This is a copy of a conversation between ChatGPT & Anonymous.
Report conversation

In quick suite (aws) you can research on it. There are two types of integrations, one is oauth and other is service to service calls, how do you look at it

I'll answer this **conceptually and architecturally**, not guessing undocumented features — because your question is really about **how to think about integrations in AWS QuickSuite–style products**, not about one checkbox in the UI.

You're essentially looking at **two very different trust and control models**.

1️⃣ OAuth-based integrations (User-delegated)

What it means

OAuth = **a human user authorizes QuickSuite to act on their behalf**.

Think:

- "Log in with Asana"
- "Connect Jira account"
- "Allow QuickSuite to read/write X"

Trust model

- **User-centric**
- Permissions scoped to *what that user can do*
- Tokens expire / refresh
- Revocable by the user

Typical use cases

- Reading/writing **user-owned objects**
    - Asana tasks
    - Jira tickets
    - GitHub issues
- Multi-tenant SaaS integrations
- Anything that must respect **per-user permissions**

Pros

✅ Least privilege (per user)
✅ Auditable ("Rehan created this task")
✅ Required by most SaaS providers
✅ Safer for external integrations

Cons

❌ Token lifecycle complexity
❌ Breaks when user leaves org
❌ Slower for high-volume automation
❌ Harder for background/batch jobs

In QuickSuite terms

OAuth is **inevitable** if:

- You want **2-way sync with Asana**
- You want to show *“tasks created by you”*
- You want compliance-grade auditability

If QuickSuite claims “Asana integration” without OAuth → that’s a red flag.

2️⃣ Service-to-Service (Machine identity)

What it means

QuickSuite talks to another system **as itself**, not as a user.

In AWS terms:

- IAM Role
- STS AssumeRole
- API keys (external systems)
- mTLS / SigV4

Trust model

- **System-centric**
- Long-lived or rotated credentials
- Broad, pre-approved permissions

Typical use cases

- Internal AWS services
- Data ingestion pipelines
- Metrics ingestion
- Event streaming
- Background automation

Pros

✅ Extremely reliable
✅ High throughput
✅ No user dependency
✅ Ideal for batch / async workloads

Cons

❌ Harder to scope precisely
❌ Riskier if misconfigured
❌ Usually **not accepted by SaaS vendors**
❌ Poor per-user attribution

In QuickSuite terms

Service-to-service is the **correct choice** for:

- Storing campaign performance data
- Ingesting metrics/events
- Automation workflows
- Cross-account AWS access

This is where **IAM + STS** should dominate.

3️⃣ How I'd look at this *specifically for QuickSuite*

Correct mental split

QuickSuite should **NOT choose one**. It should use **both**, very intentionally.

Capability	Integration Model
Asana task sync	OAuth
Jira tickets	OAuth
GitHub issues	OAuth
User actions	OAuth
Workflow engine	Service-to-service
Metrics ingestion	Service-to-service
Campaign performance	Service-to-service
Internal AWS data	Service-to-service

4️⃣ If you're designing against QuickSuite (important)

Since you said:

> *"I have knowledgebase, templates, teams information and time estimates"*

That tells me you want **automation + planning + execution**, not just dashboards.

The correct architecture looks like this:

User Plane

- OAuth tokens per integration
- Stored encrypted (Secrets Manager)
- Scoped per workspace / team
- Used only when a user-triggered action occurs

System Plane

- IAM role for QuickSuite
- Event-driven ingestion (EventBridge / Kinesis)
- Metrics written via service identity

- No dependency on user presence

This separation is **non-negotiable** if you want scale.

5️⃣ Red flags to watch for in QuickSuite docs

Be cautious if:

- They say “Asana integration” but no OAuth
- They only support OAuth and no machine ingestion
- They blur user actions and system automations
- They store long-lived OAuth tokens without rotation

Those are early-stage SaaS mistakes.

6️⃣ Bottom line (blunt take)

- **OAuth is for people**
- **Service-to-service is for systems**
- Any serious AWS-native product (including QuickSuite) **must support both**
- If QuickSuite forces OAuth for metrics → bad design
- If QuickSuite forces service credentials for SaaS → impossible in practice

If you want, next we can:

- Map this directly to **Asana 2-way sync**
- Design the **exact token + role model**
- Or compare this with how **Amazon Q / QuickSight actually do it internally**

Let’s solve it like where I want to read all projects and get performance data

If your goal is: **“Read all projects and pull performance data”** in Amazon **Quick Suite**, here’s the clean way to “solve” it using the two integration types (OAuth vs service-to-service) — and why you almost always need **both.**

Amazon Quick Suite explicitly supports **user auth (custom OAuth), managed OAuth 2.0, service auth (API key or service-to-service OAuth), and AWS credentials** as integration auth methods. AWS Documentat...

The winning architecture: split it into two planes

A) Work-management plane (Projects/Tasks) → OAuth (or Managed OAuth)

Why: “Projects” live in tools like Asana/Jira/Linear, and those platforms generally require user-delegated authorization (or an org-approved OAuth app) to enumerate projects and read their contents.

For Asana specifically, “Get projects for workspace” requires the `projects:read` scope. Asana Docs

And Asana scopes exist specifically to keep permissions least-privileged. Asana Docs

How this looks in practice

- One "integration connection" per workspace/org (or per user if you're not doing org-wide)
- Token storage + refresh handled by Quick Suite's integration connector
- You sync:
  - Workspaces/Teams
  - Projects
  - Tasks (and key fields you need)

Outcome: You can reliably "read all projects" *that the authorized identity has access to*.

> Note: Even "super admin" doesn't always mean "API can list literally everything" depending on how the SaaS models membership/visibility. This is a common gotcha when someone expects 9k projects but the endpoint returns fewer. Asana Forum

---

B) Performance-data plane (Metrics/Events) → Service-to-service

Why: Performance data is high-volume, periodic, and should not break when a user leaves or revokes OAuth.

Quick Suite supports **service auth** (API key or service-to-service OAuth) and **AWS credentials** for integrations. AWS Documentat...

How this looks

- Your trackers / pipelines push metrics to a storage layer using a **machine identity**
- Then Quick Suite reads from that storage for dashboards, agents, and automation

Quick Suite is designed to connect to your data sources/APIs and backend systems via "agent actions," and has **action connector APIs** to manage these connections programmatically. AWS Documentat... +1

So what does "read all projects and get performance data" become?

Step 1 — Ingest all projects (OAuth)

- Connect Asana (or Jira etc.) via OAuth/Managed OAuth
- Pull:
  - `workspace_id -> projects[]`
  - `project_id -> tasks[]`
- Persist a normalized model like:
  - `external_project_id` , `external_task_id`
  - owner/team, status, due dates, etc.

Step 2 — Define the join key between "project" and "performance"

You need an explicit mapping, otherwise "performance data" has nothing to attach to.

Common patterns:

- Put a `campaign_id` or `tracking_key` custom field on the project/task
- Or enforce naming conventions (“CP-12345”)
- Or maintain a mapping table in your DB

Step 3 — Ingest performance metrics (service-to-service)

- Your tracking system writes metrics keyed by `campaign_id` (or whatever you chose)
- This data lands in AWS storage you control (S3 + Athena/Glue, Redshift, DynamoDB, Timestream—depends on query style)

Step 4 — Quick Suite consumes and acts

- Dashboards over the metrics
- Agents answer questions
- Automations trigger actions (e.g., “if CPA > X, create task in Asana”)

(Quick Suite’s positioning is explicitly “insights + actions/automation,” not just BI. Amazon Web Ser... +1)

The key decision rule (simple)

- **If the source of truth is a human SaaS workspace (projects/tasks): use OAuth**
- **If the source of truth is machine-generated metrics (events/performance): use service-to-service**

What I’d do for your exact intent (most robust)

1. **OAuth** integration to enumerate projects/tasks (Asana/Jira/etc.)
2. Require a `campaign_id` field on each project (enforced by template)
3. **Service-to-service** metrics ingestion keyed by that `campaign_id`
4. Quick Suite dashboards + agent actions + workflow triggers on top

If you tell me **where your performance data currently lives** (Voluum/Clickflare/your ClickPattern DB, GA4, Meta Ads, etc.), I’ll propose the most appropriate AWS storage + ingestion pattern and the exact “join key” strategy that won’t break in 6 months.

Sources

Think that director want to read performance, he wouldn’t have an account in asana

Right — if the **Director doesn’t have (and shouldn’t need) an Asana account**, then **OAuth “per user” is the wrong mental model** for the *consumer* of the data.

What you want is:

✅ “One org-level Asana connection → mirror the data → anyone can view in QuickSuite”

That’s a **producer/consumer split**:

- **Producer identity (integration)**: a single *service identity* that can read Asana.
- **Consumer identities (viewers)**: directors/executives who only authenticate to **QuickSuite**, not Asana.

The correct pattern

1) Use a dedicated “Integration Owner” identity for Asana access

This can be:

- A dedicated Asana “service user” (preferred), or
- An admin/ops user account specifically meant to own integrations

That identity performs OAuth once (or uses an approved org-wide method), and QuickSuite stores the token.

Key point: the director never touches Asana auth.

Access boundary: Director permissions are enforced **inside QuickSuite**, not in Asana.

---

2) Mirror what you need from Asana into your own data layer

You run a sync job (polling or events/webhooks depending on what’s supported) that pulls:

- workspaces/teams
- projects
- tasks + fields you care about (status, assignee, due date, custom fields like `campaign_id`)

Store it in something queryable (e.g., relational/warehouse) so QuickSuite can serve it fast and consistently.

This turns Asana into an **upstream source**, not a runtime dependency for every director dashboard load.

---

3) Performance data should be service-to-service anyway

Your performance metrics (CPA, ROI, spend, clicks, conversions, etc.) should come from:

- ad platforms / tracking system
- your clickstream/events pipeline
- your internal DB

That ingestion is **service-to-service** and keyed by something stable like:

- `campaign_id`
- `launch_id`
- `project_external_id`

---

4) Join projects ↔ performance using an explicit key

This is the part most teams mess up.

Best practice:

- Put a required `campaign_id` (or `launch_id`) **custom field** in the Asana project (or in your template)
- Enforce it in your process ("no campaign_id = not reportable")
- In your warehouse, join:

```
asana_project.campaign_id ↔ performance.campaign_id
```

Now the director can open QuickSuite and see:

Project → KPIs → trends → anomalies → actions, without any Asana login.

What this implies about your "two integration types"

OAuth is still used — but only once, by the integration owner

OAuth isn't "for every viewer."
It's for *establishing access to the upstream SaaS*.

Service-to-service is how you serve everyone at scale

Once mirrored, QuickSuite reads from *your storage* using service credentials.
Directors authenticate only to QuickSuite and get RBAC-controlled views.

Practical RBAC model for execs (simple and effective)

Inside QuickSuite, define:

- Director role: read-only dashboards + summaries
- Managers: read + drill-down + exports
- Operators: read + "create tasks/updates back into Asana" (if you want writeback)

Writeback to Asana should be restricted (because now you're acting as the integration owner identity).

The one trade-off you must choose

Do you want directors to see **everything**, or only a subset?

Since directors don't have Asana accounts, you **must encode visibility rules in QuickSuite**, typically via:

- project owner/team mapping
- business unit tags
- "portfolio" concept
- or allowlist sets

If you tell me **what "projects" mean in your world** (launch projects? campaigns? client accounts?) and where performance data lives (ClickPattern DB, Voluum, Meta/Google

APIs, GA4, etc.), I'll give you the exact schema + sync cadence + join keys + RBAC rules so it's clean and doesn't turn into a permissions nightmare later.

Now also think adobe workfront too in the mix for production teams

Yep — once **Adobe Workfront** enters the picture, the "one tool = one source of truth" assumption dies. You need a **hub-and-spoke integration** where QuickSuite becomes the *reporting + orchestration layer*, and Asana/Workfront are *execution systems* for different teams.

