frends

Integration Center for Enablement (C4E)

A Complete Guide

Antti Toivanen

Head of Product & VP
Frends iPaaS

Integration is undergoing a necessary transformation. For too long, centralized control models have tried to keep up with increasingly decentralized development environments and failed. Today, what organizations need is not necessarily tighter control, but smarter enablement.

That's where the Integration Center for Enablement (C4E) comes in.

The C4E model acknowledges what we've seen across countless organizations: delivery speed, scale, and developer autonomy are essential. But so are governance, security, and reuse. Balancing these is not just a technical challenge — it's an organizational one.

This playbook exists to support that shift. It's built on decades of experience navigating complex integration landscapes, across technologies and industries. And while some of the ideas here reflect lessons learned at Frends, the real goal is to help teams modernize their integration capability regardless of platform.

The industry doesn't need more vendor promises. It needs frameworks that work. If this helps move the conversation forward, then it's done its job.

Antti Toivanen

Table of Contents

1 Introduction
Integration Center for Enablement 2

2 Enablement over control
Shifting from ICC to C4E 3

3 The Integration C4E Team
Platform stewardship and Enablement 6

3.1 Integration Maturity Benchmark 8

3.2 Integration C4E team's responsibilities 11

4 Platform and product thinking for integration 13

5 Governance & Standards
in a Federated Environment 16

6 Integration Delivery Operating Models 21

6.1 Centralized Integration Delivery (Legacy Model) 23

Table of Contents

6.2 Federated Delivery
with Platform Guardrails — 25

6.3 Self-Service and API-first Teams enabled by C4E — 28

6.4 DevOps-Integrated Integration
(CI/CD and Automation) — 31

6.5 Microservices
use with care — 34

6.6 Policy-Enforced Integration via iPaaS
(Microservices through iPaaS) — 37

6.7 Agentic AI–Driven Integration:
Prompting and reusable API tooling — 42

7 Consistency across multiple integration platforms — 45

8 Conclusion:
Building a modern integration capability — 49

01

Introduction

Integration Center for Enablement

Enterprise integration has evolved. In the past, integration work was usually managed by a single central team (ICC). That approach made sense at the time, but it struggled to keep up with today's faster, more decentralized development environments.

In today's world of distributed microservices, multi-cloud applications and fast-paced digital initiatives, the ICC´s strict approach can become a bottleneck.

Rigid centralized control often slows down projects and drives teams to seek workarounds, undermining the governance it intended to enforce.

Enter the Integration Center for Enablement (C4E)

A modern take that flips the script from control to enablement.

A C4E is a cross-functional team tasked with productizing, publishing, and harvesting reusable integration assets and best practices. Instead of one team executing all integration work, the C4E's mission is to empower distributed teams across the enterprise to build integrations quickly and safely.

This model accepts that not every integration has to go through one central platform. Instead, it's about **giving teams the tools and guidance they need to do things right on their own.**

For example, under a traditional ICC, if the Marketing department needed to connect a new CRM system, they would submit a request and wait in the queue for the central team to deliver it. In a C4E model, the Marketing team (or their IT partners) can build the integration themselves using approved tools and standards, with the C4E providing the necessary **platform, training, and guidelines upfront.**

The result is faster delivery to the business and a smoother path to innovation. In fact, organizations that embrace this enablement-focused approach have reported up to 60% shorter delivery cycles and higher team productivity than those sticking to centralized methods.

This playbook outlines transitioning from ICC to the modern C4E, reframing integration practices for micro/miniservices, multi-platform and the DevOps era.

We will cover the mindset shift from control to enablement, the importance of treating integrations and APIs as internal products, various operating models (from centralized to self-service), governance in a federated environment, and strategies to avoid pitfalls like microservice sprawl. This is a practical guide created for **business and IT leaders** to build an integration capability that is agile yet governed, enabling speed and maintaining compliance.

02

Enablement over control

Shifting from ICC to C4E

In the ICC model, the integration team functioned as a Center of Excellence (CoE), a centralized powerhouse of integration experts who executed or approved every integration project. While this ensured a degree of consistency, it also meant that knowledge and execution were “bottlenecked” in one team.

The CoE often unintentionally hoarded expertise, with integration knowledge protected and rationed in the central group.

The side effect? Developers and business units sometimes worked around the ICC to get things done, leading to “shadow IT” integrations that bypass governance entirely.

The C4E flips this approach. Instead of a command-and-control tower, think of C4E as an enablement hub.

The C4E's primary goals are to run the integration/API platform and enable other teams to use it effectively, rather than doing all the integration development itself. This means the C4E focuses on sharing knowledge, reusable assets and best practices across the organization.

It actively promotes the consumption of those assets and supports teams to be self-reliant in building integrations. Integration experts won't disappear, but their role evolves from implementers to coaches and facilitators.

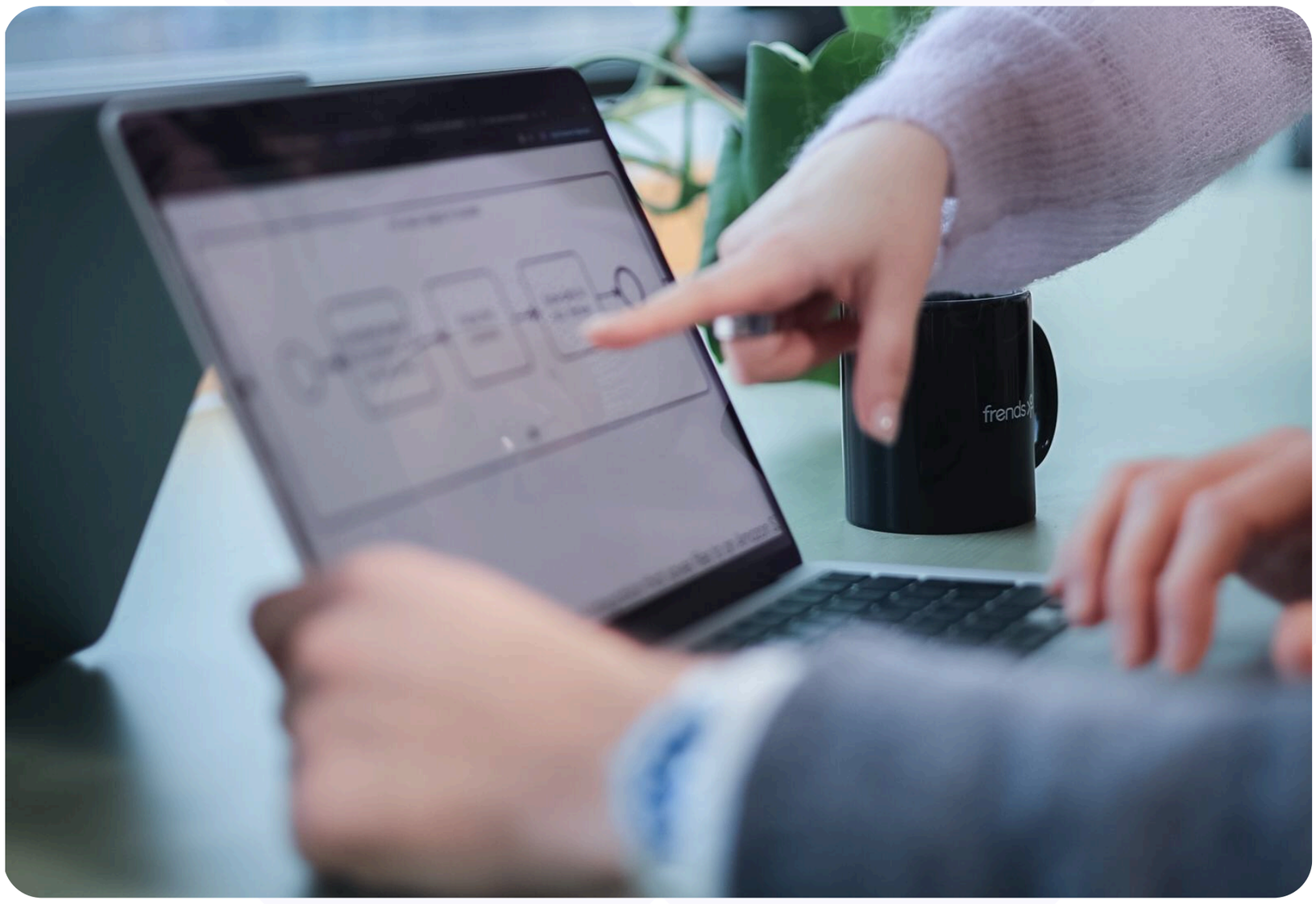
Traditional CoEs could become overwhelmed with demands, causing delays. A well-run C4E, by contrast, distributes the work: many teams can execute integrations in parallel, using common standards and avoiding bottlenecks. This federated delivery means higher throughput and agility, while the C4E ensures outcomes remain aligned to enterprise standards. The culture shifts to one of collaboration: architects and developers in different business units work closely with the C4E, not for the C4E. The C4E provides enablement services like training, templates, and advisory, but it does not need to sign off on every minor change – it trusts teams within a framework of clear rules.

Example:

Under an enablement model, a new e-commerce API might be built by the Retail IT team itself, using the tools, templates, and security standards provided by C4E. The C4E might only get involved to consult on design or to ensure compliance through automated checks. This is a stark contrast to the old ICC approach, where that API would sit in a central backlog waiting for a specialist to implement. The C4E model thereby avoids making a "bottleneck" team that others try to work around – instead, it makes the integration practice scalable by having many capable hands, all following a common playbook.

The shift from ICC to C4E also entails a change in success metrics.

