessor, the lag, or remove the dependency if the logic was wrong. Don't fight

derived dates; fix the derivation.

- **"The schedule went back to draft and nobody knows why."** Someone edited an approved schedule - the revocation is automatic and the edit history says who. Batch remaining changes, talk to whoever relies on the dates, re-approve once.
- **"Our schedule is 200 items and unreadable."** That is two or three schedules wearing one name. Split by team (see "Many schedules, one web of logic"), keep the hand-offs as cross-schedule milestone dependencies, and let All Schedules be the overview.
- **"Mobile shows the schedule but I can't edit it."** By design - phones read schedules and complete items (the site gesture that matters); construction of logic stays on the web where the views live.

> **Mechanics** - creating schedules and items, the dependency modal, baselines, approval requests, and the four views are covered click-by-click in the in-app **Field Guide → Schedules**.

22 The admin's half: schedules across the company

Approval discipline is a culture you set, not a switch: decide who approves (the approver need not be the builder - an outside eye catches wishful logic), and defend the revoke-on-edit behavior when teams complain about it. The complaint is always "re-approving is annoying"; the alternative is approved schedules nobody can trust.

The org-wide **All Schedules** view, filtered by program, is your portfolio lens - the place to see one program's projects converging on the same season, and to notice the project whose schedule is still in draft three weeks before its dates matter.

> ⚙ **Make it yours** - schedule access is part of the per-project permission grid and your permission templates (see the Members chapter): crews that live in the schedule get *edit*, clients usually get *view* of approved schedules only (the draft-visibility toggle stays off), and the reviewer-style middle ground is rarely needed here. Set the template once; stop deciding per person.

Part 4 - Logs and entry templates

The daily log is the project's memory of what actually happened - weather, work performed, who was on site, what went wrong - written the day it happened, when it is cheap, instead of reconstructed in a dispute, when it is expensive. It is the least creative tool in Ceruleo, and that is its virtue: by the time a project is pouring concrete and hanging steel, the non-linear phase has funneled into days that need faithful recording. The judgment here is almost entirely *front-loaded* - the structure of the record is decided before day one, and then the days just fill it.

23 The mental model

Three layers, deliberately separated:

- **A company log template** defines the *kinds of entries* a day can hold - "Work performed," "Weather," "Safety incident" - each with its own settings: structured fields or free text, whether photos and files attach, whether an entry type can appear only once per day, and - the important one - whether an entry of this type **raises a flag**.
- **A project log config** adopts a template into one project, names the log ("Aurora Site Log"), and can extend it with project-only entry types for the peculiarities of this job - without touching the company template every other project uses.
- **Daily logs** are then just dates: a log for the 20th, entries accumulating through the day, and a lifecycle of exactly two states - **draft** while the day is being written, **issued** when the record is distributed and sealed.

The flag-type entry is where the log stops being passive. "Safety incident" configured as a flag-type means the act of *recording* the incident also *raises* it - the entry lands in the day's record and a flag opens in the project's issue pipeline, with the log as its provenance. The log observes; the flag pursues.

24 The scenario

Avery built the **Site Daily Report** template months before Aurora Pavilion existed: work performed (free text, attachments on - site photos live here), weather (once per day, because two weather entries is a data-entry argument, not data), and safety incident (flag-type, non-negotiable). When the project went to site, Morgan adopted it as the **Aurora Site Log** in minutes - the thinking had already been done.

Now the rhythm: Riley - the fabrication vendor, phone in hand - records the day. "Queue rail sections 1 through 4 welded and staged," photos attached, "clear, 78°F, no delays." Morgan reviews and issues the log that evening; the 20th is sealed and distributed. The 31st sits in draft with the powder-coat delay already noted - tomorrow's issue, today's honesty. Months from now, when someone asks why the rails slipped, the answer is not in anyone's memory; it is in log #2, dated, attributed, and sealed.

25 Decision rules

Rule: design entry types for the person filling them in at 5 PM with dirty hands. Structured fields when you will *query* the answer later (crew counts, temperatures); free text when you won't; attachments on wherever photos are the real record; once-per-day for facts that are facts. Every field you add is a tax on every day of the project - charge it only where the data pays rent.