Here's the clean way to design it so a Director can read performance **without** needing Asana or Workfront accounts.

1) Decide the "system of record" per domain

Don't try to make both tools authoritative for the same thing.

A sane split:

- **Production execution (creative/ops)** → **Workfront = system of record**
  - briefs, assets, proofs, approvals, production statuses, dependencies
- **Launch/campaign execution (growth/marketing ops)** → **Asana = system of record**
  - checklists, launch tasks, coordination, handoffs
- **Performance truth (metrics)** → **your tracking/ads/analytics pipeline = system of record**
  - spend, CPA/ROAS, conversions, revenue, funnel KPIs

QuickSuite then **mirrors** both Workfront + Asana into a **canonical model** and joins to performance.

2) Use "integration owner" identities (not end-user accounts)

Same pattern you described for Asana applies to Workfront:

- Create a dedicated **integration identity** for each tool:
  - `asana-integration@…`
  - `workfront-integration@…`
- Authorize once (OAuth / service credentials depending on what each platform supports)
- Sync data into your storage
- Directors authenticate only to **QuickSuite**, never to Asana/Workfront

This is how you avoid "Director needs a license account" forever.

3) Build a canonical "Work Item" model in QuickSuite

You need a neutral schema that can represent either tool.

Minimum canonical objects:

- **Program / Portfolio** (optional)
- **Project** (launch / production initiative)

- **Work Item** (task/story/issue — maps to Asana task OR Workfront task)
- **Asset** (mostly Workfront: creative files, proofs)
- **Milestone / Stage** (conceptual phase)
- **Owner Team**
- **External Links** (deep links back to Asana/Workfront)

And crucially: a stable join key like `campaign_id` / `launch_id`.

The one thing that makes this work:

Every reportable project must carry a "Performance Key"

- `campaign_id` (for paid campaigns)
- `launch_id` (for product releases)
- `client_id` (for agency model)

Then you can join:

```
canonical_project.performance_key ↔ performance.performance_key
```

4) Two-way sync: be very strict about writeback

With two execution tools, writeback can get messy fast. You need a rule:

"Write back only into the tool that owns that domain"

Examples:

- Update creative proof status? → write to **Workfront**
- Create launch checklist tasks? → write to **Asana**
- Director comments? → stay in **QuickSuite** (or create "comment task" in the owner tool only if needed)

Avoid this anti-pattern

Don't mirror Asana → Workfront and Workfront → Asana as peers. That creates loops and conflicts.

Instead:

- Sync both **into QuickSuite**
- QuickSuite can create **cross-links** and "handoff tasks" when needed

5) Reporting views for Directors (no tool accounts needed)

Directors want a *portfolio view*, not task-level fidelity.

So QuickSuite should present:

- Portfolio / program rollups
- Stage health (On track / At risk / Blocked)
- Production readiness (from Workfront)
- Launch readiness (from Asana)
- Performance outcomes (from metrics pipeline)
- Exceptions + anomalies + "why" drilldowns

Permissions enforced inside QuickSuite:

- by business unit / client / program / team mapping
- not by Asana/Workfront membership

6) Data movement pattern that scales

- **Asana/Workfront**: periodic sync + incremental updates (webhooks if possible; otherwise delta polling)
- **Performance**: event/batch ingestion service-to-service (high volume)
- Store everything in a queryable layer (warehouse/lake + a fast serving layer if needed)

7) The critical decision you need to make now

Where does a “Project” *live*?

Pick ONE of these:

1. **Canonical project lives in QuickSuite** (recommended)
    - Asana/Workfront have “execution containers” linked to it
2. **Canonical project lives in Workfront**
    - Asana projects become “launch execution mirrors”
3. **Canonical project lives in Asana**
    - Workfront projects become “production execution mirrors”

If you don’t pick, you’ll end up with duplicate “project identity” and unreliable rollups.

If you tell me your real flow in one line, I’ll lock this into an exact blueprint:

- “Production starts in Workfront with briefs/assets/approvals, then launch tasks tracked in Asana, then performance from ClickPattern/Meta/Google.”

And I’ll give you:

- the canonical schema (tables + keys),
- the sync responsibility matrix,
- and the writeback rules that prevent infinite loops.

Now think tempaltes and knowledgebases are coming from S3, sharepoint and quip

Got it — now your **templates + knowledgebases** are coming from **S3, SharePoint, and Quip**, *in addition* to Asana + Workfront for execution, and you want directors to read performance without needing accounts in those systems.

This becomes a **content ingestion + permissions + indexing** problem as much as an integration/auth problem.

Here’s the clean way to think about it.

The correct “3-plane” architecture

Plane A — Execution systems (Asana + Workfront)

- tasks, projects, production workflow

- synced via an integration owner identity (as we discussed)

Plane B — Content systems (S3 + SharePoint + Quip)

- templates, SOPs, launch playbooks, QA checklists, creative guidelines
- these are "knowledge" sources you want to retrieve and cite in QuickSuite

Plane C — Performance systems (tracking/ads/analytics)

- metrics/events stored via service identity

Directors should only authenticate to **QuickSuite**; QuickSuite enforces RBAC.

Auth model per content source (and why)

1) Amazon S3 → Service-to-service (IAM role / STS)

S3 is AWS-native, so you want **temporary credentials via IAM roles**, not long-lived access keys. AWS IAM best practices recommend using roles/temporary creds for workloads. AWS Documentat... +1
S3 access is controlled via bucket policies and IAM policies. AWS Documentat...

Meaning: QuickSuite (or your ingestion job) assumes a role and crawls specific buckets/prefixes like:

- `s3://kb/templates/...`
- `s3://kb/playbooks/...`

2) SharePoint Online → OAuth, but usually "Application permissions" (service identity)

Microsoft Graph supports:

- **Delegated permissions** (act as a signed-in user) Microsoft Learn
- **Application permissions** (act as the app itself; admin-consented) Microsoft Learn +1

For your director use-case, you want **app permissions** (org-approved) so the integration is stable and not tied to an individual user.

3) Quip → typically Personal Access Token (service-ish) or OAuth token

AWS's own Quip connector overview notes Quip authentication as **Personal Access Token**, and that it supports ACL crawling. AWS Documentat... +1
Quip's API uses **Bearer access tokens**. Quip

Meaning: you'll likely use one controlled integration token (or OAuth-based access token) owned by an integration account.

The real hard part: permissions (directors without accounts)

You have two options. Pick deliberately.

Option 1 (most common): QuickSuite becomes the policy authority

- Ingest content from S3/SharePoint/Quip using service identities
- Store documents + metadata in your KB store/index
- Enforce access in QuickSuite via your own RBAC:

  - BU / team / client / program / region tags
  - “Director sees all,” or “Director sees BU-only,” etc.

Pros: directors don’t need source accounts; simplest experience
Cons: you’re not strictly honoring source permissions unless you mirror them

Option 2 (more correct in enterprises): Mirror source ACLs into QuickSuite

- Crawl documents *and their ACLs*
- Map identities (SSO groups) → QuickSuite users
- Enforce “who can see what” based on source permissions

This is why systems like Amazon Q Business support “ACL crawling” for certain connectors (e.g., Quip). AWS Documentat...

Pros: governance-friendly
Cons: identity mapping is real work, and Quip has limitations (no group concept per AWS’s connector overview). AWS Documentat...

If directors truly have **zero accounts** in those systems, Option 2 can’t fully work anyway (no identity to map). So you’ll usually end up doing **Option 1** with an internal RBAC model.

How to “solve it” end-to-end for your use case

1) Standardize metadata on every template / KB doc

Minimum:

- `content_type` (template, SOP, playbook, checklist)
- `team` / `business_unit`
- `workflow_stage` (production, launch, post-launch)
- `project_type` (campaign, product launch, client onboarding)
- `version`
- `owner`

Put these as:

- S3 object tags or sidecar JSON
- SharePoint columns / library metadata
- Quip folder conventions or doc header block

2) Ingest into a canonical Knowledge Store

Store:

- raw content
- extracted text
- embeddings (if you’re doing semantic retrieval)
- metadata + ACL/RBAC tags
- source pointers (deep links back to S3/SharePoint/Quip)

3) Retrieval strategy inside QuickSuite

When a director asks:

> “How is Campaign X doing and what’s blocking it?”

QuickSuite should do **multi-hop retrieval**:

- performance metrics for Campaign X
- project/work-items from Asana/Workfront linked by `campaign_id`
- relevant templates/SOPs for the current stage (e.g., "creative QA checklist")

The design choice you need to make now

Do you want knowledge/templates to be:

1. **Global assets** (directors can see all; teams see what they need), or
2. **Strictly source-permissioned** (mirror SharePoint/Quip ACLs)

Given "director has no accounts," you're almost certainly in (1).

If you answer these 2 things, I'll pin down the exact integration blueprint (with the minimal moving parts):

1. For SharePoint: do you have an Azure admin who can grant **application permissions** (app-only), or are you stuck with delegated OAuth? (Graph supports both. Microsoft Learn +1)
2. For Quip: is it **enterprise Quip** and are you okay using a single **integration token** as the "reader"? (That's how AWS's connector describes it. AWS Documentat... +1)

Either way, the core solution remains: **ingest + normalize + tag + enforce RBAC in QuickSuite**, not "make every viewer authenticate to every upstream system."

Sources

Now think few different usecases,

am an SMB marketer who wants to send a new email out to my audience to drive ASIN penetration. I know roughly what I want to send but need help refining the strategy. To start, I use the XYZ agent in QuickSuite. I enter some basic details on what I want to do; the agent asks me some refinement questions, then spits out a strategy for my campaign. I know this already captures the standards of our campaign process. I make a few revisions based on my knowledge of the audience, and I tell the agent to submit it. This then opens a campaign in Workfront, with all the correct information. I am then able to track progress against this campaign in my QuickSuite XYZ dashboard, and can talk in natural language to the agent, which copies decisions over to Workfront for the production teams.
• I am a PMM who is launching product A the week of unBoxed, a Tier 2 launch. I need to create a set of marketing materials. My product is in Launchpad, and I see a step in my LP action items to open my project in Workfront, and I press the link. This pulls all my launch details from LP into a chat with an agent in QuickSuite. It lets me know I am missing some details, and asks me to

upload my comms doc. I do this. The agent then asks me a few more questions to fill in the gaps. It then proposes a campaign strategy for the launch for me to review. I like what it’s created and ask it to create the request. This writes to Workfront and then spits out a list of next steps I need to take, and due dates, so I can make the launch date.

- I am an Event marketer planning to execute an activation at Cannes 2026, a Tier 1 event. I need to coordinate presence, speaking sessions, sponsored activities, and pre/post-event campaigns, all necessary steps in the event development. I open the XYZ Agent in QuickSuite and enter basic details: event name, dates, tier level, and high-level goals. The agent asks me refinement questions and then analyzes my inputs against the standard event marketing playbook and proposes a strategy. I review the strategy, adjust based on my knowledge of the audience and event specifics, then tell the agent to "create the project." This automatically opens a campaign in Asana with all the correct information, pre-populated tasks, assigned teams, and dependencies. Throughout the event planning, I can track progress in my QuickSuite XYZ Dashboard, and I can ask the agent questions in natural language like "Are we on track for our booth creative delivery?", and the agent syncs decisions back to Asana. Post-event, the agent automatically triggers the follow-up workflow, pulling lead data and engagement metrics to inform the nurture campaign strategy, which I can review and approve before execution. My agentic support ensures that I hit every step and that the quality of my output is comprehensive and informed by previous work and learnings.
- I am a marketer on the ME&I production team executing marketing communications for campaigns. When a new request comes through Workfront, my XYZ Agent in QuickSuite ingests all campaign details and proposes a production plan with asset specifications, timeline based on my capacity, dependencies, template recommendations, and approval workflow. I review and tell it to "confirm and start production." The agent updates Workfront with delivery dates, notifies the requester, and then kicks off the required production flows. It notifies me when the drafts are ready for review. As I look at them, I ask questions like "What's the latest brand guidance on AI-generated imagery?" When ready, I say, "submit for approval" and the agent routes to stakeholders, consolidates feedback, and prioritizes revisions. The QuickSuite XYZ Dashboard shows capacity utilization, upcoming deadlines, and bottlenecks. The agent proactively alerts: "You have 3 requests due this week, but Product X assets are blocking 2 downstream campaigns. Notify requesters of delay?" Post-delivery, the agent logs asset metadata, tags files in the DAM, and captures performance data for

future decisions.