In an ICC, success might be measured by projects delivered by the central team. In a C4E, **success is measured by broader outcomes:** How many integrations are being reused? How much faster can projects go live thanks to the platform? How many teams are actively contributing to integrations? These are signs of enablement. Ultimately, a C4E aims to create a **production-and-consumption model**, where teams produce integration assets (APIs, services) and consume those produced by others, rather than everything being made centrally. This networked model of delivery accelerates digital initiatives while preserving order.



03

The Integration C4E Team

Platform stewardship and Enablement

The C4E is not just a concept, but an actual team (or virtual team) within the organization. It is typically **cross-functional**, drawing members from central IT, business units, and enterprise architecture groups. This blend ensures the C4E has a broad perspective and credibility across silos. Think of the C4E team as both **platform stewards and integration coaches** – they own the integration platforms and standards and enable other teams to use them effectively.

The dual nature of the C4E's work can be thought of in two broad categories: **Platform and Enablement**

On the platform side, the **C4E manages the integration infrastructure as an internal product**, including architecture, tools, and operational aspects. On the enablement side, the **C4E drives the adoption of integration best practices**, reusable components, and supports project teams. The diagram below illustrates these facets and the key responsibilities of a C4E team.

The C4E team's responsibilities span two domains: maintaining the integration Platform (left) and providing Enablement to teams (right).

In platform stewardship, C4E defines architecture, security and automation standards, and manages the integration environment (deployment models, monitoring, support, etc.).

In enablement, C4E sets up API strategy and governance, curates best practices and reusable assets, accelerates delivery through training and onboarding, and generally acts as the integration "coach" for the organization. This structure positions the C4E as both the guardian of the platform and the guide for teams.

3.1 Integration Maturity Benchmark

The following maturity model provides **a structured way to benchmark where your organization sits on its journey** from ad-hoc, project-by-project integration toward a fully productized, self-service ecosystem, underpinned by an Integration Center for Enablement (C4E).

Each level represents a step-change in both the platform stewardship and enablement responsibilities of the C4E:

Level 1: Ad-Hoc	Characteristics: Teams build integrations one-off, using bespoke scripts or point-to-point code. There is no shared platform, few standards and no central visibility or reuse.	C4E Role: Reactive firefighting — answering tickets, sharing sample code by request.
Level 2: Defined	Characteristics: A basic integration platform exists (e.g., catalogue, security policies), and a minimal set of guidelines has been documented. Some reusable templates or connectors are available, but adoption is spotty.	**C4E Role:** Publish "starter kits," run workshops on standards, begin curating an initial API catalogue.
Level 3: Enabled	Characteristics: Self-service is real. Development teams can consume platform APIs and templates with little handholding. Automated CI/CD pipelines enforce quality gates. Usage metrics begin to appear in dashboards.	**C4E Role:** Maintain templates and governance-as-code, onboard new teams via training programs, monitor adoption and surface blockers.
Level 4: Managed	Characteristics: Governance is largely automated. The C4E tracks SLAs, error rates, throughput and publishes regular consumption reports. A formal roadmap for new capabilities (e.g., event streaming, low-code connectors) guides evolution.	**C4E Role:** Operate real-time monitoring and alerting, drive continuous improvement cycles, coordinate cross-team "guardrail reviews."

3.1 Integration Maturity Benchmark

Level 5: Optimized	●	Characteristics: Integration is treated as a first-class product. The C4E, in partnership with product managers, uses usage analytics (and even ML-driven recommendations) to proactively evolve the platform. Teams increasingly innovate at the edges, confident in the underlying capabilities.	**C4E Role:** Act as strategic adviser, spotting new use-cases, piloting advanced patterns (e.g., API-mesh, event-driven fabrics) and evangelizing best practices enterprise-wide.

Using the model as a diagnostic and roadmap

Assess current state

1 Run a short survey or workshop with your integration teams to map where they land on each of the five dimensions (platform, governance, enablement, automation, metrics).

Identify gaps

2 Highlight the biggest delta between today's level and the next—e.g., lack of automated compliance (Level 3→4) or missing self-service templates (Level 2→3).

Prioritize C4E initiatives

3 Focus your roadmap on the capabilities that will unlock the greatest productivity gains — whether that's plugging governance into your CI/CD pipeline, expanding connector libraries, or building a consumption dashboard.

Measure progress

4 Re-run the assessment quarterly to track shifts in adoption, reuse rates, cycle times and defect rates. Adjust your C4E's charter and resource allocation accordingly.

5. Build impact

5 By using this maturity model as both a mirror and a guide, the Integration C4E can systematically grow its impact, shifting from reactive support toward proactive platform governance and business-driven innovation.

3.2 Integration C4E team's responsibilities

Concretely, the C4E team's responsibilities include:

Integration strategy and roadmap:

Define the vision for integration in line with business goals. For example, decide on an API-first approach for all new projects or target certain legacy systems for modernization.

Platform architecture & operations:

Build and manage the integration platform(s). This involves defining reference architectures, ensuring security is baked in, setting up CI/CD pipelines for integrations, and handling capacity planning and upgrades. The C4E provides a stable, scalable foundation (whether it's an iPaaS, API gateway, message broker, etc.) on which others can build.

Standards and governance:

Establish enterprise-wide standards for integration design and development, covering areas like API design guidelines, data formats, error handling, logging and security policies. The C4E documents these standards and ensures they are accessible. This role is inherited from the ICC concept of enforcing consistency (e.g. naming conventions, metadata standards, version control), but the C4E enforces them through enablement and tooling rather than dictatorial control.

Reusable assets and tooling:

Develop and curate reusable integration assets – for instance, common API frameworks or templates, connectors/adapters for common systems, shared data models, and "accelerators" (like code snippets or low-code templates). By harvesting and publishing these assets, the C4E productizes integration know-how for easy consumption by teams. Teams can start projects faster by leveraging these building blocks.

Training and support:

Provide training programs, hands-on workshops and documentation to upskill teams on integration tools and best practices. The C4E often runs an internal "integration academy" or community of practice. It may also offer consultation or architects-on-demand for projects, acting as experts who can be pulled in to help solve tricky integration problems. By fostering a knowledge-sharing culture, the C4E boosts overall integration competency across all teams.

Delivery enablement and oversight:

Accelerate projects by onboarding teams to the platform quickly, assisting with initial architecture or design reviews and then stepping back to let them work. The C4E might require a review for only the most critical aspects (e.g. security design or going live to production) as a safeguard. It balances empowering teams with ensuring no major compliance gaps. Essentially, the C4E provides guardrails – if teams stay within them, they can move fast. If they hit an issue, the C4E is there as a safety net.

Metric tracking and continuous improvement:

Monitor key metrics such as the number of integrations delivered by teams, API reuse rate, time-to-market improvements, platform usage, support tickets, etc. This data helps demonstrate the C4E's value (e.g., "we delivered 50% more integrations this quarter with the same resources") and identifies areas for improvement. For example, if certain teams are not adopting the platform, the C4E can reach out to understand why (maybe they need additional support or are missing a capability).

The C4E team is the chief enabler and custodian of integration practices. They do not necessarily own all integration development, but the ecosystem in which it happens. The integration C4E team is 50% platform team, 50% enablement team, depending on the approach to your choice of platform. While ensuring that the platform is in pristine condition and used as intended, the effort put into it varies depending on approach: fully on-premises own integration platform, hybrid iPaaS or full-cloud native iPaaS without hybrid. On the other side, they support scale and governance by equipping other delivery teams with standards, tools, and reusable components. This way, the enterprise benefits from both centralized expertise and decentralized execution.

04

Platform and product thinking for integration

At the heart of the C4E model is a mindset shift: **APIs and integrations should be treated like real products** — not just one-off fixes, side projects, or technical afterthoughts.

This means shifting to a platform/product mindset in which integration capabilities are managed through their full lifecycle, and where other teams (the “customers”) are encouraged to use and reuse these capabilities.

What does it mean to consider an API or integration as a product? It means:

1. It has a clear lifecycle

from idea to design, development, deployment, version updates and eventually retirement. The C4E ensures there are processes for each stage (e.g. design reviews, publishing version 2.0 of an API, deprecating an old service gracefully, etc.).

2. It has an owner or team

responsible for its maintenance and evolution (often the producing team, with the C4E setting up ownership guidelines).

3. It is designed with the consumer in mind:

just as an external product API would be designed for ease-of-use, performance and reliability, internal APIs should be treated equally. This includes providing good documentation, self-service access (through an API portal or repository), and collecting feedback from consumers.

4. It is measured by its adoption and value

For instance, reusing an internal order management API across five projects is a success metric. The C4E helps publish such metrics to highlight the benefits of reuse.

Practically, adopting product thinking leads to establishing an internal API (or integration) marketplace. Many organizations create an API catalog or developer portal accessible to all developers in the company. Through this portal, teams can discover existing APIs/integrations, request access, read documentation, and even publish their own for others to use. The C4E often curates this marketplace. By publishing and “marketing” reusable assets, it drives greater consumption and collaboration.

The C4E facilitates this scenario by ensuring such APIs exist and meet quality standards, while encouraging teams to harvest their integration work to be reused by others.

Adopting platform thinking also means the integration platform itself is a product. The iPaaS or integration tools provided by the C4E are treated as an internal product/service that needs continuous improvement and support. The C4E might operate an "integration platform as a service" within the company – where they onboard new teams, gather requirements for new features (e.g. need a new connector or a better monitoring dashboard), and iterate on the platform's capabilities just like an external vendor would.

In some organizations, the C4E even has a product manager role for the internal integration platform.

The **benefits of this product mindset are significant**. It leads to **higher reuse of integration assets and fewer redundant** efforts, because teams are aware of and leverage what already exists. It also **improves quality**, since products are maintained and refined over time (instead of "fire-and-forget" scripts that accumulate technical debt).

And it **dramatically speeds up delivery:** one study by Mulesoft found that a C4E approach enabled projects to deliver integrations 3x faster and with 300% more productivity by reusing assets rather than rebuilding everything. When every integration is built as a potential reusable product, the whole organization's integration capability is composed of building blocks, enabling rapid assembly of new solutions.