Cost of choosing wrong: an over-engineered form gets filled with "n/a" until people stop opening it; an under-engineered one ("Notes: text box") records a project you cannot search when it matters.

Rule: make anything that demands follow-up a flag-type entry. Safety incidents, damage, visitor issues - if recording it should *start* something rather than merely note it, wire the flag at the template level so the escalation is automatic, not a memory.

Cost of choosing wrong: incidents recorded but never pursued - the log says it happened, nobody's plate says it matters.

Rule: issue daily, backfill honestly, reopen loudly. A log issued the same evening is a record; a week of drafts issued together is a reconstruction wearing a record's clothes. Backfilling a missed day is supported and legitimate - dated as the day it describes, created when it was created. Reopening an issued log is possible (admin-level) and leaves the reopen on the record - treat it as a correction with a paper trail, never a quiet rewrite.

Cost of choosing wrong: the first time a sealed record turns out to have been quietly edited, every sealed record in the project becomes negotiable.

⚙ **Make it yours** - the template family is the company's (see the Company Admin edition), but each project's config can add project-only entry types for this job's realities - a themed-water-feature project can log water chemistry without the company template ever knowing. `Allow flags` and flag persistence are per-config choices. Capture is phone-first by design (the seeded rhythm: field crew drafts on mobile through the day, a lead reviews and issues on the web at night - issuing is deliberately web-only). Pocket captures (Part 5) attach straight into log entries, so the photo taken at 10 AM finds the 5 PM entry without a cable in sight.

26 Troubleshooting

- **"The log won't let me add a second weather entry."** Once-per-day is set on that type, and it is protecting you - edit the existing entry instead.
- **"Someone recorded an incident and nothing happened."** Check whether the entry type is flag-wired. If it is, the flag exists - chase it in Flags. If it isn't, that is a template gap: fix the type, not the person.

- **"We missed three days."** Backfill them now, dated correctly, and issue them. An honest gap filled late beats a fabricated continuity - and the creation timestamps keep everyone honest anyway.
- **"I need to change an issued log."** Reopen it (your project admin can), make the correction, re-issue. The reopen is part of the record - that is the feature, not the embarrassment.

> **Mechanics** - template building, project configs, entry capture on mobile, issuing and distribution, and reopening are covered click-by-click in the in-app **Field Guide → Logs**.

27 The admin's half: the template family

Log templates live in company settings, and their design is the highest-leverage twenty minutes in this part - every project that adopts a template inherits your choices for its entire site life. Build one general-purpose site template well (the scenario's three types are an honest minimum), and resist per-project template forks: the extension point for job peculiarities is the *project config's own entry types*, which is exactly what it is for. Fork the company template only when a whole *category* of project (interiors vs. site builds) needs a different record.

Retire templates by deactivating, not deleting - projects mid-flight keep their adopted structure, and your library stops offering the stale option to new ones.

> ⚙ **Make it yours** - flag-type wiring is where you encode company policy ("safety incidents are never just notes") so it executes without supervision. Distribution lists, reopen rights, and which grid level can issue are per-project decisions that your permission templates should already answer.

Part 5 - Everything else

The remaining tools are smaller, but each hides exactly one judgment call. Here they are, one page at a time. Mechanics for all of them: in-app **Field Guide**, section by section.

28 Flags, RFIs, and Notes - three shapes of "something needs saying"

A **flag** is a problem with a lifecycle - it stays open until someone resolves it, and it nags by existing. An **RFI** is a question with an official answer - it exists to get a response on the record from whoever owes it. A **note** is knowledge with no demand attached - it informs whoever finds it. The test is what "done" looks like: fixed → flag; answered → RFI; read → note. *Cost of choosing wrong:* problems filed as notes die quietly; questions filed as flags never capture who answered what; knowledge filed as RFIs demands answers nobody owes. Place all three on their elements (Part 1) - pre-filled automatically when you create from an element page.

Typed notes: if your company defines note types (site visit report, client call summary), a typed note carries structured fields worth querying later; an untyped note is a page. Same rule as log entries - structure only where the data pays rent.