- I am a Partner marketer responsible for organizing our annual Partner Awards event. I open the purpose-built XYZ Agent in QuickSuite and say, "I need to launch the 2026 Partner Awards program." The agent asks me refinement questions about award categories, nomination criteria, timeline, judging panel composition, and ceremony details, then proposes a comprehensive Partner Awards project plan based on our standard recognition playbook. I review the strategy, adjust the award categories based on this year's strategic priorities and agency feedback, then tell the agent to "create the project." This automatically opens a project in Asana with all phases pre-mapped. Throughout the awards cycle, I can track progress in my QuickSuite XYZ Dashboard, and I can ask the agent questions in natural language like "How many nominations have we received for the Creative Excellence category?" The agent syncs all updates back to Asana and automatically notifies agencies when nominations open, reminds them of deadlines, and confirms submission receipt. As the ceremony approaches, the agent helps me prepare by generating winner announcements, creating presentation materials with nominee highlights and performance data, and coordinating logistics. Post-event, the agent automatically archives all submissions, captures winning campaign insights to inform best practices, and summarizesparticipation metrics and feedback to improve next year's program.
- I am a director on the Marketing leadership team who needs to understand what is happening across the entire marketing organization and the status of each campaign without getting lost in operational details. I open my XYZ Dashboard in QuickSuite and immediately see a real-time overview of all active campaigns, budget utilization, performance against goals, and risk indicators. I can ask the XYZ Agent questions in natural language like "What campaigns are at risk of missing their launch dates?" or "Which campaigns are underperforming their KPIs?" The agent pulls data from Workfront, Asana, and our performance systems to provide instant answers with visualizations. When I need to go deeper, I can say "Tell me more about the unBoxed campaign" and the agent provides a comprehensive brief: objectives, timeline, team leads, status, blockers, budget vs. actuals, and early performance indicators. If I see an issue, I can ask the agent to deep dive into a campaign, and it identifies the bottleneck, shows dependencies, and suggests mitigation options. I can also ask strategic questions like "Compare our event marketing ROI across the last three quarters" or "What percentage of our campaigns are using AI-generated content?" and the agent analyzes patterns across all campaigns to inform strategic decisions. I can also see business health metrics, like the

% of campaigns that have briefs attached, which have quadrupled against our pre-agent baselin

This is a **very coherent vision**, and importantly: all of these use cases are *the same system* viewed through different **roles, entry points, and authority boundaries**.

I'm going to do three things:

1. **Normalize all these stories into a single agentic system model**
2. **Show how QuickSuite, agents, Workfront, Asana, KBs, and performance fit together without contradictions**
3. **Call out the non-obvious design constraints you must get right (or this collapses)**

I'll be direct and slightly critical where needed.

---

1. One system, many personas (don't overfit per use case)

All six use cases resolve to the same core loop:

> **Intent → Strategy → Structured Plan → Execution System → Continuous Sync → Performance → Learning**

The difference is:

- *who initiates*
- *which execution system*
- *how much autonomy the agent has*
- *how much detail the human sees*

So don't design 6 agents.
Design **one orchestration fabric with role-scoped agents**.

---

2. The canonical architecture (this is the backbone)

A. Canonical objects (non-negotiable)

If you don't standardize these, everything breaks:

1) Canonical Campaign / Project

This lives in **QuickSuite**, not Asana or Workfront.

Fields (subset):

- `campaign_id` (primary key everywhere)
- type (email, launch, event, awards, partner)
- tier (1 / 2 / 3)
- goals (structured + freeform)
- KPIs
- timeline
- owner role(s)
- execution_system = `asana | workfront | both`

Asana/Workfront projects are **execution mirrors**, not the source of truth.

---

2) Canonical Strategy Artifact

What your agent generates and revises.

- Versioned
- Human editable
- References standards/playbooks/templates
- Becomes the **brief** that execution teams trust

This is what:

- SMB marketer edits
- PMM reviews
- Event marketer adjusts
- Director reads at a glance

---

3) Canonical Knowledge Layer

From S3 / SharePoint / Quip:

- playbooks
- templates
- brand guidance
- historical learnings

This is **retrieval + grounding**, not execution.

---

4) Canonical Performance Model

Service-to-service metrics:

- spend
- delivery
- engagement
- ROI
- pipeline impact

Always joined via `campaign_id`.

---

3. How each use case maps cleanly (no special cases)

I'll go through them fast but precisely.

Use Case 1: SMB marketer → Email campaign → Workfront

What's really happening

- Agent is in **Strategy + Intake mode**
- User authority: *can create campaigns*
- Execution system: **Workfront**

Flow

1. User enters intent
2. Agent:
   - pulls email playbook
   - asks refinement questions
   - generates **canonical strategy**
3. User edits
4. “Submit”
5. QuickSuite:
   - creates Canonical Campaign
   - creates Workfront project (mirror)
6. Sync loop starts

✓ Director doesn’t need Workfront
✓ Production team trusts Workfront
✓ Strategy stays visible in QuickSuite

Use Case 2: PMM → Launchpad → Tier 2 launch

Key insight: **Launchpad is just another upstream signal**, not a system of record.

Flow

- Launchpad → QuickSuite (read-only ingest)
- Agent fills gaps
- Strategy proposed
- Write → Workfront
- Agent outputs *PMM-specific next steps*

The important design choice:

> The agent is allowed to **block creation** until required fields are present.

This is where quality jumps vs legacy workflows.

Use Case 3: Event marketer → Cannes → Asana

Same system, different execution target.

Why Asana instead of Workfront?
Because event ops ≠ production ops.

Flow

- Agent uses **Event Playbook**
- Tier 1 changes:
  - more dependencies
  - longer timelines
  - more stakeholders
- Agent creates Asana project
- Post-event:

  - performance ingestion
  - follow-up workflows triggered automatically

No new architecture required — just a different playbook + execution adapter.

Use Case 4: ME&I production marketer → Workfront-centric

This is the **mirror persona** of #1 and #2.

Here the agent runs in **Execution Assistant mode**, not Strategy Lead mode.

Agent capabilities:

- capacity planning
- template selection
- dependency awareness
- approval routing
- proactive risk alerts

Critical rule:

> The agent **never invents scope** here — it executes against the approved canonical strategy.

This is how you avoid production teams hating it.

Use Case 5: Partner Awards → Asana + content heavy

This is:

- long-running
- structured
- knowledge-heavy
- repetitive year over year

Perfect agent use case.

Key value:

- institutional memory
- automation of reminders
- post-event learning capture

Same campaign object, different lifecycle length.

Use Case 6: Director → Portfolio + performance

This is where everything pays off.

Director:

- never touches Asana
- never touches Workfront
- never uploads docs

- never manages tasks

They see:

- campaign rollups
- risk signals
- KPI trends
- compliance metrics (brief attached, AI usage, etc.)

Agent answers are **read-only + advisory** unless explicitly authorized.

---

4. The hard constraints you must enforce

This is where most systems fail.

Constraint 1: One-way authority by role

- Marketers: strategy + approve
- Production: execute + update delivery
- Directors: observe + intervene strategically

The agent must **refuse actions** that violate role authority.

Constraint 2: No free-text → execution writes

All writes to Asana/Workfront must come from:

- structured objects
- validated fields
- known templates

Otherwise you'll corrupt downstream systems.

Constraint 3: Execution tools are not conversational

The **conversation lives in QuickSuite**.
Asana/Workfront are:

- mirrors
- audit trails
- delivery systems

Never let conversation logic leak into them.

Constraint 4: Knowledge ≠ truth

Templates, playbooks, guidance:

- inform strategy
- do not auto-execute

The agent must *cite* them, not blindly apply them.

---

5. Why this actually works (and scales)

Because:

- You separated **thinking** from **doing**
- You separated **people auth** from **system auth**
- You created a **canonical campaign spine**
- You treated agents as **orchestrators**, not bots

This is *exactly* how mature internal systems at Amazon / Adobe / Salesforce are evolving — just not unified yet.

Final blunt take

You are not designing:

> “an agent that creates projects”

You are designing:

> **a marketing operating system where agents enforce quality, memory, and consistency across tools humans already trust**

If you want, next we can:

- Draw the **exact state machine** for a campaign
- Define the **agent permission matrix**
- Or design the **schema + adapters** for Asana vs Workfront writes

But structurally: what you described is sound — and notably **hard to fake** without real integration depth.

Alrite, let’s deep think and come up with high level designs, and also let’s not limit ourselves to Quick Suite only, we can build custom solutions, anothet aspect is there are some internal tools we have to integrate too like measurement systems. Deep think, analyze like an architect, a principal engineer

Alright. I’m going to treat this like a Principal Engineer design review: **clear system boundaries, canonical data model, integration patterns, security/permissions, and the “agent” as a product surface—NOT the architecture.** QuickSuite (or any vendor) becomes an implementation detail.

---

1) The north-star: a Marketing OS, not an “agent”

Your use cases all collapse into one system:

Marketing OS = Canonical Campaign Spine + Connectors + Knowledge + Workflow + Analytics + Agent surfaces

- **Canonical Campaign Spine**: one truth of campaign intent, tiering, goals, timeline, owners, approvals, and “what should happen.”

- **Execution systems** (Workfront/Asana/internal tools): where tasks/assets/approvals actually live.
- **Measurement systems**: where performance truth lives.
- **Knowledge systems** (S3/SharePoint/Quip): playbooks, templates, brand guidance, historical learnings.
- **Agent**: a UI + orchestration layer that reads/writes via policy-controlled APIs.

If you skip the canonical spine and try to "LLM glue" everything together, it becomes ungovernable quickly.

---

2) High-level architecture (platform view)

Core services

1. **Campaign Service (System of Record)**
   - Canonical campaign object, strategy artifact versions, approvals, state machine
   - Owns `campaign_id` and lifecycle
2. **Workflow Orchestrator**
   - Deterministic state machine for Tier 1/2/3 launch/event/awards/etc.
   - Steps, gates, SLAs, dependencies, auto-remediation suggestions
   - Think: "process engine," not "task engine"
3. **Integration Hub (Connectors + Webhooks + Sync)**
   - Adapters for Workfront, Asana, Launchpad, Quip, SharePoint, S3, internal tools
   - Normalizes inbound/outbound
   - Handles rate limits, retries, idempotency, mapping, conflict resolution
4. **Knowledge & Templates Platform**
   - Ingestion + parsing + metadata enrichment
   - Retrieval index (keyword + semantic)
   - Policy enforcement: what can be used/seen by whom
5. **Measurement Platform**
   - Canonical metrics model, ingestion pipelines, attribution mapping
   - Joins performance ↔ campaign spine (by `campaign_id` , `asin` , `launch_id` , etc.)
6. **Agent Runtime**
   - Tool-calling, retrieval, policy checks, action execution, audit logs
   - "Chat" is just one client; others include dashboards, forms, Slack, email
7. **Portfolio & Executive Insights**
   - Risk scoring, schedule confidence, budget health, KPI anomaly detection
   - Portfolio comparisons over time ("event ROI last 3 quarters")

---

3) Canonical data model (the spine)

You need a **minimal but strict** canonical model. Here's the "principal engineer" version:

Campaign (core)

- `campaign_id` (global primary key)
- `type` (email | launch | event | awards | partner | …)
- `tier` (1/2/3)

- `objective` (structured + freeform)
- `audience` (segments, constraints)
- `entities` (ASINs, products, geos, channels)
- `kpis` (target, measurement method)
- `timeline` (milestones, launch date, critical path)
- `owners` (requester, DRI, approvers)
- `execution_targets` = {workfront_project_id?, asana_project_id?, internal_project_id?}
- `state` (draft → approved → in-flight → shipped → postmortem)

Strategy Artifact (versioned)

- `strategy_version_id`
- `inputs` (filled questionnaire)
- `assumptions`
- `recommended_plan` (channels, cadence, creatives, budgets)
- `risks + mitigations`
- `required_assets`
- `citations` (KB references used)

Work Items (normalized)

A normalized layer for tasks/approvals/assets across systems:

- `work_item_id`
- `campaign_id`
- `kind` (task | approval | asset | dependency)
- `owner_team` , `assignee`
- `status` , `dates`
- `external_ref` (workfront/asana ID)
- `dependencies` edges (graph)

Measurement Mapping (critical)

This is where most orgs fail.