We recommend establishing clear API/integration lifecycle management practices to implement this.

For instance, use source control and CI/CD for integration code (more on that can be found under the section “DevOps-Integrated Integration (CI/CD and Automation)”), have a versioning scheme for APIs (with deprecation policies so consumers have time to migrate), and maintain documentation as a first-class deliverable.

The C4E can provide templates for API documentation, service level agreements (SLAs) for internal services and even an internal “developer portal” where these products are showcased.

By treating integration assets as products, the enterprise creates an internal ecosystem of services that teams can rely on, accelerating innovation and reducing duplication of effort.

> Example:
>
> Imagine a team that needs to integrate with the customer's master data. Under a productized approach, they might find that there’s already a “Customer API” available in the internal catalog. Instead of building a new integration from scratch, they can utilize the already existing API, saving resources and ensuring consistency in accessing customer data enterprise-wide access. The team that owns the Customer API treats it as a product: they have versioned it, documented usage guidelines, and maybe even published a roadmap (e.g. “API will support new fields next quarter”).

05

Governance and Standards

in a Federated Environment

Moving to a federated integration model, with several teams building integrations, does not mean abandoning governance. In fact, governance becomes even more critical – it's the glue that holds the distributed model together.

The C4E must establish a clear governance framework that all teams follow, ensuring that whether an API is built by Team A or an integration workflow by Team B, they all comply with the agreed-upon standards for quality, security, and interoperability.

Key governance elements to institute:

Security Policies

All integrations and APIs must adhere to enterprise security requirements. For example, APIs should implement standard authentication and authorization (OAuth 2.0, JWT, etc.), use encryption for data in transit and at rest as appropriate, and not expose sensitive data without proper controls. The C4E works with the security team to define these policies and possibly bake them into the platform (e.g., providing a centralized API gateway or service mesh enforcing authentication).

API Design and Naming Guidelines

The organization should have a consistent style for APIs (naming conventions for endpoints, consistent error response formats, use of HTTP verbs, etc.) and for other integration artifacts (e.g. consistent naming of integration jobs, mappings or queues). Consistent design makes it easier for developers to use any API because they look familiar. A design review checklist or an automated linting tool can help enforce this.

Data and Metadata Standards

If the enterprise has canonical data models or standard schemas (for example, a common format for customer data or product data), the C4E should enforce their usage in integrations. This avoids the situation where each team defines its own data format, causing translation overhead. Metadata management (like clear documentation of what data is exchanged, field definitions, etc.) is also important, so integrations are not black boxes.

Lifecycle Management

Standards for versioning of APIs/integrations (e.g. semantic versioning), deprecation policy (how long will an old version be supported after a new one is out), and documentation of changes. Every integration should be traceable through its lifecycle – from development to testing to production, with change management in place. This also includes DevOps processes – e.g., any production deployment of an integration must go through the CI/CD pipeline and meet quality gates.

Key governance elements to institute:

Documentation and Discoverability

Every integration (API, workflow, etc.) must be properly documented – ideally in a centralized portal or repository. Documentation should include purpose, owners, usage instructions, dependencies, and compliance considerations. The C4E might enforce, for instance, that an API will not be published for others to use until an API specification (like an OpenAPI document) is provided and reviewed.

Quality and Testing

Define testing protocols for integrations – e.g., unit tests for integration flows, contract testing for APIs, performance testing for critical services. Also, set SLAs/SLOs for integrations if needed (e.g. an API should respond within 200ms for 95% of calls, or a nightly batch completes by 6 AM daily). Monitoring standards (every service should log to the central logging solution and expose metrics) also fall here.

Compliance and Regulatory

For industries with compliance needs (GDPR, HIPAA, etc.), European companies that require their data to be stored within the EU, businesses with other regulatory needs or even those cautious about political instability, the C4E must ensure integrations comply with data handling rules. This might involve classifying data in integrations and applying rules (for example, PII data must be masked in logs). Governance includes ensuring those rules are uniformly applied regardless of who builds the integration.

The C4E should document these standards in an **Integration Guidelines handbook**

an internal guide to ensure that every department and third-party follow the same rules and restrictions. However, more than documentation, C4E should strive for **automated governance where possible**. Automated processes and tools can enforce many standards so that teams get immediate feedback if they violate a rule.

For instance, a CI/CD pipeline can include a static code analysis or linting step to check an API specification against the style guide. An integration platform might have built-in policy enforcement – e.g., the platform could automatically reject any service that doesn't include specific security headers or schema validations.

This kind of "governance as code" is key to scaling federated delivery. It moves governance from after-the-fact manual reviews to proactive, built-in checks. As the Wikipedia entry on ICC notes, the highest maturity ("self-service ICC") achieves independent innovation by "strict enforcement of a set of integration standards through automated processes enabled by tools and systems." – that is precisely what a C4E aims to do.

Another governance mechanism is establishing an **Architecture or Governance Board** that includes C4E members and representatives from development teams. This board can periodically review new integration proposals for alignment with standards, especially for major initiatives, and handle exceptions. The emphasis, however, is on lightweight governance – we don't want to recreate a heavy change of control bureaucracy, but rather an enablement forum that ensures critical issues are caught early.

Compliance is non-negotiable: Even in a multi-platform, multi-team world, certain things must be true for all integrations.

For example, if the policy is "all personal data transfers must be logged and auditable," then every integration (no matter the technology) must build that in. The C4E's role is to communicate these rules clearly, provide tools to achieve them, and monitor compliance. Regular audits or reports can be used – e.g., the C4E might run a scan to verify every API is registered in the portal and uses approved authentication.

It's worth noting that **strong governance actually enables more decentralization.** When teams know the guardrails and those guardrails are firm, they have the confidence to build things themselves. Conversely, if governance is unclear or inconsistent, organizations may revert to centralizing out of fear. The playbook, therefore, stresses investing in governance early in the C4E journey. As one 2024 integration blog put it, C4E creates "standardized practices, guidelines, and templates for API design, development, and governance, ensuring project consistency and reducing errors or inconsistencies". By doing so, even a variety of teams and technologies can produce integrations that look and behave like they came from one unified program, which, in effect, they did, guided by the C4E.

06

Integration Delivery

Operating Models

Every organization's integration journey is different. Some may start with a fully centralized model and gradually move toward federated or self-service delivery.

Others might already be decentralized and need to introduce more governance. It's useful to understand several operating models for integration delivery, from the old-school centralized approach to the modern autonomous approach.

Below is a visualization that maps different models against two key dimensions: the **degree of development centralization vs. distribution, and the degree of compliance control (strict vs. loose).**

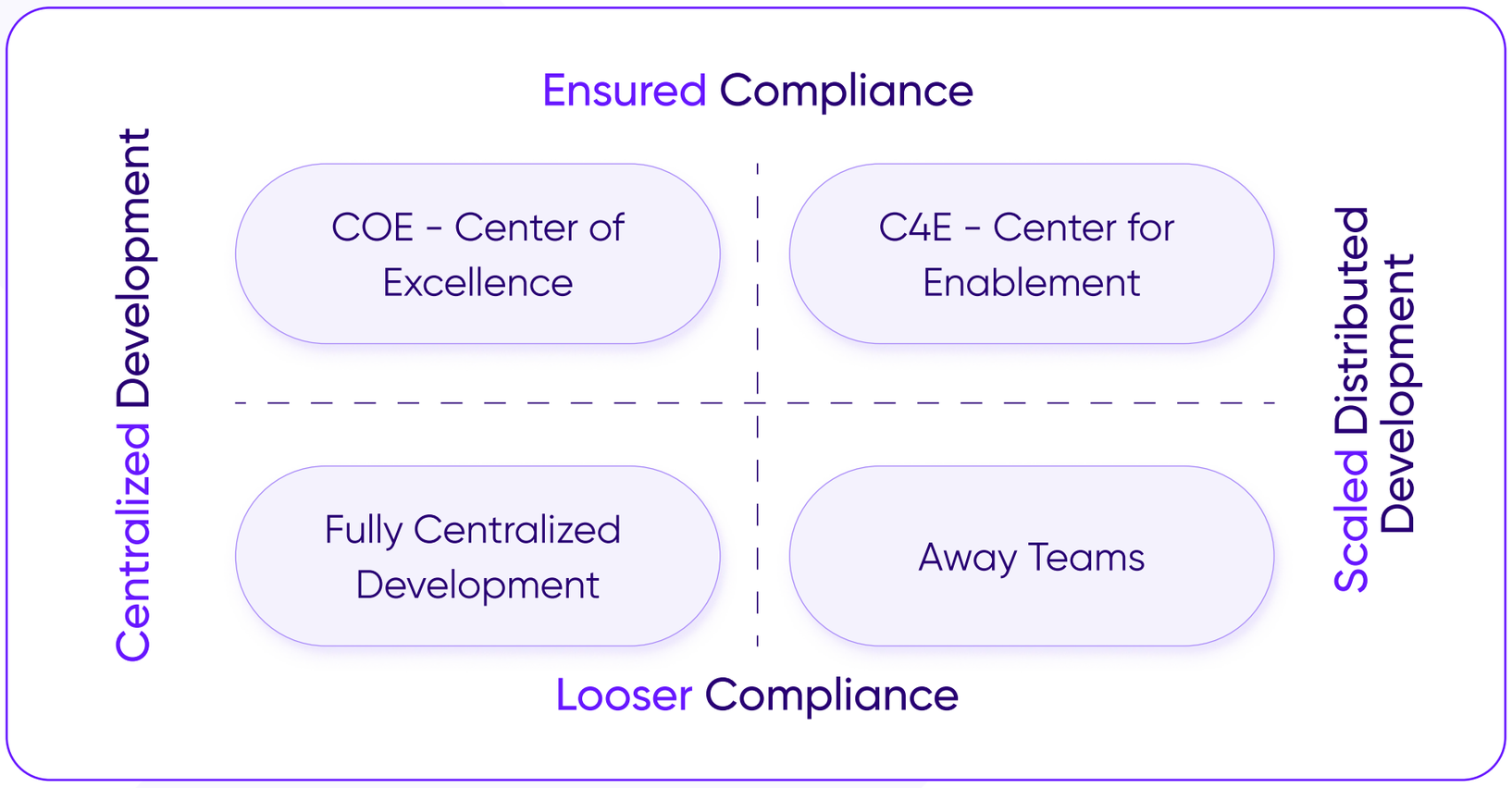
Align your team topology with your business goals.