29 Meetings - agendas that become minutes

A meeting here is a lifecycle, not a calendar invite: build the agenda in advance, send it (attendees get the PDF and a calendar file - external guests participate by email without accounts), run the session with roll call and per-topic notes and decisions, then issue the minutes - sealed, numbered, distributed. Amendments after issue are explicit revisions, never quiet edits - the same sealed-record ethic as logs. A **series** carries open and persistent topics forward with their numbering intact, so item 3.01 means meeting 3, topic 1, forever *(recurring*

project rhythms deserve a series; everything else doesn't). Action items raised in a meeting are real tasks on real plates - see next.

30 Tasks - one plate, many sources

Tasks flow in from everywhere - created by hand, raised in meetings, spun out of rallies, generated by review requests - and land on one plate per person. The judgment: a **manual task** stands alone; an **action item** created inside a meeting or rally stays traceable to its origin. Prefer raising work *inside its context* so the container's record shows what it produced; reach for a manual task when the work has no container. An unplaced task means "whole project" by design - that is a meaning, not an omission.

31 Library vs Pocket - filing vs catching

The project **library** is deliberate filing: durable documents in a folder taxonomy, uploaded because someone will need them. **Pocket** is the catch-all in your pocket: photos and captures grabbed in the moment, sorted later. The crucial asymmetry: placing from Pocket is **destructive and final** - the capture moves to its destination (a flag, a spark, a sandbox, a rally post, a log entry) and unplaced captures **expire**. Pocket is a funnel, not storage. *Cost of choosing wrong:* treat Pocket as an archive and your photos evaporate; treat the library as a dumping ground and the signal drowns.

32 Submittals - the formal yes

When work needs a documented approval chain - samples, shop drawings, cut sheets - a submittal runs the review, records each reviewer's verdict, and distributes the outcome. It overlaps studio reviews (Part 2) on purpose; the difference is formality and audience: studio reviews pressure-test *creative work in progress*; submittals certify *deliverables* to stakeholders. If the result belongs in a contract file, it is a submittal.

33 Discussions, mentions, and #links

Every item carries its discussion, and the whole project has one. Two habits multiply their value: **@mention** the person you need (that is what pings them - writing their name does not), and **#link** the item you reference - the chip carries readers straight to the flag, rally, or schedule item you mean. #links never notify anyone; @mentions do. Threads support replies-on-replies rendered flat - jump chips carry you to the parent.

When the whole topic needs to hear it, **@team** notifies everyone on the item - its creator, its assignees, and its collaborators, the same people you see in the avatar stack on the detail page. It deliberately does *not* ping project or company admins who aren't otherwise involved, so it is safe to use without spamming leadership. The judgment: @team is for the moments the item's whole cast must act or know - a decision made, a direction changed, a deadline moved. Used as a louder @mention it trains everyone to ignore it, and an ignored @team is worse than none.

34 Notifications and watching

You are notified for what you are *part of* - assignments, mentions, replies, reviews, meetings - and you can **watch** anything to opt into its life. The discipline is pruning: watch what you would act on, unwatch what you were merely curious about, and tune per-channel (push vs. bell vs. daily digest) in your profile settings rather than learning to ignore the bell. An ignored bell is no bell.

35 Programs and statuses - the portfolio lenses

Projects can belong to **programs** (a season, a client, a campus - many-to-many, so one project can sit in several) and carry a company-defined **status** whose names and colors are your company's vocabulary. Neither changes anything inside a project - they are lenses for reading across projects. Use programs for *belonging*, statuses for *state*.

36 Company Sparks

> ⚑ **Company users only.** Project-invited collaborators do not see the company Sparks pool - their window starts at project sparks inside sandboxes they can access.

The porch from Part 2, company-wide: ideas with owners, waiting for their project. The rhythm that keeps it alive (convert, keep, or archive - on a schedule) is described there.

37 Company contacts

> ⚑ **Company users only** - and visible only to members whose contacts permission allows it.

The company rolodex: people and businesses, faces and logos, tags, attachments, and an activity log of dated interactions. Project contact lists draw from it; a contact who becomes a real user links to their profile automatically. Log interactions when they happen - a rolodex without history is a phone book.