- `campaign_id`
- `measurement_keys` : {asin, offer_id, utm, pixel_id, internal_tracking_id, etc.}
- `attribution_rules` per channel
- `source_of_truth` per metric

If you don't do this mapping layer, "director wants performance" becomes hand-wavy forever.

---

4) Integration strategy: stop thinking "OAuth vs service-to-service" only

Think **integration archetypes**:

A) Execution connectors (Workfront/Asana)

- Pattern: **Mirror + Writeback**
- "Create project" → creates external project + stores mapping
- Bi-directional sync for status + dates + key fields

- Writes must be **strictly structured** and idempotent

B) Knowledge connectors (S3/SharePoint/Quip)

- Pattern: **Ingest + Index + Govern**
- You don't "sync" like tasks; you crawl, parse, tag, enforce access

C) Measurement connectors (internal systems, ads platforms, analytics)

- Pattern: **Ingest + Normalize + Join**
- High volume, event/batch pipelines
- Service-to-service always
- Late arriving data, backfills, and versioned metrics

D) Internal tools

- Pattern depends on the tool:
  - If it's "source of truth": you mirror it
  - If it's "auxiliary": you reference it
  - If it triggers workflows: event integration + orchestration

Architectural rule: integrations should speak to the platform via **one internal contract**, not per-agent code.

---

5) Workflow engine design (deterministic, not LLM)

Your best reliability comes from:

"LLM proposes; workflow engine enforces"

- LLM creates draft strategy and suggests next steps
- Workflow engine ensures:

Also consider that we want to give the autonomy to users to create more agents in future

Perfect — "users can create more agents later" changes the architecture a lot. If you don't design for this now, you'll end up with a brittle set of one-off bots that can't be governed.

Here's the Principal Engineer way to do it: **agents are plugins on a governed platform**, not bespoke code paths.

---

1) Product framing: "Agent Studio" on top of a common platform

You're building two layers:

Layer A — The Marketing OS Platform (shared, non-negotiable)

- Canonical Campaign Spine (SoR)
- Workflow engine (state machine + gates)

- Integration Hub (connectors, sync, idempotency)
- Knowledge/Template platform (ingest/index/ACL)
- Measurement platform (metrics, joins, attribution mapping)
- Policy & audit (RBAC, approvals, logging)

Layer B — Agent Experience (extensible)

- Prebuilt agents (XYZ, Launch Agent, Production Agent, Exec Agent)
- A **user-facing "Agent Studio"** to let teams create more agents safely

The key is: Agent Studio should let users assemble agents **from approved building blocks**.

---

2) Agent autonomy without chaos: what is an "agent" really?

Define an agent as:

> **(Prompt + Tools + Policies + Memory + Workflows + UI)**

Not "a model that can do anything."

So in your system, an agent is **a configuration artifact** backed by a runtime.

Agent definition (high-level)

- **Persona**: role + objective (e.g., Event Marketer Agent)
- **Capabilities**: allowed tools/actions (read/write)
- **Data access scope**: which campaigns/teams/regions
- **Workflow bindings**: which playbooks/state machines it can invoke
- **KB bindings**: which corpora it can cite
- **Output contracts**: schemas it must produce (brief, plan, risk report)
- **Escalation policy**: when to request approval/human intervention
- **Audit & logging**: mandatory

This becomes a "governed app," not a chat prompt.

---

3) The platform primitives users can compose

To enable user-created agents, you must provide primitives that are safe and composable:

A) Tools (strict interfaces)

Tools are the only way an agent can act.

Examples:

- `create_campaign()`
- `update_campaign_strategy()`
- `create_workfront_project()`
- `sync_asana_project_status()`
- `query_performance(campaign_id, window, metrics)`

- `retrieve_kb(query, scope)`
- `create_task_in_asana(schema)`
- `post_decision_to_workfront(schema)`

Rule: Tools are schema-first. No free-text writes into external systems.

B) Workflows (state machines)

Reusable playbooks:

- "Tier 1 Event"
- "Tier 2 Launch"
- "Email ASIN Penetration"
- "Partner Awards"

User-created agents can *invoke* workflows, not redefine business-critical ones (unless you permit workflow authoring later).

C) Schemas (contracts)

Everything meaningful is structured:

- Brief schema
- Campaign plan schema
- Risk schema
- Creative spec schema
- Post-mortem schema

LLM outputs must validate against schema before any write.

D) Knowledge Packs (curated corpora)

Instead of "search everything," bind agents to KB packs:

- Brand guidance pack
- Event playbook pack
- Launch standards pack
- Legal/compliance pack

This is how you prevent random hallucinated policy.

E) Policy Packs (permissions and approval rules)

Example:

- "Exec agent is read-only"
- "Production agent can update due dates but cannot change scope"
- "Agency-facing agent cannot access internal-only KB packs"

---

4) Agent Studio: what users can customize (and what they can't)

Let users customize

- Agent name, description, persona
- Which KB packs it can use
- Which workflows it can invoke
- Which input forms it asks (questionnaire)

- Which output templates it produces
- Notification rules (Slack/email)
- Read-only vs write-with-approval vs write-direct (within their role)

Do NOT let users customize freely (at first)

- Raw API credentials
- Direct “execute arbitrary HTTP”
- Unbounded retrieval over all knowledge
- Direct write access to Workfront/Asana fields without schemas
- Ability to bypass approvals

Think of this like AWS IAM: you can’t give arbitrary power without guardrails.

---

5) Governance model: avoid “agent sprawl”

You’ll need an internal governance loop:

A) Agent lifecycle

- Draft → Test (sandbox) → Published → Deprecated

B) Sandboxed testing

- Run agent against synthetic campaigns
- Replay historical scenarios
- Validate:
  - schema outputs
  - tool usage
  - policy compliance
  - hallucination rate / citation behavior

C) Observability

Per-agent dashboards:

- actions taken
- success/failure rate
- time saved
- escalations
- user edits after generation (quality proxy)
- “policy violations blocked”

D) Approval and ownership

Every agent has:

- owner
- approver (optional)
- security scope
- audit trail

---

6) Safety & correctness: the “agent action firewall”

This is the architectural centerpiece.

All actions flow through:

Policy Decision Point (PDP)

Evaluates:

- who is the user
- what agent
- what campaign state
- what tool action
- what data scope
- what approval rules required

Policy Enforcement Point (PEP)

Actually enforces:

- allow
- block
- require approval
- require confirmation
- redact data

This is how you safely allow user-created agents without losing control.

---

7) Memory design for user-created agents

Agents will want memory. You need two kinds:

A) Declarative memory (safe)

- saved preferences
- saved templates
- saved playbooks references
- “house style”

B) Factual operational memory (scoped)

- campaign decisions
- approvals
- outcomes
- postmortems

Rule: operational memory must be tied to canonical objects (campaign_id) and be auditable. Don’t let agents store opaque “notes” that later drive actions without traceability.

---

8) Recommended maturity path (pragmatic)

Phase 1: Platform + curated agents

- Build the platform primitives
- Ship 3–4 high value agents (Launch, Event, Production, Exec)
- No user creation yet, but everything is already “config-driven”

Phase 2: Agent Studio (restricted)

- Users can create read-only agents
- Or write-with-approval only
- Limited KB packs + workflows

Phase 3: Advanced Agent Studio

- Workflow authoring (guarded)
- Custom connectors (via secure integration templates)
- Domain-specific tool creation (still schema-first)

---

9) The key design choice you must make now

Do you want user-created agents to be:

1. **Prompt-first** (easy, dangerous, ungovernable), or
2. **Contract-first** (slower initially, scalable, safe)

For your org + Workfront/Asana + internal measurement + director visibility, you want **contract-first**.

Concrete deliverable I can produce next

If you want to move forward, I can write a high-level design doc outline with:

- system components
- data model (canonical campaign spine)
- agent runtime architecture
- policy model (PDP/PEP)
- tool schema contracts
- Agent Studio MVP spec (what's in/out)
- rollout plan + risks

Say the word and I'll draft it in a structured format you can paste into Quip.

I didn't understand contract first vs prompt first, can you explain and we can move to high level

Good catch — this distinction is subtle, but it's **the single most important architectural decision** you'll make for an agent platform like this. I'll explain it *cleanly*, *without jargon*, then we'll zoom back up to high-level design.

Prompt-first vs Contract-first (plain English)

Prompt-first = "Tell the AI what to do"

Contract-first = "Define what *can* be done, then let AI help within that"

Think of this like **human processes vs APIs**.

---

1) Prompt-first (what most people build)

How it works

You write prompts like:

> “You are a marketing agent. Create a campaign in Workfront with tasks, timelines, and owners.”

The LLM:

- reasons
- decides what fields to fill
- decides what tasks to create
- decides what to write to systems

Why it feels attractive

- Fast to prototype
- Very flexible
- “Feels intelligent”
- Easy demos

Why it breaks at scale

From an architect perspective, this is **unbounded authority**.

Problems:

- ❌ No guarantee required fields exist
- ❌ No stable structure across runs
- ❌ Silent schema drift (“today it created 12 tasks, tomorrow 9”)
- ❌ Hard to audit (“why did it do this?”)
- ❌ Dangerous writes into Workfront/Asana
- ❌ Impossible to give users agent-creation safely
- ❌ Directors will eventually distrust it

Prompt-first is fine for drafting text.

It is terrible for operating systems.

---

2) Contract-first (how real systems scale)

Core idea

You **define contracts first**, then the LLM operates *inside* those constraints.

A **contract** is:

- a schema
- a state machine
- a permission boundary
- an approval rule

The agent:

- proposes
- fills
- reasons

  ...but cannot violate the contract.

---

3) Concrete example (your world)

Example: “Create a campaign”

Prompt-first version

> “Create a Tier 1 event campaign for Cannes.”

The agent:

- guesses tasks
- guesses owners
- guesses timelines
- writes directly to Asana

Sometimes good. Sometimes wrong. Always risky.

Contract-first version

You define **beforehand**:

Campaign Contract

json Copy code

```json
{
  "campaign_type": "event",
  "tier": "1",
  "required_fields": [
    "event_name",
    "dates",
    "goals",
    "budget_range",
    "owner",
    "success_metrics"
  ],
  "workflow": "tier1_event_playbook",
  "execution_system": "asana",
  "approvals": ["marketing_lead"]
}
```

Task Template Contract

json Copy code

```json
{
  "tasks": [
    {"name": "Booth creative", "owner_team": "Creative"},
    {"name": "Speaker submissions", "owner_team": "Events"},
    {"name": "Pre-event campaign", "owner_team": "Growth"}
  ],
  "dependencies": true,
  "sla_rules": true
}
```

What the agent does now

- Asks **only** for missing required fields
- Proposes a strategy **that fits the contract**
- Cannot create the project until validation passes
- Cannot write fields that don't exist
- Cannot skip approvals
- Cannot invent workflow steps

The AI is still "smart" — but it's safe.