Alignment of integration operating models with business goals of scale and compliance. A traditional CoE (Center of Excellence) or ICC emphasizes ensured compliance through centralized development (upper-left), but can struggle to scale delivery.

C4E (Center for Enablement) strives to maintain high compliance and scaled, distributed development (upper-right) by enabling teams within guardrails. Fully Centralized Development (bottom-left) has neither the agility nor necessarily the enforcement if standards are not mature – it represents an outdated model where one IT team does everything but without formalized best practices.

At the opposite extreme, "Away Teams" or ad-hoc decentralized development (bottom-right) maximize speed by allowing teams to work independently with looser compliance, often leading to integration sprawl and inconsistency. **The C4E model aims to be the optimal quadrant, balancing speed and innovation with governance and consistency.**

Most organizations will recognize elements of these models in their environment. Below, we detail four primary operating models and how the C4E playbook addresses each.



6.1 Centralized Integration Delivery (Legacy Model)

In a **fully centralized model**, a single team (the old ICC or integration CoE) is responsible for designing, building, and operating all integrations and APIs in the enterprise.

Business units request integrations from this central team, which prioritizes and delivers them. This model was common in the past, especially when integration technology was complex (e.g., on-premise ESBs) and scarce expert skills had to be concentrated.

Advantages

Centralized delivery offers strong control and oversight. Standards can be enforced by direct supervision, since the same team is doing all the work. Duplicated efforts are minimal because everything funnels through one group. It can be efficient for a small number of projects and ensure a high level of compliance by default (nothing goes live without central approval).

Drawbacks

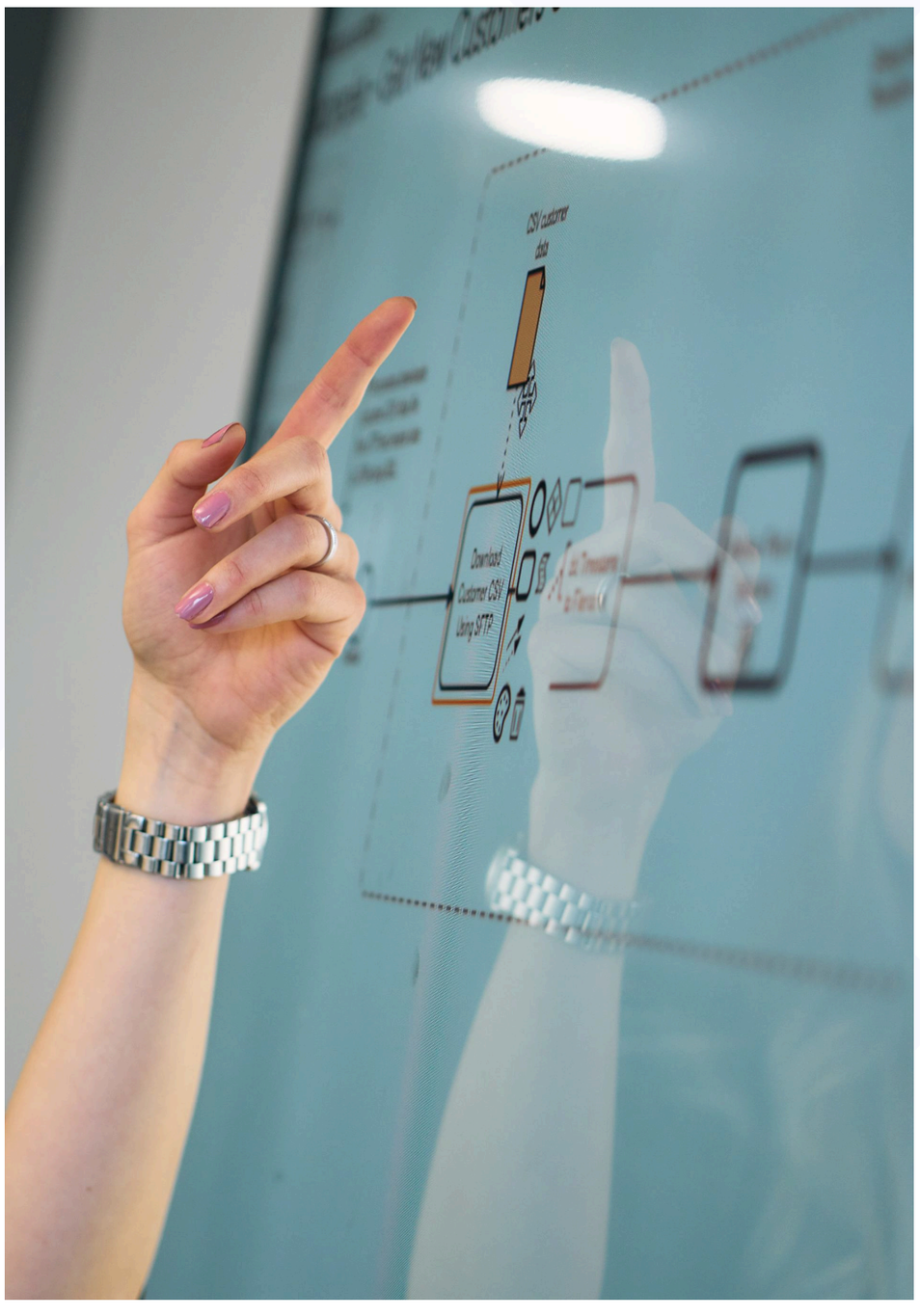
This approach does not scale well in large enterprises. The central team often becomes a bottleneck, unable to keep up with the integration demand from various projects. The more the business relies on digital processes, the more integration work piles up, and a finite central team can't deliver fast enough. This leads to long queues, frustrated business units and sometimes projects going ahead without proper integration (or doing quick-and-dirty workarounds). Additionally, centralized teams might lack the full context of each business domain, resulting in solutions that are technically sound but not optimally aligned with business needs.

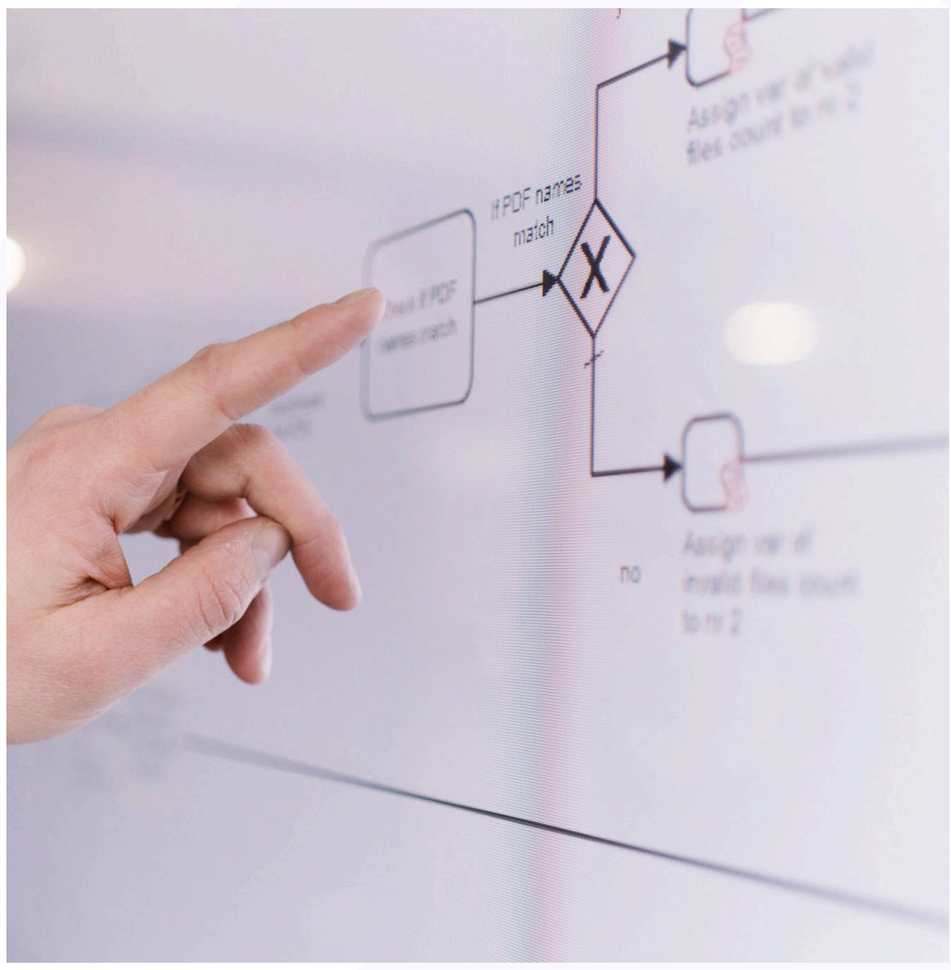
Role of C4E:

If an organization is currently here, the C4E's job is to gradually break the logjam by introducing enablement.

The C4E might start as essentially the existing central team but will focus on creating playbooks, templates and governance that allow involvement from other teams.