38 Your profile, your phone, your theme

Push preferences, digest timing, avatar, pronouns, timezone - and appearance: the web app ships a light mode built for readers who live in documents (Settings → appearance). The mobile app is a first-class citizen for the *field half* of the platform - capture, logs, completing schedule items, reading everything - while heavy construction (schedule logic, markup, issuing) stays on the web.

⚙ **Make it yours** - nearly everything in this part is shaped by company settings: note types, flag types, submittal setup, contact categories, program and status vocabularies, and the per-project permission grids that decide who sees which tabs at all. The Company Admin edition walks each knob.

39 The admin's half

Every "⚙ Make it yours" note in this part resolves to a company-settings page you own: template families (note types, flag types, spark categories, asset types), programs, project statuses, contact categories, and the permission templates that stamp new members' grids. The pattern across all of them is the same one this manual keeps repeating: short vocabularies chosen so fast nobody mis-files, templates that carry the sketch rather than the answer, and defaults that make the right habit the lazy habit. The dedicated chapters that follow walk each settings surface in setup order.

Members, invitations, and capabilities

The welcome chapter called the company-vs-project invitation the one distinction that governs everything. This chapter is its operating manual.

40 The two invitations, mechanically

- **Company invitation** (company settings → members): creates a company user in your workspace. They see company tools, can be granted capabilities, and can then be added to any project. One company per person - an invitation to someone already in another company will be refused, not merged.
- **Project invitation** (project → team): creates a collaborator on that project only, with a per-tool permission grid. It never creates company membership, and no amount of project work promotes anyone - promotion is always an explicit company invite from you.

When in doubt: *is this person's relationship with the company, or with a project?* Employees and long-term contractors → company. Clients, vendors, venue staff, one-project partners → project.

41 What a project-only collaborator actually sees

Their projects' contents per their grid, plus the cross-project pages those feed (their tasks, flags, schedules), Pocket, and the Field Guide. They do **not** see: program names, status vocabulary, company contacts, company Sparks, or your company's name and branding surfaces. If a collaborator reports "missing" tools, check membership before permissions - it is the answer far more often.

42 Per-member capabilities

Company membership is the floor; capabilities are the individually granted extras: **create projects**, **manage templates**, **manage the company library**, **manage permission templates**, **contacts** (view - on by default - edit, and manage), and explicit **Sparks** participation where you run Sparks opt-in. Grant by role-in-fact, not title: capabilities describe what someone *does*, and they compose - a studio lead might hold templates and library; an office manager, contacts and nothing else.

43 Permission templates

⚙ **Make it yours** - the per-project grid (tool-by-tool: none / reviewer / view / edit / admin) is powerful and tedious, which is why permission templates exist: define "Client," "Fabricator," "Full team" once, and stamp new project members instead of hand-setting ten switches. Templates are a starting stamp, not a live link - adjusting one member afterward never rewrites others.

Rule: invite to the narrowest true relationship; widen on evidence.

Cost of choosing wrong: the over-invited vendor browses your idea pipeline and client list; the under-invited employee works for months not knowing half the platform exists. The first is a disclosure you cannot un-ring; the second is capability you paid for and never used.

Mechanics - invitation flows, grids, and templates: **Field Guide → Team & permissions**.

Template families

Company settings holds a family of template surfaces - **structure templates** (Part 1), **log templates** (Part 4), **note types**, **flag types**, **spark categories**, and **asset categories & types** (Part 2). Six lists, one philosophy, stated once here so the individual pages don't repeat it:

Templates are scaffolding, not answers. Every family exists to make the right habit the lazy habit - the create dialog that already offers the right categories, the project that starts with the right sketch. The moment a template starts *deciding* things the work hasn't decided yet, it has crossed from scaffolding into cage. The work drives the shape; templates just make the shape cheap to adopt.

Three operating rules that apply across the family:

> **Rule: short lists get chosen; long lists get defaulted.** Every vocabulary (note types, flag types, spark categories, asset types) should be short enough that the right choice takes no thought. Past roughly seven options, people stop choosing and start defaulting to the first entry - and your structured data becomes noise wearing structure's clothes.