---

4) Analogy that usually clicks

Prompt-first is like:

> Letting a junior employee directly edit production databases because "they seem smart."

Contract-first is like:

> Giving them a form, a checklist, and approval gates — and letting them *fill it intelligently*.

---

5) Why this matters for *user-created agents*

This is the big one.

Prompt-first + user agents = 🔥

If users can write prompts like:

> "Create an agent that manages campaigns end-to-end"

You've just given them:

- write access
- no guardrails
- no audit
- no enforcement

You will shut this down after one incident.

Contract-first + user agents = scalable

Users create agents by:

- selecting allowed **contracts**

- selecting allowed **tools**
- selecting allowed **KB packs**
- selecting allowed **workflows**

They are **configuring behavior**, not inventing power.

This is how:

- AWS IAM
- Salesforce flows
- ServiceNow
- Stripe workflows

actually work.

---

6) Where LLMs still shine (even contract-first)

This is important: contract-first does **not** reduce intelligence.

LLMs still:

- ask the right questions
- reason over tradeoffs
- synthesize strategies
- summarize status
- explain risks
- draft content
- propose mitigations

They just don't get to:

- invent schema
- bypass approvals
- write arbitrary data
- break invariants

---

7) The exact line to remember

> **Prompt-first decides *what* to do.**
> **Contract-first decides *what is allowed*, then AI helps *how* to do it well.**

---

8) Why your use cases *require* contract-first

Because you have:

- Workfront + Asana
- internal measurement systems
- directors reading but not acting
- production teams depending on accuracy
- future user-created agents

Any one of those alone pushes you toward contract-first. All together? It's mandatory.

9) Now we're ready to go high-level

If you're aligned, next we should:

1. Define the **core contracts** (Campaign, Strategy, Work Item, Measurement)
2. Define the **agent runtime** that enforces them
3. Define **Agent Studio** as a contract composer, not a prompt editor
4. Map this to build vs buy (QuickSuite vs custom)

Say **"yes, move to high-level design"** and I'll lay out the full architecture cleanly end-to-end.

Let's do it

Cool — here's the **high-level design** as if we're writing the "Marketing OS + Agent Platform" architecture doc. I'll keep it crisp, but architect-grade.

---

0) Goals and non-goals

Goals

- Single experience to **plan → execute → track → learn** across marketing
- Agents that:
  - ask the right questions
  - generate strategies
  - create/monitor execution projects (Workfront/Asana/internal)
  - summarize status + risks + performance
  - write back decisions safely
- Directors can get portfolio/performance **without accounts** in execution tools
- Platform supports **user-created agents** later, safely
- Integrate internal **measurement systems** + internal tools

Non-goals (at first)

- Replacing Workfront/Asana/Launchpad/SharePoint/Quip
- Letting any agent do arbitrary writes without contracts/approval

---

1) System overview (major components)

A) Canonical Campaign Spine (System of Record)

Owns the truth of:

- campaign intent, tier, dates, goals, KPIs
- strategy artifact versions + approvals
- mapping to execution projects (Workfront/Asana/others)
- campaign lifecycle state machine

Output: a single `campaign_id` that everything joins on.

B) Workflow Engine (Playbooks)

Deterministic state machines for:

- Tier 1 events
- Tier 2 launches
- Email/ASIN penetration
- Partner Awards
- Production execution

Responsibilities:

- required fields/gates
- step ordering + dependencies
- SLA rules
- approval rules
- auto-next-step suggestions (but enforcement is deterministic)

C) Integration Hub (Connectors + Sync)

Adapters for:

- Workfront, Asana, Launchpad, Quip, SharePoint, S3
- internal tools (measurement systems, DAM, capacity tools, etc.)

Responsibilities:

- auth (OAuth/app-permission/service-to-service)
- mapping external IDs ⇔ `campaign_id`
- bidirectional sync with conflict resolution
- idempotency, retries, rate limits
- event ingestion (webhooks) where possible

D) Knowledge Platform (Templates + KB)

Sources:

- S3 templates
- SharePoint documents
- Quip docs

Responsibilities:

- ingest + parse + chunk + embed
- metadata tagging (team, playbook type, version, region)
- access control (RBAC/ACL mirroring as needed)
- retrieval APIs returning **citations**

E) Measurement Platform (Performance + Attribution)

Sources:

- internal measurement systems
- ads platform feeds
- clickstream / conversions

Responsibilities:

- canonical metrics schema
- mapping layer: measurement keys → `campaign_id`
- backfills, late data, versioned metrics
- aggregation service for dashboards + agent queries
- anomaly detection signals (optional)

F) Agent Runtime (Tool-using + Governed)

This is not “a chatbot.” It’s a **runtime** with:

- tool registry (what actions exist)
- policy engine (what’s allowed)
- schema validator (contracts)
- conversation + task memory (auditable)
- approval workflow integration
- action/audit logs

G) Experiences (Clients)

- XYZ dashboard (role-based)
- chat agent UI
- campaign creation wizard (optional structured UI)
- director portfolio view
- production cockpit
- notifications (Slack/email)

---

2) The “contract-first” core: what contracts exist?

These are the minimum contracts you define up front.

Contract 1: Campaign

Fields + invariants:

- type, tier, owner, timeline, KPIs
- must have required fields by tier before “Create Project”
- has lifecycle states: draft → approved → in-flight → shipped → postmortem

Contract 2: Strategy Artifact

- structured strategy sections
- required citations to KB playbooks
- versioned and approvable
- links to required assets/work items

Contract 3: Work Item

Normalized representation of tasks/assets/approvals

- maps to Workfront/Asana/other
- dependency graph

- SLA semantics

Contract 4: Measurement Mapping

- defines how metrics attach to campaigns
- validation rules (“campaign has no measurement key → performance cannot be reported”)

Contract 5: Action contracts (tools)

Every write tool accepts/returns strict schemas.

Example: `create_workfront_project(payload)` must have:

- campaign_id
- required fields
- task template references (not free text tasks)

---

3) The campaign lifecycle state machine (high-level)

Draft

- capture intent + basic fields
- agent asks missing questions

Strategy Proposed

- agent generates strategy with citations
- user edits

Approved

- approvals recorded
- required fields validated (tier-specific)

Execution Created

- external project created (Workfront/Asana/internal)
- mapping stored: `campaign_id ↔ external_project_id`
- baseline schedule + dependencies recorded

In-flight

- continuous sync of statuses, dates, blockers
- performance metrics populate as available

Shipped

- deliverables done, launch happened
- early performance tracked

Postmortem + Learnings Captured

- agent compiles results + insights
- writes learnings back into KB as “case study” (optionally gated)

---

4) Integration patterns (how data actually flows)

A) Execution systems (Workfront/Asana)

Pattern: **Mirror + Writeback**

- Creation:
  - Campaign Spine triggers `create_project` in target system
- Sync:
  - inbound: status, dates, blockers, asset links
  - outbound: decisions, timeline updates (policy gated)
- Conflict:
  - define precedence per field:
    - “status” authoritative in execution system
    - “objectives/KPIs” authoritative in campaign spine
    - “budget actuals” authoritative in measurement system

B) Knowledge systems (S3/SharePoint/Quip)

Pattern: **Ingest + Index + Govern**

- scheduled crawls + incremental updates
- normalize metadata
- retrieval returns chunks + citations

C) Measurement systems (internal)

Pattern: **Ingest + Normalize + Join**

- streaming or batch ingestion
- canonical metrics tables
- join via mapping keys

---

5) The policy engine (the guardrail that enables user-created agents)

Core concept

Every tool call goes through:

- PDP (policy decision point): allow/deny/needs-approval
- PEP (policy enforcement point): executes or blocks

Policy factors

- user role (marketer, production, director)
- agent identity (which agent is acting)
- campaign state (draft vs in-flight)
- action risk level (read vs write)
- data scope (BU, region, org)
- approvals present or not

Modes

- Read-only
- Write with approval
- Write direct (rare, role-scoped)

This is what lets you safely let teams build their own agents later.

6) Agent Runtime: how an agent behaves (deterministic loop)

For any use case, the runtime does:

1. **Intake**
    - gather structured fields (form-like via chat)
2. **Retrieve**
    - pull relevant KB + historical learnings
3. **Plan**
    - propose strategy (schema + citations)
4. **Validate**
    - does it satisfy required fields? tier rules?
5. **Approve**
    - user edits + explicit “submit”
    - approvals collected if required
6. **Execute**
    - call tool(s) to create/update Workfront/Asana/internal
7. **Observe**
    - subscribe to events / poll for status
8. **Explain**
    - answer questions with data + citations + confidence
9. **Learn**
    - record outcomes, create case study drafts

---

7) Agent Studio (user-created agents): the safe design

Users create agents by configuring:

- Persona + objective
- Allowed KB packs
- Allowed workflows/playbooks
- Allowed tools (read-only, or write-with-approval)
- Scope (teams, campaigns, regions)
- Output templates (brief, plan, exec summary)

They do NOT create raw credentials or arbitrary HTTP calls.
They compose approved capabilities.

---

8) Build vs Buy (QuickSuite included but not required)

You can implement this platform in two ways:

Option A: QuickSuite as UX + partial platform

- If QuickSuite has strong connectors, agent runtime, and policy controls
- You still likely need:
    - your canonical campaign spine
    - measurement join layer

  - knowledge ingestion governance

Option B: Custom platform (recommended if you need deep internal integrations)

- You own:
  - data model
  - workflows
  - connectors
  - agent runtime + policy engine
- You can still embed vendor tools for BI/search if helpful

My principal-engineer bias: **build the spine + policy + integration hub**, and treat UI/LLM vendor components as replaceable.

---

9) MVP slice (what to build first)

To avoid boiling the ocean:

MVP 1: Tier 2 Launch + Workfront + Measurement read

- Campaign Spine
- 1 playbook workflow
- Workfront create + sync (basic fields + status)
- KB ingest from S3 + one doc source
- Measurement join for 3–5 KPIs
- One agent + one dashboard (marketer + director view)

MVP 2: Add Asana + Event workflow

MVP 3: Production cockpit + writeback decisions

MVP 4: Agent Studio (read-only first)

---

10) Biggest risks (callouts)

- **Identity & permissions** across sources (directors w/o tool accounts)
- **Mapping performance to campaigns** (measurement keys discipline)
- **Writeback conflicts** (need clear authoritative fields)
- **Agent sprawl** without contracts/policy (hence this architecture)
- **Data freshness** expectations (near real-time vs daily)

Next step (what I'll produce in the next response)

Pick one of these and I'll write it as a real design artifact:

1. **System diagram + sequence diagrams** for 2 flows: "Create campaign" and "Director asks status"
2. **Canonical schema + field-level authority matrix** (who owns each field)
3. **Integration hub contract + tool registry** (the internal APIs agents use)
4. **Agent Studio MVP spec** (permissions, UI, publishing model)

If you want my recommendation: start with **(2) + (3)** — that locks the foundation.

Can you do a deep dive on the feasibility with 1) Quick Suite + partial platform 2) any other 3P solution 3) Custom platform and 4) if salesforce has any such offerings?

Here's the feasibility deep dive across the 4 paths, using your actual requirements (Workfront + Asana + internal measurement + KB from S3/SharePoint/Quip + "directors don't have upstream accounts" + future "users create more agents").

---

1) Amazon Quick Suite + partial custom platform

What Quick Suite is strong at

- **BI + dashboards + NLQ + agents + automation** in one AWS-native surface. AWS Documentat...
- It has an **integration model that explicitly covers both**:
  - **Actions** (call external APIs and "perform actions" via action connectors; shareable; usable in automated workflows) AWS Documentat...
  - **Knowledge bases** built from "data access integrations," usable in spaces or agents AWS Documentat...
- It supports multiple auth patterns out of the box: user OAuth, service-to-service OAuth/API key, managed OAuth2, AWS creds, etc. AWS Documentat...