Over time, responsibilities can be federated. The C4E can still manage critical integrations centrally (especially in highly regulated environments or where specialist skills are non-negotiable), but it should actively push to empower other teams for less critical work.





Stepping stone to the modern view

A fully centralized, monolithic integration team is generally seen as counterproductive to speed and agility in the digital age. Unless your enterprise integration needs are very small, this model will likely hinder digital transformation.

The recommendation is to use it only as a stepping stone or for specific scenarios (e.g. core banking integrations in a bank might remain central due to risk, but even then, satellite teams can handle other integrations).

The modernization of ICC-style centralized approach is to shift to a distributed, yet centrally controlled, delivery model – **Integration Center For Enablement.**

In summary, centralized integration delivery equates to the old ICC model – great for control, poor for speed. The C4E's mission is to evolve the organization beyond this, retaining the good (common standards, consolidated expertise) but alleviating the bad (single-threaded delivery).

6.2 Federated Delivery with Platform Guardrails

A **federated model** is a hybrid approach: integration development is spread across multiple teams (often aligned to business units or product lines),

but a central C4E (or CoE) provides a unified platform and sets guardrails. You can think of this as a **"shared services"** model – the C4E offers integration as a shared service (providing tools, environments, guidelines and maybe some central resources), and **distributed teams** actually build and run the integrations in their areas, in collaboration with the C4E.

Advantages

This model significantly improves the scalability of integration delivery. Multiple teams can work in parallel on different integration projects, reducing bottlenecks. At the same time, because they share a platform and standards, the outputs remain consistent.

It **encourages ownership** at the team level – those who know the business domain best (say, the HR IT team for HR system integrations) actually do the work, resulting in solutions better tailored to business needs.

The central C4E still maintains **visibility and control** through the platform: they might monitor all running integrations and ensure that only approved tools are used, etc., but they are not micromanaging every project.

Structure

Often in this model, the central team is smaller and focuses on governance and platform ops, while "dotted line" relationships link them to integration developers embedded in other teams. For instance, each business unit might have one or more integration specialists (or just developers trained in integration) who are part of that unit's IT and the wider C4E community. They adhere to the standards set by C4E and use the shared integration infrastructure.

Guardrails in practice

The C4E might enforce guardrails like code reviews for any integration before deployment (maybe a central architect signs off or an automated scan is run), mandatory use of the central Git repository and CI/CD pipeline, and compliance checks (security, QA) that are built into the process.

Within these guardrails, however, the business unit teams have autonomy on how to design and implement their specific solutions.

Challenges

Federated models require good **communication and training.** There's a risk that different teams might diverge in practices if the governance isn't strong. The C4E must invest in keeping the distributed teams up-to-date on best practices and in facilitating knowledge exchange among them (e.g., regular sync meetings or an internal forum to discuss integration patterns).

Another challenge is resource allocation, ensuring each business unit has people with the right integration skills. Sometimes federated models start with "centers of excellence" in each BU plus an enterprise CoE – that can be too siloed. The C4E should instead foster a community of practice where all integration developers, regardless of who they report to, feel part of one network.

When to use

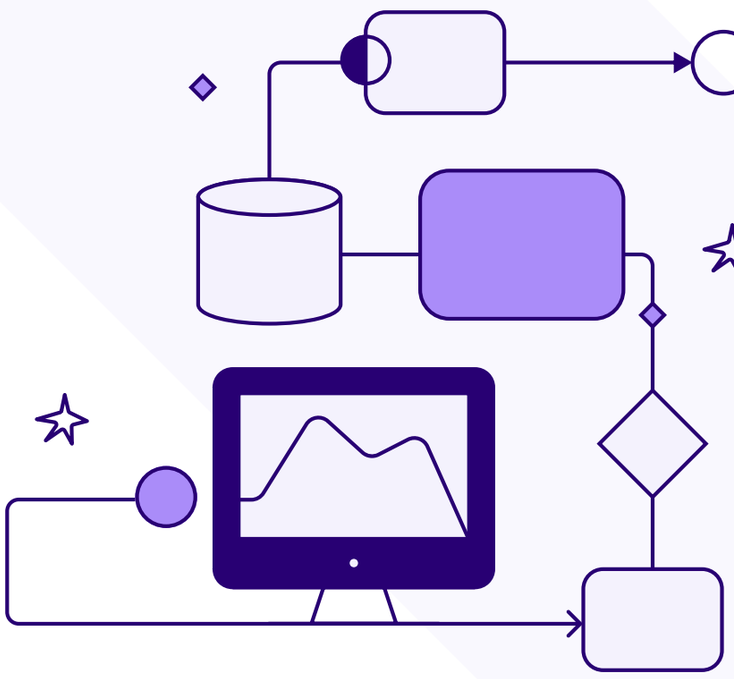
This model is often a transitional stage for organizations on the journey to full enablement.

It's very practical for large enterprises – you maintain a strong central governance via platform and policy but achieve scale by leveraging many hands in many teams. It's also effective when different domains have specialized knowledge; letting them handle their integrations (with support) results in better outcomes than a distant central team trying to understand every domain.

The federated model aligns well with the C4E philosophy. It acknowledges that "not every organization is ready to completely decentralize overnight" – factors like risk, regulatory environment and internal culture might necessitate a gradual approach.

Federated delivery provides a controlled way to decentralize. The playbook for C4E in this scenario is to clearly define the division of responsibilities (what the central team owns vs. what the distributed teams own) and to heavily enable those distributed teams (through training, clear standards, and readily available support).

Over time, as confidence grows, the guardrails can become more automated and less manual, allowing for even faster delivery.



6.3 Self-Service and API-first Teams – enabled by C4E

In **the self-service** model, integration capabilities are so ingrained and the platform so user-friendly that teams across the organization can accomplish integration tasks with **minimal direct involvement from the C4E.**

This represents a high level of maturity. Essentially, the C4E has made itself "invisible" by embedding integration know-how into tools and processes used daily by development teams. Every development team is an "integration team" in this approach. Characteristics of self-service integration include:

API-first development culture

Teams design and expose APIs as part of building any new functionality. It's a default behavior, just like writing unit tests. This means when a team creates a new service or application, they automatically consider how others will integrate with it (and thus produce APIs or events for consumption).

Accessible integration tools

The integration platform (e.g., iPaaS, API management, message brokers, etc.) is self-service. Developers can log in, create integrations or APIs, deploy, and monitor them without needing the central team to intervene. The platform provides out-of-the-box templates and wizards that even those who are not integration specialists can build standard integrations (for example, a simple data sync between two SaaS systems) following best practices.

Automated compliance

As noted earlier, standards enforcement is largely automated. Quality gates, security scans, and templates ensure that even when teams self-serve, they are doing the right things by default. If a team tries to deploy something out-of-policy, the system flags or blocks it automatically.

Minimal central oversight

The C4E central team in this scenario might be very small – a handful of architects and platform engineers – because they are not managing projects, only the framework. They intervene only on exception or to roll out new capabilities. The organization might not even refer to them as a separate team anymore, as their functions are part of the standard SDLC and DevOps processes.

Innovation and independence

This model fosters a high degree of innovation. Teams can experiment and implement integrations rapidly. It's akin to how cloud self-service enabled teams to provision infrastructure on their own rather than waiting for IT – here, teams integrate systems on their own rather than waiting for an integration specialist.

Pros:

This is the fastest and most scalable model. If done well, it achieves both agility and compliance (through invisible governance). It enables mass reuse and collaboration, since everything is built on common standards, and everyone is contributing to the integration ecosystem. It can also lead to high morale among development teams – they are autonomous and not blocked waiting for another team.

Cons:

It requires upfront investment in platform, automation and culture. Not every organization will reach this stage immediately. Without sufficient automation, attempting self-service can backfire – you might end up with inconsistency if people aren't following guidelines. Therefore, self-service works best after a period of federated model where patterns have been established, and tooling is mature. Additionally, some highly sensitive integrations might still require central oversight (e.g., anything with very sensitive data or compliance implications might remain gated).

C4E's role:

In a way, the ultimate success of a C4E is to make integration so natural that the C4E team's direct involvement shrinks. But the C4E doesn't disappear – it transitions to focusing on platform improvement, new technology R&D and governance monitoring. It also continues evangelizing best practices. A self-service environment still needs someone to maintain the "self-service portal". Also, a C4E would measure things like how many integrations are built by teams independently, the reuse rates, and if any teams are struggling (then go and help them).

Example of self-service:

Consider a company that has an internal API portal and an automated pipeline for API deployment. A developer needs to create an integration to onboard a new customer and connect their systems directly to the company's core services. They log into the portal, select a "Customer Onboarding" integration template (which enforces company standards for data mapping and security), fill in the customer-specific details, and utilize pre-built connectors for the customer's systems. With one click, the integration is deployed. The system automatically configures data transformations, applies security protocols, registers the integration in the system catalog, and initiates testing. The result is a streamlined, secure connection to the new customer's systems, achieved rapidly and efficiently. The C4E's role is to provide the templates, connectors and governance policies that enable this self-service approach. This frictionless experience, where "integration" is as simple as spinning up a server in the cloud, exemplifies the shift from a traditional ICC to a modern C4E model.

6.4 DevOps-Integrated Integration (CI/CD and Automation)

This is less a separate “model” and more **a crucial practice that overlays the above models** (especially federated and self-service).

DevOps-integrated integration means that building and deploying integrations is fully integrated into the same continuous integration/continuous delivery (CI/CD) pipelines and agile processes that application development follows. Integration shouldn’t be treated as something outside the normal development flow. In a modern setup, they’re built and deployed just like any other software using standard DevOps tools.

In an ICC world, it was common for integration to have a slower, separate release cycle (because of centralized control). In a C4E world, integration assets (APIs, integration flows, etc.) go through the **same automated build and release process** as any code. Key aspects include:

1 Source control

All integration code or configurations (for example, Frends API specifications, logic app definitions, Boomi processes, Frends workflows, etc.) are stored in Git or an equivalent version control system. This enables versioning, peer review via pull requests, and rollback if needed – just like application code.