> **Rule: adopted copies don't follow the template.** Projects copy templates at adoption (structure at creation, log configs at setup). Editing a company template improves *future* adoptions, never running projects - so template investment compounds forward and can't break anything mid-flight. Corollary: fix a running project *in the project*; fix the pattern *in the template*; usually you should do both.

> **Rule: retire by deactivating, never deleting.** Deactivation removes an option from new adoptions while everything already built on it keeps working. Deletion is for things that never shipped.

Each family's specific judgment lives with its tool: structure templates in the Organizer chapter, log templates in the Logs chapter, spark and asset vocabulary in Sandboxes and Rallies, note and flag types in Everything Else. This chapter's job is only the shared doctrine - and the reminder that **manage templates** is a per-member capability (chapter 06), so template stewardship can live with your best organizers rather than with whoever happens to be admin.

> **Mechanics** - each family's settings page: **Field Guide → Company settings**.

Programs, statuses, and the project lifecycle

These are the portfolio's vocabulary - how the company reads across projects without opening them.

44 Programs

A program is a belonging: a season, a client relationship, a campus, a tour. Projects join programs many-to-many - the pavilion can sit in both *Pavilion Series 2027* and a client's program - and membership changes nothing inside the project. Programs pay off in the cross-project views (All Schedules filters by program) and in your own head: they are how "how is the 2027 season doing?" becomes a question the tool can answer.

> **Rule: create programs for questions you actually ask.** A program nobody filters by is a tag with a salary.

45 Project statuses

> ⚙ **Make it yours** - this is one of the most visible customizations in the platform: your status names, your colors, shown on every project card. Each custom status *behaves as* one of four underlying categories - active, on-hold, completed, or archived - so the platform always knows what "Tentative" or "Confirmed" *means* even though the vocabulary is yours.

The behaves-as category does real work: "open projects" everywhere in the platform means the living categories - archived and completed projects drop out of default views, pickers, and

counts without being deleted. So choose the category honestly: a status that *reads* dormant but *behaves as* active keeps stale projects cluttering every list; the reverse quietly hides live work.

> **Rule: statuses describe state, programs describe belonging - never encode one in the other.** A "2027 Season" *status* would force projects to choose between saying where they belong and saying how they're doing.

46 Lifecycle

Projects end. Completed and archived projects keep their full history - the element dashboards, the sealed logs, the issued minutes remain the company's memory - while leaving the daily views. Archive promptly when work truly stops: the open-projects convention only helps if "open" stays true. Deletion is the rare, deliberate exception, and the company settings delete log records what was removed, by whom, when - the same paper-trail ethic as everywhere else in the platform.

> **Mechanics** - program and status CRUD, lifecycle settings, and the delete log: **Field Guide → Company settings**.

The company library and contacts administration

The last two settings surfaces are the company's shared shelves: documents and people.

47 Company library

The company library is the durable, cross-project shelf - brand assets, standard details, boilerplate, certifications - organized in a folder taxonomy, distinct from each project's own library. Curation is a granted capability (**manage library**, chapter 06), and the same filing ethic as project libraries applies at higher stakes: this shelf is every project's upstream, so a mess here is inherited everywhere.

> **Rule: the company library holds what outlives projects.** If a document belongs to one job, it belongs in that project's library; if the next three jobs will need it, it belongs here.

48 Contacts administration

The company rolodex - people and businesses with faces and logos, tags, file attachments, and a dated **activity log** of interactions. Project teams draw on it when they add project contacts, and when a contact later becomes a real platform user, the platform links the contact to their profile automatically (matched by email; the live profile wins from then on).

Access is deliberately layered *(chapter 06)*: viewing contacts is on by default for company members but revocable per member; editing and managing are separate grants. Project-only collaborators never see the company rolodex at all - it is structurally company-side, like Sparks.

Rule: log interactions when they happen, attach what was exchanged. A rolodex is only as valuable as its history - "who talked to this vendor last, and what did we promise?" should be a lookup, not an archaeology.

⚙ **Make it yours** - contact tags and categories are your vocabulary (same short-list discipline as every other family), and per-member contact permissions let you split the difference between "everyone can look someone up" and "two people own the client list."

Mechanics - library folders and uploads, contact cards, tags, attachments, and the activity log: **Field Guide → Library / Company contacts**.