Feasibility for your use cases

High for:

- Director dashboard/portfolio view (QuickSight heritage + expanded suite) AWS Documentat...
- "Ask the agent" questions across synced campaign objects and metrics
- Building "agent surfaces" quickly (especially if your org is already AWS-first)

Medium for:

- Robust *bi-directional* sync into Workfront/Asana with strict contracts (it can call APIs, but you still need a strong **canonical campaign spine + policy** to avoid bad writes)

Key gap you still need to build

Quick Suite gives you the *surface + automation + connectors*, but you still need a **custom "canonical campaign spine + measurement join layer"** if you want:

- stable `campaign_id` everywhere
- field-level "source-of-truth" rules
- safe writeback (approval gates, idempotency, conflict resolution)

Best-fit architecture

- Quick Suite = **UX + agent runtime + dashboards**

- Your custom platform = **SoR + policy engine + integration hub + measurement mapping**
- Quick Suite actions call **your internal APIs**, not Workfront/Asana directly (so you enforce contracts centrally).

My take: very feasible and often the fastest enterprise path **if** you treat Quick Suite as the *presentation/orchestration layer*, not the system of record.

---

2) Other 3P solution (non-Salesforce): iPaaS / workflow platforms + agent layer

This category is: **Workato, Tray.ai, ServiceNow Integration Hub, Microsoft Copilot Studio, Atlassian Rovo**, etc.

2A) iPaaS (Workato / Tray.ai)

- Workato positions itself as a large connector library for automations. Workato
- Tray explicitly markets "integrate apps + automate + build agents under IT control." Tray AI

Feasibility

- **High** for: connecting Workfront/Asana/internal tools quickly, event-driven workflows, retries, rate limits, basic governance
- **Medium** for: deep agent behaviors tied to canonical contracts (you can do it, but you'll often end up building more custom control-plane than you expect)

Where this shines

- If your biggest pain is **integration speed** and "workflow glue" across many SaaS tools.

Where it struggles

- Your "Marketing OS" needs a **canonical spine** + strong "director view" + measurement joins. iPaaS can move data, but it's not usually the best SoR.

2B) ServiceNow (Integration Hub + Now Assist / AI Agents)

ServiceNow explicitly has "Now Assist AI agents" + "Integration Hub" concepts in docs. servicenow.com +1

Feasibility

- **High** if your org already runs big chunks of workflow in ServiceNow (IT/HR/ops patterns map well)
- **Medium** for marketing-specific campaign + production workflows unless you're willing to "ServiceNow-ize" a lot of your marketing process

Important caution

- Multi-agent systems introduce real governance risks (prompt injection / privilege escalation patterns are a known concern class). TechRadar
  This doesn't make it unusable, it just means you need serious PDP/PEP controls (which you already intend).

2C) Microsoft Copilot Studio

Copilot Studio is positioned as an "end-to-end conversational AI platform" to build agents and publish to Microsoft 365 Copilot. Microsoft
It's also adding curated "Action Groups" for connectors like Outlook/SharePoint. Microsoft

Feasibility

- **Very high** for your KB story (SharePoint is first-class), and for orgs standardized on M365
- **Medium** for Workfront/Asana + internal measurement unless connector coverage + custom actions are strong enough in your environment

2D) Atlassian Rovo (Jira/Confluence ecosystems)

Rovo is strong if your knowledge and work live in Confluence/Jira, with "search + agents" integrated. Atlassian +1

Feasibility

- **Low-to-medium** for your stated world (Workfront + Asana + SharePoint/Quip + internal measurement). It's not the natural hub unless Atlassian is already your primary.

---

3) Custom platform (build your own Marketing OS)

Feasibility

High, but only if you scope it correctly.

What you must build (minimum)

- Canonical Campaign Spine (SoR)
- Policy engine (PDP/PEP) + audit
- Integration hub (Workfront/Asana + internal tools)
- Knowledge ingestion (S3/SharePoint/Quip) + retrieval with citations
- Measurement mapping + metrics store + APIs

What you gain

- Best alignment with "contract-first"
- Clean "director can see everything without upstream accounts"
- You can support "user-created agents" safely as *configurations over approved tools/workflows*

What will bite you

- Connectors and edge cases (rate limits, webhook weirdness, access models, admin consent, etc.)
- Time-to-first-value unless you pick a thin MVP slice

My take: if internal measurement + internal tools integration is heavy and strategic, custom becomes the long-term winner—*but* pair it with a strong surface (Quick Suite / Salesforce / Microsoft) to avoid building a whole BI + agent UX from scratch.

---

4) Salesforce: does it have offerings that match?

Yes—Salesforce is explicitly pushing an "agent + data + actions" platform direction.

Agentforce (agent framework)

Salesforce describes Agentforce as agents with **data + reasoning + actions**, able to connect to data sources and leverage workflows/APIs. Salesforce
They also emphasize RAG + unstructured + structured enterprise data grounding. Salesforce +1

Next-Gen Marketing Cloud (agentic marketing)

Salesforce positions Next-Gen Marketing Cloud as an "agentic marketing solution" built on their platform. Salesforce +1

Integration tooling (MuleSoft / Composer)

They also have MuleSoft Composer for connecting apps and automating across systems. Salesforce

Feasibility vs your requirements

High if:

- your customer/audience data already lives in Salesforce (or you want it to)
- you want "marketing execution" + personalization + orchestration in that ecosystem

Medium if:

- your center of gravity is AWS + Workfront/Asana + internal measurement systems
- you primarily need a Marketing OS and not a customer engagement platform

Key question
Salesforce will be strongest when the "output" is marketing journeys, personalization, CRM actions, etc.
Your use cases are **marketing operations + production + project execution + portfolio governance**. Salesforce can do it, but you'd be bending it toward **Marketing Ops OS** rather than its native "customer relationship + activation" sweet spot.

Practical recommendation pattern (architect view)

If you're AWS-first and want fastest path:

Quick Suite + partial custom platform

- Quick Suite for UX/agents/dashboards
- Custom canonical spine + policy + measurement mapping
- Quick Suite actions call your internal APIs (your "contract firewall") AWS Documentat... +1

If you're M365-first and KB is mostly SharePoint:

Copilot Studio + custom spine

- Copilot Studio for agent UX + SharePoint grounding Microsoft +1
- Custom spine for campaign + measurement join + governance

If you're Salesforce-first and care about customer activation end-to-end:

Agentforce + Next-Gen Marketing Cloud (+ MuleSoft)

- Strong agent/data/action story Salesforce +2
- Still likely need a canonical campaign spine if Workfront/Asana remain core execution

If you need "connect everything fast" and can live with a thinner SoR:

iPaaS + a lightweight canonical layer

- Workato/Tray style orchestration Workato +1
- Keep the canonical campaign model minimal initially

The next thing I'd do (to make this decision real)

Pick **one representative workflow** (e.g., "Tier 2 launch → Workfront → director dashboard → measurement insights"), and do a feasibility spike that answers:

1. Can the platform do **role-scoped actions** cleanly (approval gates)?
2. Can it support **canonical IDs + field authority** (SoR not drifting)?
3. Can it ground answers in **KB with citations** (S3/SharePoint/Quip)?
4. Can it join **internal measurement** to campaigns at scale?

If you want, I'll write a 1–2 page "evaluation scorecard" you can use with stakeholders: criteria, weights, and a recommendation.

Recent AI agent platform context

Reuters
Salesforce deepens AI ties with OpenAI, Anthropic to power Agentforce platform
Oct 14, 2025

The Verge
Microsoft lets Copilot Studio use a computer on its own
Apr 16, 2025

TechRadar
Second-order prompt injection can turn AI into a malicious insider
Nov 21, 2025

Sources

Just so you know director usecase can be combined with other usecases too, because anyone would wanna see performance data. And we have our own sharepoint hosted on prem

Got it — two important implications:

1. **"Director view" (performance + health) isn't a separate persona**. It's a **capability** that every persona wants, just at different depth + permissions.

2. **SharePoint on-prem** changes the integration approach a lot (it's not "Graph + app permissions" in the same way as SharePoint Online).

Below is how I'd adjust the architecture and feasibility thinking.

---

1) Reframe "director use case" as a universal capability

Instead of "Director Agent" vs "Marketer Agent", model it like this:

Every user sees 3 layers (role-scoped)

- **Portfolio layer**: rollups, KPIs, risk, budget, ROI (what directors want)
- **Campaign layer**: brief, timeline, blockers, owners, dependencies
- **Execution layer**: task/asset details in Workfront/Asana + approvals

Same data, different slices.

How to enforce this cleanly

In your Canonical Campaign Spine, add:

- **Audience scopes**: BU / region / team / program / client
- **Visibility policies**: who can view which campaign + which metric categories
- **Metric redaction rules**: some metrics may be exec-only (e.g., budget)

Then your dashboard/agent just queries:

- "What am I allowed to see?" (policy)
- "What's the latest state and performance?" (facts)

So: build **one performance layer**, not "director performance".

---

2) SharePoint on-prem: what changes

Why it's different

- No Microsoft Graph "easy mode"
- Auth is typically **AD/LDAP** based (Kerberos/NTLM/claims/ADFS depending on setup)
- Access control often relies on **SharePoint ACLs + AD groups**
- Connectivity might require **network access to on-prem** (VPN, Direct Connect, private link, etc.)

Two viable integration patterns

Pattern A: Crawl + index inside your environment (recommended)
You run an on-prem (or VPC-connected) ingestion service that:

- crawls SharePoint on-prem content (sites/lists/docs)
- extracts text + metadata
- captures ACLs (optional but ideal)
- pushes normalized docs into your Knowledge Platform (S3/OpenSearch/Vector DB)

Pros:

- Works even with complex auth
- More reliable, scalable, controllable

- Lets you implement “contract-first” governance and citations

Cons:

- You own the crawler + incremental sync + ACL mapping

Pattern B: Federated search (faster, less control)
Your agent/search layer queries SharePoint Search remotely and retrieves results on demand.

Pros:

- Faster to stand up
- Less content duplication

Cons:

- Harder to guarantee citations + relevance + caching
- Latency and auth become runtime dependencies
- Much trickier when “directors don’t have SharePoint accounts” (no identity to delegate)

Given your earlier constraint (leaders/users may not have upstream accounts), **Pattern A is the practical choice**.

---

3) Permissions when users don’t have SharePoint accounts

You have to choose one of these governance models:

Model 1: QuickSuite/custom platform becomes the policy authority (common)

- You ingest from SharePoint using an integration identity
- You apply **your own RBAC** based on campaign/team metadata
- Directors can see what your internal policy allows

This is simplest and aligns with “no upstream accounts required.”

Model 2: Mirror SharePoint ACLs (more correct, more work)

- Ingest doc ACLs + AD group membership
- Map users to AD identities
- Enforce doc-level permissions in your platform

This only works well if your users exist in AD and you can map them. If many viewers truly have no identity in that realm, you’ll still end up with Model 1 for them.

My recommendation: **start with Model 1** + add “mirror ACLs” later for sensitive corpora.

---

4) How this affects the “platform options” discussion

If SharePoint is on-prem, any “3P SaaS” option must solve connectivity

- Either you deploy an **on-prem connector/gateway**
- Or you set up private connectivity into your network (often slow politically)

That tends to push you toward:

- **Custom knowledge ingestion service** (runs where SharePoint lives)
- Then feed whichever surface you choose (QuickSuite/Salesforce/M365/custom)

So even if you use QuickSuite (or Salesforce), you'll likely still build/operate an **on-prem content ingestion bridge**.