2 Continuous integration

Whenever a change is made to an integration artifact, automated builds and tests run. For an API, this might mean static code analysis (linting the API spec against standards), running unit tests for any custom code in the integration, and perhaps spinning up a test instance to run integration tests (like calling the API with stubbed systems). Integration-specific tests (like mapping validations, contract tests with a downstream system mock, etc.) are part of the pipeline.

3 Continuous deployment

Whenever a change is made to an integration artifact, automated builds and tests run. For an API, this might mean static code analysis (linting the API spec against standards), running unit tests for any custom code in the integration, and perhaps spinning up a test instance to run integration tests (like calling the API with stubbed systems). Integration-specific tests (like mapping validations, contract tests with a downstream system mock, etc.) are part of the pipeline.

4 Automated quality gates

The pipeline enforces the governance discussed earlier. For instance, if a developer tries to deploy an API that doesn't meet the naming convention or lacks a required test, the pipeline can fail the build. Only integrations that meet the quality bar are allowed to proceed. This ensures that compliance is checked continuously, not just at go-live.

5 Environment consistency

Using DevOps practices, one can ensure that the integration runtime environments are configured consistently (using containerization or scripts). This avoids the "it works on my machine" syndrome for integration flows. If using containers or cloud functions for microservices, the deployment of those is scripted and standardized.

6 Monitoring and ops in DevOps

Integration DevOps also extends to having integrated monitoring, logging and alerting. For example, the same APM (Application Performance Monitoring) or logging solution used for apps is also collecting integration logs. Developers get alerts if their integration flow fails a deployment or if it throws runtime errors in production, closing the DevOps feedback loop.

Why is this important? Because in a microservices and multi-platform landscape, **rapid, reliable changes are needed.**

If every integration update requires manual deployment or waits for a specialized integrator, it slows down the whole software delivery lifecycle. By automating integration delivery, **you align it with the high-frequency release cadence modern businesses need.**

The C4E should collaborate with the DevOps or platform engineering teams to ensure the integration tools are plugged into the enterprise's CI/CD framework. This might involve selecting tools that have good CI/CD support or writing scripts to bridge gaps. For instance, if using a particular iPaaS that doesn't natively integrate with pipelines, the C4E might develop a custom script or use the tool's APIs to enable automated deployments.

DevOps integration example:

A development team working on an e-commerce site makes a change that requires a new API endpoint in the Order Management integration. They update the integration code in Git. The CI pipeline triggers: it lints the API spec (finding no issues thanks to templates), runs tests (all pass), then automatically deploys the new API to a staging environment where it's tested with the front-end. After automated and manual tests, the team merges to the main branch, and the pipeline promotes the API to production, updating the API gateway and documentation. All of this might happen in hours, with full traceability. The C4E's influence is in the pre-built pipeline jobs and templates that made it easy for the team to integrate this into their workflow, rather than a separate process.

In essence, **DevOps-integrated integration** ensures **that the culture of DevOps – rapid, iterative development with continuous improvement – extends to integration work.** It dovetails with the self-service model: teams can deploy and manage their integrations just like any microservice. It also supports federated models, by giving all teams a common pipeline. For leadership, this means integration is no longer a slow lane; it's fully part of the high-speed digital delivery highway.

6.5 Microservices – use with care

Microservices have been a buzzword and a trend for years, **promising greater agility and scalability** by breaking down monolithic systems into smaller, independently deployable services.

However, with experience, the industry has learned that microservices come with their own challenges. A C4E playbook must caution against the **misuse or overzealous adoption of microservices,** which can lead to **sprawl and operational burden** rather than efficiency.

Each microservice is effectively a standalone project – it has its own codebase, pipelines, infrastructure, and must be monitored and supported independently.

When an organization ends up with dozens or hundreds of microservices, the complexity of managing them can grow exponentially.

Common pitfalls include:

1 Microservice sprawl

Developers might slice applications into too many microservices (sometimes hundreds), each doing a very granular function. While each microservice might be simple, the system as a whole becomes incredibly complex. Teams can become overwhelmed if they need to keep track of what service does what, the network interactions and interdependencies. As a Cortex.io article noted, “each new service that’s built brings even more complexity to your architecture”.

2 Operational overhead

Every microservice needs deployment, scaling, monitoring, logging and failure recovery. The overhead of setting up proper CI/CD, observability and on-call support for numerous small services is significant. Without strong platform support, teams may struggle to provide production-grade support for all these services.

3 Inconsistent implementations

If each team builds microservices in their own preferred way (different languages, frameworks, logging approaches), the heterogeneity adds friction. Lack of standardization can lead to some services not meeting quality standards or security requirements. It also lowers the “bus factor” (the number of people who fully understand a given service) – often only the original developers understand their microservice, making support difficult

4 Integration burden

Microservices are heavily dependent on integration as they communicate via APIs, events, etc. The more services there are, the more integration points. It’s somewhat ironic: microservices were meant to simplify development by isolation, but they amplify the importance of integration. Without careful design, you can end up with a messy web of APIs (sometimes called “distributed spaghetti”). The C4E needs to ensure API interfaces between microservices are well-designed and managed, and that services are not duplicating each other’s functionality.

Given these challenges, the playbook advice is to use microservices judiciously: **not every component needs to be a microservice.**

Sometimes a modular monolith or a clustered application can suffice and be simpler. If microservices are chosen, design them around clear **business capabilities or bounded contexts,** not arbitrary technical breakpoints. Ensure each microservice is meaningful and relatively sizable in terms of function (too fine-grained is a red flag).

The C4E should issue **guidelines for microservice architecture** as part of integration architecture governance. For instance, guidelines can include:

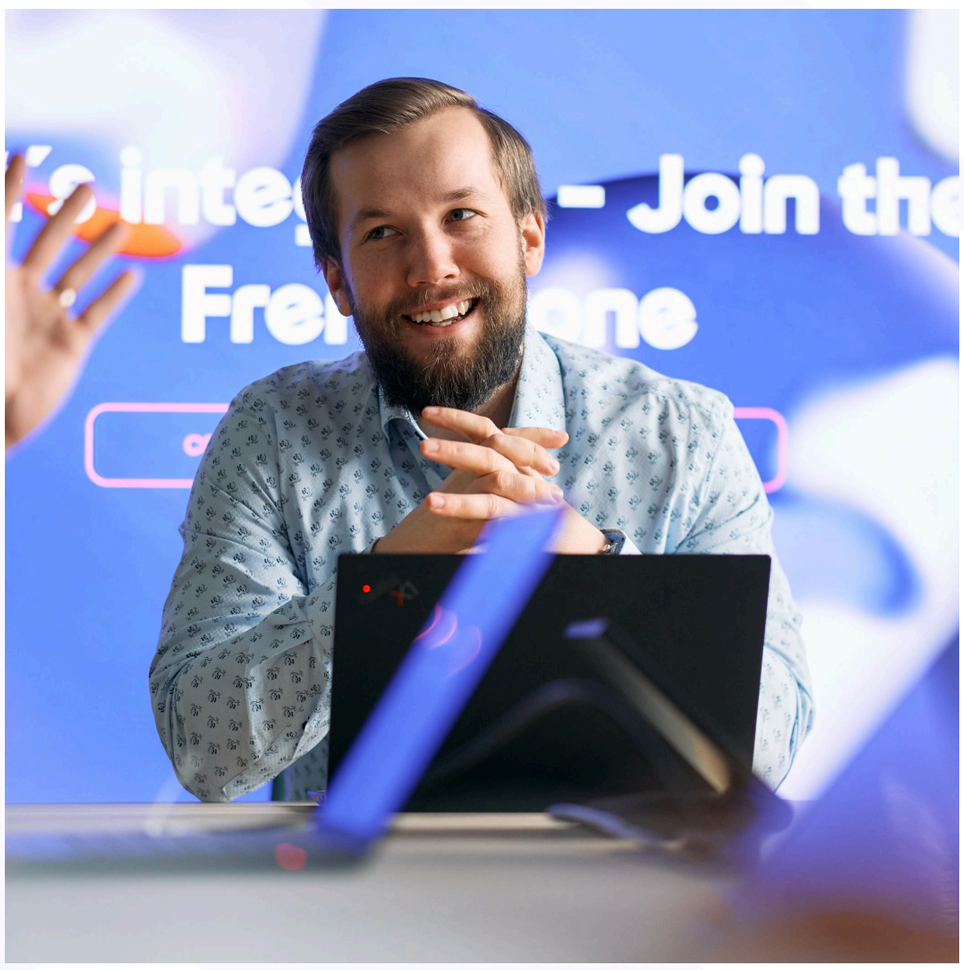
1. **Do not create a microservice for trivial code** that could live with other services. Only separate if there is a strong justification: independent scaling needs, clear ownership boundaries, etc.

2. **Each microservice must adhere to the same standards** as any product: proper documentation, logging, monitoring and security. If a team cannot commit to operating a microservice with these, it shouldn't be a microservice.

3. **Consolidate where it makes sense:** If two microservices are always changed together or one cannot function without the other, that might indicate they should be one service.

4. **Plan for orchestration:** More microservices mean a growing need for orchestration or choreography (using workflow engines or event buses). The architecture should include how these services will integrate (synchronous REST calls, asynchronous events, etc.) and the C4E can provide frameworks for these (e.g. an enterprise service mesh or messaging standard).

It's also important to highlight real-world lessons. Many big tech companies that pioneered microservices did so with significant investment in internal platforms to handle them (think Netflix's engineering tools, or Google's Borg / Kubernetes). Enterprises should not blindly emulate microservice counts without investing in automation and reliability engineering. In fact, there have been high-profile cases (like a well-known Amazon Prime Video team's reversion to a monolith for certain use cases) that illustrate microservices are not a silver bullet; they introduced too much latency and complexity, and **a consolidated approach proved simpler** in that case.



From the C4E perspective, one practical approach to mitigate microservice sprawl is to promote **service standardization and common DevOps pipelines** (as discussed earlier).

Establishing consistency in how microservices are built and managed can drastically reduce the operational burden of supporting them.

For example, **having a common microservice template** (with built-in logging, health checks, error handling) can save each team from reinventing that plumbing in ten different ways.

A standardized approach allows the broader team to understand services more easily and even share the on-call burden, since the operational aspects are uniform.

To sum up, microservices should be approached with a balance of enthusiasm and caution. They align well with agile, decentralized development – but only if your organization (and C4E) provides the **engineering rigor and platform support** to manage them. The playbook recommends that architects (possibly via an Architecture Review Board with C4E representation) evaluate the granularity of services early in a project. When in doubt, err on the side of fewer, more multi-functional services that can always be split later, rather than an explosion of tiny services upfront. And whenever microservices are built, **design for integration:** clear API contracts, backward compatibility for changes, and robust monitoring from the get-go. This disciplined approach will prevent the dark side of microservices – runaway complexity – from undermining the benefits of autonomy and speed.

6.6 Policy-Enforced Integration via iPaaS (Microservices through iPaaS)

One of the key recommendations of a modern C4E is to leverage **Integration Platforms as a Service (iPaaS)** to deliver integration solutions in a controlled yet flexible manner.

An **iPaaS** can serve as a powerful enabler for **building "microservice-like" integrations** – essentially lightweight, focused integration processes – **without incurring the full overhead of custom microservice development.**

It's an approach to get the best of both worlds: **the autonomy and distribution of microservices, and the governance and ease-of-use of a centralized platform.**

How using an iPaaS can help implement a scalable integration architecture:

1 Rapid development with built-in best practices

iPaaS solutions provide low-code or config-driven ways to build integration flows and APIs. This means developers (or even business analysts in some cases) can compose integrations via visual designers or templates. The advantage is that the iPaaS inherently applies many best practices – for example, data mappings, error handling frameworks and connectors are pre-built and tested. A team using an modern iPaaS, that offers reusable integration blocks, citizen integrator portals and templates (prepackaged integration processes) like Frends, can create a "mini-service" to sync customer data between two systems within hours, and that service will automatically include logging, retry on failure, and so forth as provided by the platform.

2 Autonomous agents, centrally managed

Many modern iPaaS (Frends included) allow deployment of agents or runtime nodes that can run integration processes closer to where the systems are (on-premises or specific cloud region), giving you distributed execution. Each agent can function independently to run the workflows assigned to it (even if temporarily offline from the central control), much like a microservice running in a container. However, these are all centrally managed – the C4E can push updates, enforce policies, and monitor all agents from a single control plane. In effect, you get a fleet of microservices (the flows on various agents) without having to manually handle each one's infrastructure and monitoring; the iPaaS takes care of that.

3 Policy enforcement and governance baked in

A good iPaaS will let you define global policies (security rules, naming conventions, logging formats) and apply them automatically. For instance, if company policy states that all APIs must check a user's authentication token, the iPaaS API gateway component can ensure no API endpoint is deployed without that check configured. This built-in governance means teams can't easily "go rogue". Even if they're quickly building something, the platform guards compliance.

4 Observability and centralized monitoring

With hundreds of custom microservices, gathering logs and performance metrics can be a huge effort. But if those “microservices” are built on an iPaaS, the platform usually provides a unified monitoring dashboard, where you can see all processes, their success/failure rates, throughput and so on – all in one place. Alerting can be standardized. Essentially, the operational burden is abstracted away by the platform – you don’t need separate monitoring setup for each service. This is a big win for the ops teams.

5 Scalability and performance management

iPaaS platforms handle scaling by allowing more agent instances or leveraging cloud elasticity. The C4E can set up the platform in a way that if integration workloads grow, it’s just a matter of configuration change or an automatic scale event, rather than each team figuring out scaling individually. Also, capacity planning can be done globally (monitoring overall usage) rather than per microservice.

6 Consistency across technologies

If multiple programming languages or frameworks were used in custom microservices, it is difficult to ensure consistency. With iPaaS, the platform provides a consistent execution environment. Whether a team is integrating SAP or a REST API or an IoT device, they do it using the same platform primitives. This greatly reduces the variability in how services are built across the enterprise.

7 Faster onboarding and delivery

New developers can get up to speed faster on a standardized platform. Also, building an integration in an iPaaS is often quicker than coding from scratch – there are pre-built connectors for common apps, drag-and-drop mappers for transforming data, etc. This speaks to the earlier point of accelerating time-to-market via reusable components. It’s not unrealistic for the C4E to establish an iPaaS-based “integration factory” where frequently needed integrations (like syncing customer records to a CRM or onboarding an employee in multiple systems) are delivered rapidly by configuring templates.

Example use case:

Suppose a company needs to implement a series of microservices for an e-commerce platform: one service to handle orders, one for inventory updates, one for sending notifications, etc. Using a traditional microservice approach, you'd write separate applications, set up separate CI/CD, container deployments, monitoring for each – significant overhead. If the company uses an iPaaS (say Frends or Boomi or MuleSoft) for this, the integration developers can create each "service" as an integration process in the platform: e.g., an Order Process that exposes an API to take orders and orchestrates DB and CRM calls, an Inventory Process triggered by events, a Notification Process that calls an email/SMS API. Each of these can be deployed to a runtime agent near the systems (maybe on Azure cloud for e-commerce stack). They behave like independent services from a functional perspective, but the platform centrally manages them. Security (like API keys, OAuth) is configured through the platform's common interface, logs from all go to the central log aggregator of the platform, and if any process fails, it can alert the support team via the platform's alerting engine.

From the outside, it looks like a suite of microservices; from the inside, **it's all orchestrated by the iPaaS.**

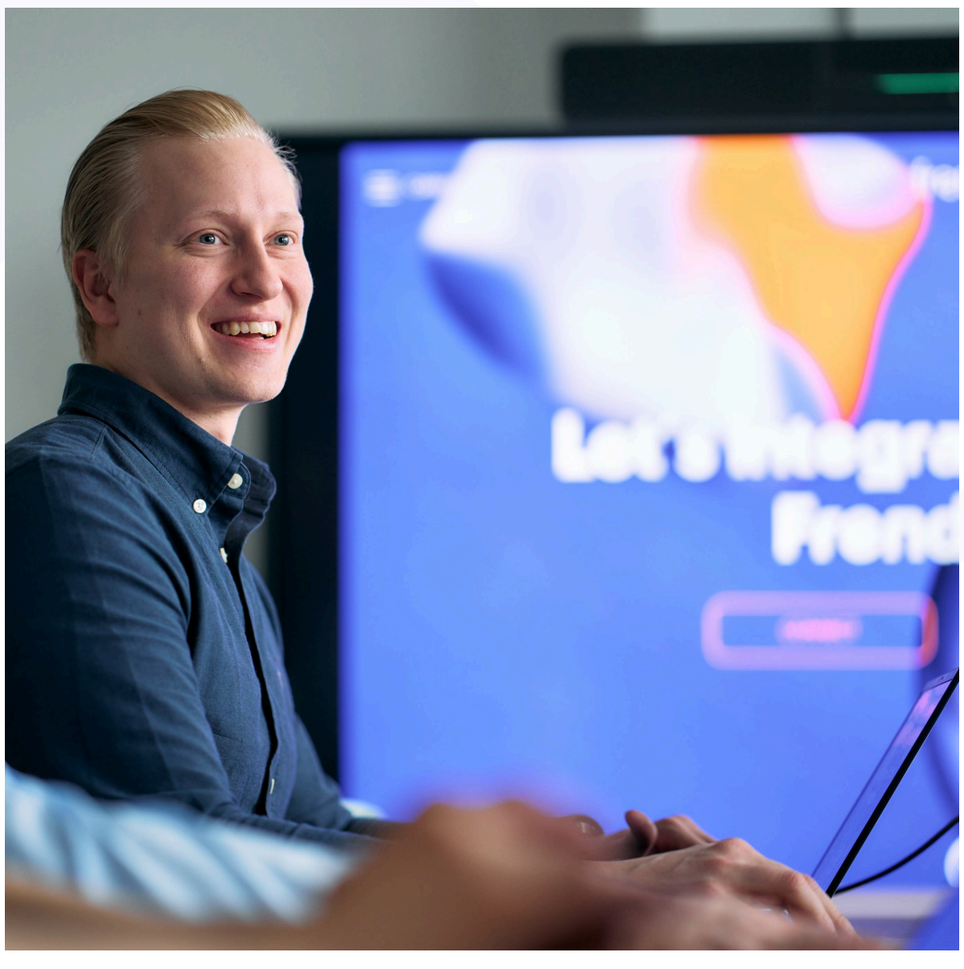
This approach often **reduces the headcount and skill required to manage the environment** – you don't need a full DevOps pipeline per service or deep cloud expertise in each team, because the platform handles many of these aspects.

Frends allows deploying agents on-prem or in cloud that execute workflows autonomously, with central management.

This aligns well with a scenario where you have many small integrations distributed across locations.

You deploy a Frends agent at each site or network zone, build your microservices in the Frends UI, and the central C4E team can enforce global policies and collect logs from all agents.

This means, for instance, if you have 50 microservices across global offices, you're not worrying about 50 separate servers and apps – you might have, say, five agents each handling ten processes, all observed by the central. If an agent goes down, central knows. If a policy needs updating (like a new password rule), you update in one place.



In summary, using an iPaaS to implement microservices and integrations can **significantly reduce the unmanaged sprawl.**

It offers a way to implement a distributed architecture but with a centralized brain, so to speak.