---

5) Immediate architecture tweak I'd bake in

Add two first-class services to the earlier design:

A) Knowledge Ingestion Bridge (supports on-prem)

- Pluggable crawlers: SharePoint on-prem, file shares, Quip, etc.
- Produces: `doc_id` , `chunks[]` , `metadata` , `citations` , optional `acl`

B) Performance Aggregation API (universal capability)

- One API powering dashboards + agent answers for *all roles*
- Enforces policy at query time (metric visibility + scope)
- Joins measurement ↔ campaign via `campaign_id` mapping

This makes "anyone wants performance" trivial: the experience changes, not the backend.

If you want the next step "architect style", I can produce a **reference architecture diagram** (components + trust boundaries) and a **data authority matrix** that explicitly answers:

- Which system owns each field (goal, status, budget actuals, asset readiness)?
- Who can view which metrics (role-based)?
- How on-prem SharePoint content becomes citeable KB for agents?

And I'll include the on-prem connectivity/security boundary (where the bridge runs, how it authenticates, and what gets replicated).

Let's do architecture

Below is a **reference architecture** for the "Marketing OS + Agent Platform" that covers:

- Workfront + Asana execution
- Internal measurement systems
- Knowledge/templates from **S3 + Quip + SharePoint on-prem**
- "Everyone wants performance" (not just directors)
- Future **user-created agents** with guardrails
- Strong governance (contract-first, approvals, audit)

---

1) Reference architecture at a glance

Trust boundaries + data planes

text Copy code

```
+-------------------------------------------------------------------------
|                               USER EXPERIENCE LAYER
|  - Dashboards (role-based)  - Chat/Agent UI  - Forms/Wizards  - Notifica
|  (Director view is just a permissioned slice of the same data)
+------------------------------+-----------------------------+------------
                               |                             |
                               v                             v
+-------------------------------------------------------------------------
|                           AGENT & ORCHESTRATION LAYER
|  Agent Runtime (LLM tool-calling)
|   - Conversation state (auditable)
|   - Retrieval (KB) w/ citations
|   - Planning (strategy draft)
|   - Executes tools via Tool Gateway
|
|  Workflow Engine (deterministic)
|   - Tier playbooks/state machines
|   - Gates, required fields, SLAs, dependencies
|   - Approval checkpoints
+------------------------------+-----------------------------+------------
                               |
                               v
+-------------------------------------------------------------------------
|                        POLICY / CONTRACT ENFORCEMENT LAYER
|  Tool Gateway (PEP) + Policy Decision Point (PDP)
|   - RBAC/ABAC (team/BU/region/program)
|   - Field-level authority rules
|   - Approval required?
|   - Idempotency / rate limit / safety checks
|
|  Contract Validator
|   - Schema validation for plans/actions
|   - State machine invariants
+------------------------------+-----------------------------+------------
                               |
                               v
+-------------------------------------------------------------------------
|                           PLATFORM SYSTEMS OF RECORD
|  Canonical Campaign Spine (SoR)
|   - campaign_id (global key)
|   - strategy artifacts (versioned) + approvals
|   - execution mappings (asana/workfront/internal ids)
|   - normalized work graph (tasks/assets/approvals)
|
|  Performance Platform (SoR for metrics)
|   - metric ingestion + normalization
|   - measurement mapping (campaign_id <-> keys: ASIN/UTM/etc.)
|   - aggregates + anomaly signals
|
|  Knowledge Platform (SoR for indexed KB)
|   - doc ingestion + chunking + embeddings
|   - metadata tagging + optional ACL mirroring
|   - citation service
+------------------------------+-----------------------------+------------
```

```
                              |
                              v
+-------------------------------------------------------------------------
|                             INTEGRATION HUB
|  Connectors (read/write) + Sync Engine
|   - Workfront adapter (mirror + writeback)
|   - Asana adapter (mirror + writeback)
|   - Internal tool adapters (measurement, DAM, capacity, etc.)
|   - Event/webhook intake + retries + dead-letter
|   - Conflict resolution + field precedence
+-----------------------------+------------------------------+------------
                              |
         +--------------------+----------------------+--------------------
         |                                           |
         v                                           v
+------------------------+                 +---------------------+   +----
| EXECUTION SYSTEMS      |                 | KNOWLEDGE SOURCES    |   | ME
| Workfront / Asana      |                 | S3 / Quip / SP onprem|   | in
| tasks/assets/approvals |                 | docs/templates       |   | pe
+------------------------+                 +---------------------+   +----
                                                   ^
                                                   |
                                     +-------------------------------+
                                     |  ON-PREM KNOWLEDGE BRIDGE     |
                                     |  (runs near SharePoint)       |
                                     |  crawl/parse/tag -> KB ingest |
                                     +-------------------------------+
```

Key idea: Agents never talk to Workfront/Asana/measurement/SharePoint directly.
They talk to your **Tool Gateway**, which enforces **contracts + policy + approvals**.

---

2) The systems and what they "own"

Canonical Campaign Spine owns

- `campaign_id` (global identity)
- intent, tier, owners, goals, KPI targets, key dates
- versioned strategy artifact + approvals
- mapping to external execution objects (Workfront/Asana project IDs)
- normalized "work graph" snapshot (read model) for dashboards/agent

Execution systems own

- day-to-day task status, asset proofs, operational workflow states
- detailed production fields that only they understand

Performance platform owns

- metric truth (actual spend, conversions, ROI, pipeline impact)
- anomalies, comparisons, rollups
- measurement-key mapping (how to attach metrics to campaigns)

Knowledge platform owns

- indexed docs, templates, playbooks, citations

- metadata tags and (optionally) mirrored ACLs

---

3) Core contracts (the guardrails)

These are what make “user-created agents” feasible.

Contract A: Campaign

- required fields by campaign type/tier
- state machine invariants (can’t “create execution project” if missing required fields)
- authority model (who can change what)

Contract B: Strategy artifact

- structured plan schema (channels, assets, timelines, risks)
- citations required for certain sections (brand guidance, compliance)
- versioning + approval

Contract C: Action contracts (tools)

- `create_workfront_project(campaign_id, payload_schema_ref, …)`
- `update_asana_tasks(changeset_schema, …)`
- `query_performance(campaign_id, metric_set, window, …)`

No free-text writes to external systems.

---

4) On-prem SharePoint: recommended pattern

On-Prem Knowledge Bridge (runs “inside” your network)

Responsibilities:

- authenticate to SharePoint on-prem (Kerberos/NTLM/claims—whatever your environment uses)
- crawl incremental changes
- extract text + metadata (site, library, owner team, doc type, version)
- (optional) extract ACLs / AD groups
- push into Knowledge Platform ingestion API

This avoids runtime dependency on SharePoint for every question and supports your “viewers without upstream accounts” constraint.

---

5) Data flows (sequence views)

Flow 1: Marketer creates campaign → Workfront/Asana project created

text Copy code

```text
User -> Agent UI: "I want to run ASIN penetration email"
Agent -> KB Retrieval: pulls playbook + template snippets (with citations)
Agent -> User: asks missing required fields (audience, ASINs, KPI targets,
User -> Agent: provides inputs
Agent -> Workflow Engine: selects playbook (Email/ASIN) + tier gates
Agent -> Contract Validator: validates Strategy schema + required fields
Agent -> Approval: (if needed) collects approvals
Agent -> Tool Gateway: request create_execution_project(campaign_id, targe
Tool Gateway -> PDP: check role, state, approvals
Tool Gateway -> Integration Hub: create project + tasks from template
Integration Hub -> Workfront: create project
Integration Hub -> Campaign Spine: store mapping + baseline
Dashboards: show status + next steps immediately
```

Flow 2: Any user asks performance/status ("director view" is just higher scope)

text Copy code

```
User -> Agent: "Are we on track and how is performance?"
Agent -> Tool Gateway: query_campaign_health(scope=user_scope)
Tool Gateway -> Campaign Spine: status, blockers, dependencies
Tool Gateway -> Performance API: KPIs, trends, anomaly flags
Agent -> KB Retrieval (optional): "latest guidance" / playbook references
Agent -> User: summary + charts/insights + citations + recommended actions
If user says "push decision":
  Agent -> Tool Gateway: propose update (structured changeset)
  PDP: checks if allowed / approval required
  Integration Hub: writes to Workfront/Asana
```

---

6) Field authority + conflict resolution model

You need an explicit matrix (high-level):

- **Objectives/KPI targets**: Campaign Spine authoritative
- **Task status / asset approval status**: Execution system authoritative
- **Budget actuals / conversions / ROI**: Performance platform authoritative
- **Playbooks/templates/brand rules**: Knowledge platform authoritative
- **Dates**: depends:
  - planned dates in Spine
  - execution actual dates in Workfront/Asana
  - reporting shows both (plan vs actual)

Conflict resolution policy is implemented in the Integration Hub (and enforced by Tool Gateway).

---

7) "User-created agents" design in this architecture

You ship an **Agent Studio** that lets users *configure*, not invent power:

Users choose:

- persona + prompt
- which workflows/playbooks it can invoke
- which KB packs it can use
- which tools it can call (read-only, write-with-approval, write-direct)
- scope tags (team/BU/program)

- output templates (schemas)

They cannot:

- add arbitrary HTTP calls
- bypass contracts/policy
- store secrets/credentials
- write into Workfront/Asana fields outside schemas

Everything still routes through **Tool Gateway + PDP/PEP + Contract Validator**.

---

8) Build vs buy placement

This architecture works with:

- **QuickSuite as UX/agent shell/dashboards**, with your platform behind it
- **Salesforce Agentforce** as UX/agent shell, with your platform behind it
- **Custom UI** (web app) calling your Agent Runtime directly

The "spine + policy + integration hub + measurement join + on-prem bridge" remains the same.

---

9) What I'd define next (to make this actionable)

Pick one "thin slice" (Tier 2 Launch or Tier 1 Event) and I'll produce:

1. **Component responsibilities + APIs** (Tool Gateway endpoints, Spine schemas)
2. **Authority matrix** (field-level owners + sync direction)
3. **Connector contracts** (Workfront/Asana minimal objects and idempotent write model)
4. **On-prem SharePoint bridge spec** (crawl strategy, metadata tags, ACL option)

Tell me which slice you want first: **Launch (Workfront)** or **Event (Asana)**.

We might connect with Adobe DAM for creatives too, to tag creatives created by design team.

Makes sense — bringing in **Adobe DAM** (usually **Adobe Experience Manager Assets / AEM Assets**) adds an "asset system of record" to your platform. The key is to treat DAM as **authoritative for asset identity + metadata**, while Workfront/Asana remain authoritative for *work*, and your spine remains authoritative for *campaign intent + requirements*.

Here's how I'd extend the architecture cleanly.

---

1) Where DAM fits in the platform

Add a new integration archetype:

Asset systems (DAM)

Pattern: **Register → Link → Tag → Govern → Measure**

- Register/ingest assets created by design team
- Link assets to campaigns and work items
- Tag consistently (campaign_id, channel, format, rights, AI usage, brand compliance, etc.)
- Use metadata for governance and reporting

Authoritative

- DAM owns: `asset_id` , renditions, file metadata, rights, approvals (if DAM handles)
- Campaign Spine owns: required asset specs + what assets are needed
- Execution owns: "asset needed by X date" tasks + handoffs
- Measurement owns: performance for assets (if you can attribute creative-level metrics)

---

2) Canonical data model additions

A) Asset contract (new canonical object)

Minimum fields:

- `asset_id` (canonical)
- `external_asset_ref` (DAM ID/URL)
- `campaign_id`
- `work_item_id` (optional)
- `asset_type` (image/video/copy/deck/landing page)
- `channel` (email/paid social/display/site/event)
- `format` (dimensions, duration, file type)
- `status` (draft/ready/approved/archived)
- `owner_team` / `creator`
- `rights` / expiry / region restrictions
- `ai_generated` (yes/no/unknown) + provenance
- `tags` (taxonomy)
- `renditions[]` (pointers)

B) Asset spec contract (ties strategy → production)

From your strategy artifact:

- "Required assets" becomes structured:
  - count
  - format
  - deadlines
  - acceptance criteria
  - brand rules references (citations to KB)

This is how your production agent can propose a plan deterministically.

---

3) DAM integration flows (what actually happens)

Flow A: Campaign creates "asset requirements"

1. Agent generates strategy
2. Workflow engine derives required assets (structured spec)
3. Workfront project created with asset tasks (or workfront "documents" links)
4. Asset specs stored in Campaign Spine

Flow B: Design team produces creatives → DAM tagging

Two ways to do tagging reliably:

Option 1 (recommended): System-driven tagging

- When Workfront marks an asset task "Ready for DAM"
- Integration Hub triggers:
  - create/update asset metadata in DAM
  - apply tags (campaign_id, channel, spec_id, etc.)
- DAM becomes consistently tagged without relying on humans

Option 2: Human-assisted tagging

- Designer uploads to DAM
- Agent/automation finds new assets in DAM folder and suggests tags
- Designer confirms (safer if DAM taxonomy is strict)

Flow C: Agent answers "show me creatives for Campaign X"

- Query Campaign Spine for campaign_id
- Pull linked assets from DAM (via integration hub cache)
- Show asset status (draft/approved) and missing specs

Flow D: Post-delivery learning

- Store creative metadata + performance in spine/metrics store:
  - "which creative formats performed best"
  - "AI-generated usage rate"
  - "approval cycle time"
  - "time-to-first-draft"

---

4) Key design decision: source of truth for asset approval

You must pick one:

Model 1: Workfront is approval system, DAM is repository

- Approval state in Workfront
- DAM just holds files + tags + renditions
- Pros: aligns with production flows
- Cons: DAM metadata may lag

Model 2: DAM is approval system (or co-equal)

- Final approval state in DAM
- Workfront links out
- Pros: clean for asset governance/rights
- Cons: changes how production teams work

Most orgs do **Model 1** initially, then migrate some approvals to DAM later.

---

5) What to add to the Integration Hub

Add a **DAM Adapter** with capabilities:

- `find_assets(query_by_tags)`
- `get_asset(asset_id)`
- `set_asset_metadata(asset_id, tags, fields)`
- `create_asset_placeholder(spec)` (optional: creates a "slot" for required assets)
- `link_asset_to_campaign(asset_id, campaign_id)`
- `publish_rendition(asset_id, channel)` (optional)

All writes go through Tool Gateway → policy checks.

---

6) Governance & reporting (why DAM is worth it)

Once creatives are tagged with `campaign_id`, you unlock:

- "% campaigns with all required creatives ready"
- "approval cycle time by team"
- "creative reuse rate"
- "AI-generated creative usage"
- "creative-to-performance correlation" (if measurement can attribute)

This directly powers your "director capability" (but also helps PMMs and production teams).

---

7) Practical MVP path for DAM integration

Start small:

1. **Tagging**: ensure every asset created ends up with `campaign_id + asset_type + channel`
2. **Linking**: Spine stores asset references and status
3. **Read-only dashboards**: creatives ready vs missing
4. Later: placeholders, rendition publishing, rights checks, creative-level performance

---

Next step (so we keep it concrete)

Tell me which Adobe DAM you mean:

- **AEM Assets** (most common), or
- **Adobe Experience Platform / Content Hub**, or something else.

Without waiting for that answer, I can still draft the **DAM integration contract** (schema + APIs + event triggers) and show exactly how it plugs into Workfront + Asana flows with no duplication.

Now think one more thing, one of my sister team also have asana, workfront and few more (i'll confirm later), I was thinking to combine the effort to have one integration hub we both can use, That's a smart move *if* you set it up as a **shared integration platform** (with tenancy + governance), not "one team's bot that happens to connect to another team's tools." Otherwise you'll get political friction and security pushback fast.

Here's the architect-level way to design a **single Integration Hub** both orgs can use.

---

1) The right framing: "Integration Hub as a platform product"

Platform contract

The hub exposes **stable internal APIs** like:

- `CreateProject(execution_system, payload)`
- `SyncProject(external_id)`
- `UpsertWorkItem(campaign_id, changeset)`
- `GetWorkGraph(campaign_id)`
- `RegisterAsset(dam_ref, tags)`
- `QueryPerformance(scope, metrics, window)`
- `IngestKnowledge(source, metadata)`

Each consuming team (your Marketing OS, their apps) builds on these contracts.

Don't share raw connectors

Don't let both teams "just use the same Workfront connector creds."
Share the **hub**, not the secrets.

---

2) Multi-tenancy model (this is the make-or-break)

You need at least **three** layers of separation:

A) Tenant (Org / Business unit)

- Tenant = your org boundary (or a BU boundary if you must)
- Each tenant has:
  - its own connector instances (Workfront, Asana, DAM, etc.)
  - its own sync configs (which projects, fields, mappings)
  - its own secret store entries

B) Environment (dev/stage/prod)

- tenants per environment
- prevents accidental cross-writes

C) Identity + scope (who is calling the hub)

- Every API call carries:
  - `caller_service`
  - `user_context` (optional)
  - `tenant_id`
  - `requested_scope`

Then the hub enforces:

- this caller can only access these tenants/scopes

- write actions may require approvals

Result: both teams can use the hub without data leakage.

---

3) Connector instance strategy

Workfront

- Often requires enterprise-level configuration.
- Best practice: **one connector instance per tenant** (per Workfront domain/org)
- Some tenants may share one Workfront instance but use different portfolios/queues — still model them as separate tenants in the hub.

Asana

- Same story. Separate connector instances per Asana org/workspace.

"Sister team has more tools"

No problem — tools become connector plugins. Hub stays stable.

---

4) Data & event model: normalize once, reuse everywhere

Normalized objects inside the hub

- `Project`
- `WorkItem` (task/approval/asset request)
- `DependencyGraph`
- `Asset`

These integrations are on their roadmap for this year (2026)

Nice — if this is on both roadmaps for **2026**, you have a real window to turn it into a **shared platform initiative** instead of two parallel connector efforts.

Here's how I'd architect and *operate* it so it actually works cross-team.

---

1) Treat it as a shared product: "Integration Hub Platform"

If you frame it as "we'll share connectors," it'll die in security + ownership debates.

Frame it as:

- **One Integration Hub platform**
- Multiple **tenants**
- Multiple **consumer products** (your Marketing OS, their tooling, future agents)

Deliverable: a stable internal API + clear governance.

---

2) The 2026 plan that avoids getting stuck

Phase 0 (now): Alignment artifacts (2–3 docs, not months of meetings)

Create three 1-pagers both teams can sign:

1. **Shared Scope Charter**

- In-scope: Asana, Workfront, DAM, Knowledge sources, Measurement systems
- Out-of-scope: building UIs, replacing Workfront/Asana
- “Hub owns integration correctness + sync; products own experiences”

2. **Tenancy & Security Model**

- Tenant definition (BU/org/workspace)
- Connector instances per tenant
- Secrets isolation
- Audit requirements

3. **Contract Set v1**

- Minimal internal APIs: `CreateProject` , `SyncProject` , `UpsertWorkItem` , `GetWorkGraph` , `RegisterAsset` , `QueryPerformance`
- No product-specific payloads. Everything schema-based.

If you do only these three, you’ll move 5× faster.

---

3) Multi-tenancy: the non-negotiable architecture requirement

Because it’s cross-team, you must assume:

- different Asana orgs/workspaces
- different Workfront setups
- different permission models
- different compliance expectations

The hub must enforce:

- **Tenant isolation** (data + secrets + configs)
- **Scoped access** (caller + user context)
- **Write controls** (policy gates + approvals)

Otherwise your sister team will opt out (rightly).

---

4) Ownership model (so it doesn’t become “everyone’s problem”)

You want one team to “own the hub,” but with shared influence.

A workable model:

- **Hub Platform Owner** (one team, accountable for uptime + releases)
- **Connector Owners** (per connector: Workfront owner, Asana owner, DAM owner)
- **Consumer Owners** (your product + sister product)
- **Steering** monthly: prioritize connector/features by business impact

Also define:

- SLOs (sync latency, webhook reliability, write success rate)
- on-call ownership

- versioning policy for API contracts

---

5) How to split work so you don’t block each other

This split works in practice:

Hub team builds:

- Connector framework (auth, retries, idempotency, rate limits)
- Normalized object model (Project, WorkItem, Asset, Mapping)
- Sync engine + event intake
- Tool gateway hooks (policy checks)
- Observability (metrics, audit logs, replay)

Each consumer team builds:

- their UI/agent experiences
- domain workflows/playbooks
- role-based views
- business logic that maps their concepts → hub contracts

This prevents the hub from turning into “custom requests for every team.”

---

6) Roadmap sequencing for 2026 (pragmatic)

To avoid boiling the ocean, sequence by “highest shared value”:

Q1: Foundation + Read-only sync

- Hub skeleton + tenancy + secrets + audit
- Read-only sync from Workfront + Asana into normalized work graph
- Consumers can already show status and blockers reliably

Q2: Create project + controlled writeback

- Project creation from templates
- Writeback limited to safe fields (assignments/dates/status notes)
- Approval gating for higher-risk writes

Q3: Knowledge + DAM integration

- On-prem SharePoint ingestion bridge (if needed)
- DAM tagging/registration pipeline
- Asset readiness + compliance views

Q4: Measurement joins + “performance everywhere”

- Canonical measurement mapping layer
- Standard portfolio metrics API
- Creative-to-performance correlation (optional)

This sequencing gives visible value early, and makes later agent use cases much easier.

---

7) Key design choices to settle early (to prevent rework)

These are the decisions that, if delayed, cause architecture churn:

1. **Canonical ID strategy**
- Global `campaign_id` or `project_id` owned by the spine vs hub
- How external IDs map to canonical IDs
2. **Field authority matrix**
- Which fields are authoritative in Workfront/Asana vs spine vs measurement
3. **Writeback policy**
- What's "safe auto-write," what requires approval, what is forbidden
4. **Template strategy**
- Where templates live (KB vs execution system templates)
- How templates map to normalized work items
5. **On-prem SharePoint access**
- Crawl/index vs federated search
- ACL mirroring vs platform RBAC

---

8) What you should do next week (concrete)

If you want to kick this off cleanly, do these three things:

1. **Pick the first shared "thin slice"**
   Example: "Create campaign → Workfront project created → status sync visible in dashboards."
2. **Define the "Contract v1"**
   Just 5 endpoints and 3 object schemas (Project, WorkItem, Mapping). Keep it tiny.
3. **Set tenancy boundaries**
   Even if both teams share one Workfront instance, model separate tenants logically.

If you want, I can turn this into a concise **"Integration Hub Platform RFC (v0)"** with:

- architecture diagram
- tenancy + auth model
- API contracts v1
- 2026 milestone plan
- risk register and mitigations

Just tell me whether your org prefers **AWS-native** stack (API Gateway/Lambda/EventBridge/etc.) or a containerized platform (EKS), and I'll bias the RFC accordingly.

How are we handling the oAuth issue for asana or slack or adobe workfront? Be concise

Use a **single "integration owner" connection per tenant** and keep OAuth out of the user path.

- **One OAuth app per connector** (Asana/Slack/Workfront), **admin-approved** where possible.

- **Connect once** using a dedicated **service user** (or org-level install), store **refresh tokens** encrypted (KMS/Secrets Manager), rotate and monitor.
- **Hub acts as the app**: it uses the stored token to sync/read/write; end users authenticate only to your platform.
- **Permissions enforced in your platform** (RBAC/ABAC + approval gates). Don't rely on Asana/Slack/Workfront membership for viewers.
- **Slack**: prefer **workspace app install** (bot token) for posting/listening; avoid per-user tokens unless you need "post as user."
- **Write actions**: always via **schema-validated tool calls** + idempotency + audit logging; higher-risk writes require approval.
- If a SaaS supports it, prefer **admin-consented/org-wide** auth over "random user connects it."