The C4E should evaluate the iPaaS options available and choose one (or a few) that meet their needs (consider factors like cloud vs. on-prem, supported connectors, cost, etc.). Once in place, the C4E will act as the **product owner** of that integration platform (as discussed earlier in Platform thinking), continuously improving it and guiding teams to use it effectively.

The platform, in turn, will enforce much of the governance automatically and provide the observability needed for trust. This strategy enables having many "micro-integrations" running freely – each solving a specific problem – without becoming a governance headache. Essentially, it channels the power of microservices within a **controlled environment.**

6.7 Agentic AI–Driven Integration: Prompting and reusable API tooling

What is Agentic AI in the C4E context :

Agentic AI represents a new class of intelligent automation that not only reasons over data but **acts autonomously** by chaining multiple "thought" steps into executable actions. Unlike traditional RPA or simple API-first models, an agentic system can:

1. Listen for events or triggers (e.g., incoming tickets, API calls)
2. Plan a sequence of tasks via a chain-of-thought orchestration
3. Invoke reusable integration assets (APIs, connectors) as "tools"
4. Execute end-to-end flows with minimal human supervision

Triggers & chain-of-thought orchestration

In C4E, we extend our self-service and DevOps-integrated models by embedding an orchestration "brain" that:

1. Detects an event (e.g., "New invoice arrived," "Support ticket created")
2. Invokes a prompt to outline the necessary steps in BPMN 2.0 notation
3. Calls AI reasoning (LLM) to plan and refine each step
4. Dispatches Frends tasks to implement decisions (e.g., data lookups, sys
5. **This prompt-chaining** approach (see Figure 3 on page 9 of the Frends whitepaper) enables the agent to adapt dynamically to varying inputs and use cases.

AI Toolboxing & reusable API tools

A cornerstone of C4E is treating integrations as **products** — versioned, documented and discoverable via an internal API catalog. In an agentic AI model, these APIs become tools the agent can **prompt-invoke:**

1. Toolboxing (Frends 6.x roadmap): define which APIs (and users/roles) the agent may call in a given context.
2. Computer Use Access (CUA): grant the agent credentials to interact with third-party UIs or systems when no API exists.
3. By surfacing Reusable APIs in the catalog as callable "tools" we empower business users to compose high-level prompts ("Generate monthly sales report, notify stakeholders, and archive results in SharePoint") that the agent translates into API calls, data transformations, and notifications.

Business-User Prompting Experience to make agentic AI accessible:

1. Pre-built prompt templates for common patterns (ticket triage, invoice processing, order orchestration)
2. Low-code UI where users select a template, fill in parameters (e.g., date range, recipient list), and hit "Execute"
3. Transparent audit trails showing each AI "thought" and API invocation
4. Governance hooks enforcing SLA, security and compliance checks at every step
5. This keeps the C4E's enablement ethos intact — teams own their processes, with the C4E coaching on best practices and guardrails.

Getting Started: Low-Hanging Fruit

Begin with **high-volume, well-bounded** processes that today still require manual review or rule-heavy logic, such as:

- Invoice exception handling (AI summarizes and recommends approval)
- Support-ticket categorization (AI classifies and routes automatically)
- Data-entry tasks (AI reads unstructured inputs, invokes APIs)

Steps to pilot:

1. Identify manual review steps in your automation backlog
2. Wrap an AI “thought” step around the decision point
3. Register underlying integration APIs as agent “tools” in the C4E catalog
4. Publish a prompt template and train business users
5. Measure time savings, error reduction and agent-invocation metrics
6. Over time, the Agentic AI model becomes another Delivery Operating Model within C4E, complementing self-service, DevOps and policy-enforced paradigms, while unlocking true autonomous digital workforces.

07

Consistency across multiple integration platforms

It would be ideal if an enterprise could standardize on a single integration platform and stick to it. However, the reality in large organizations is often more complex – multiple integration tools and platforms end up being used.

This can happen due to historical reasons (legacy systems with their own integration brokers, previous strategic choices), acquisitions (bringing in a different tech stack), or simply choosing the right tool for different types of tasks (for example, using one iPaaS for real-time APIs and another for big data batch transfers).

Acknowledging the reality of multiple iPaaS platforms is important. The C4E's challenge is to maintain consistency and avoid duplication across these diverse tools.

Some scenarios leading to multiple platforms:

A company might have **Informatica** or **IBM DataPower** as older on-prem integration middleware, while newer projects use **MuleSoft** or **Frends**. Both continue to exist for a time.

Different departments might have independently adopted platforms: IT chose one standard, but a marketing SaaS team might use a simpler Zapier/Workato for certain automations, or a dev team might use Azure Logic Apps because they are heavily in Azure.

Specialized use-cases: Perhaps a high-volume data integration uses a distinct ETL tool, whereas event streaming is done via Kafka, and classic app integration via an iPaaS – all of these could be in play simultaneously.

Running multiple integration platforms **can fill specific gaps** or optimize certain use cases, but it undeniably **adds complexity in governance.** To ensure this doesn't devolve into chaos, consider the following C4E strategies.

1. Integration architecture governance

Establish an Integration Architecture Board under the C4E that reviews architecture for new integration projects. One of its tasks is to decide which platform should be used for a given integration requirement. For example, "If you need real-time request-response with our core systems, use Frends; if you need scheduled file transfers or simple SaaS-to-SaaS data sync, use Frends." By delineating use-cases, you prevent teams from using whatever they fancy and then duplicating efforts. This also helps teams know where to go. They won't try to build a real-time API on a tool meant for batch, etc.

2. Unified standards across platforms

The governance standards (security, design, logging, etc.) should be tool-agnostic at their core. The C4E should express them in a way that each platform's implementation can adhere. For instance, "All APIs must have authentication via OAuth/OIDC" – in Platform A that might be a policy, in Platform B maybe you manually configure it, but either way the outcome is the same. Or a naming convention: if the convention is that every integration flow ID starts with the department code, ensure that's followed in each platform. The C4E can create platform-specific checklists derived from the master standards to help local admins comply.

3. Cross-platform monitoring and cataloguing

Aim to create a **central catalogue of integrations and APIs** regardless of platform. This could be as simple as a spreadsheet or Confluence page at first, but ideally, a tool (some API management systems can catalogue APIs from multiple gateways). The idea is anyone can search one place to find out "do we have integration X connecting system Y to Z?" and get an answer, even if one is implemented on Azure Logic Apps and another on Frends, for example. This prevents duplication because teams can discover existing assets. Additionally, consider central monitoring if feasible: some enterprises route logs from all integration tools into a single SIEM or logging service (like Splunk or Elastic), tagged by source. This way, operationally, you have one pane of glass to see errors across all platforms.

4. Avoid duplicative integrations

A major risk with multiple platforms is two teams unknowingly build the same integration on two different platforms. To counter this, the C4E should require that any new integration project be registered with them (not for heavy approval, but for visibility). During design, the C4E (or the integration architecture board) can point out "hey, we already have an API for customer data on Platform X, you don't need to create a new one on Platform Y – just reuse or extend the existing one." Cultivating this awareness saves time and prevents maintaining two things that do the job of one. It ties back to the catalogue – if well maintained, duplication can be caught early.

5. Rationalize platforms over time

While multiple platforms may be reality now, the C4E should continually assess if all are needed. If two platforms provide similar capabilities, maybe plan to retire one after migrating integrations. Standardizing reduces cognitive load and costs. For instance, if you have both Dell Boomi and Azure Logic Apps doing similar light-weight integrations, and you determine Azure covers all needs, you might phase out Boomi. This is a longer-term strategy; it might not be immediate due to contractual or skill reasons, but having a roadmap helps. Communicate this to teams so they know where to invest their learning focus.

6. Leverage integrations between platforms

Sometimes, you can integrate the platforms themselves to avoid silos. For example, if you have an API management tool on one platform, you might still expose integrations from another platform through that same API portal (e.g., publish an Azure function API in Frends´s catalogue). Or use a message bus like Kafka to tie things – one platform drops messages on a topic, another picks them up. The C4E can design these cross-platform glue mechanisms to ensure the multiple tools act as parts of a larger cohesive integration architecture, rather than isolated islands.

7. Consistent team skills and processes

Host common trainings and ensure that integration developers on different platforms still feel part of one community. The C4E's efficiency is in the prebuilt pipeline jobs and templates that cover enterprise-wide best practices, not just tool specific how to's. It would be good idea to rotate staff or have them cross-review each other's work. A developer primarily on Platform A could benefit from understanding Platform B's style – it reduces key-person risk and increases the unity. Additionally, keep processes like deployment and change management similar. Even if Platform A and B have different tech, maybe both use similar branching strategies, both create design documents that go into a common repository, etc.

Running multiple iPaaS **does require more diligence for monitoring and management.** It's like managing a multi-cloud environment – complexity grows. But with the right governance and a C4E overseeing it, it's manageable. The goal is to mitigate the downsides: avoid silos (where each platform has completely separate ways of working), avoid duplication (doing same integration twice), and avoid inconsistency (different security levels or data definitions in different platforms).

Illustrative example:

A global enterprise might use Frends in its core IT, but one regional team heavily uses AWS and prefers native AWS services like API Gateway and Step Functions for integrations. The C4E can allow this duality but sets rules: e.g., "All customer-facing APIs, regardless of platform, must appear in the global API developer portal with proper documentation and use the global OAuth security" – so whether an API is actually on Frends or AWS, the consumer sees a unified front. Meanwhile, internal governance might say: "if integrating deeply with AWS data lakes, use the AWS-native pipeline; if integrating on-prem apps or third-party SaaS, use Frends." This specialization cuts down overlap. And the C4E monitors both environments for compliance and provides support. Over time, if AWS pipelines prove very effective, they might shift more to that, or vice versa – the C4E will make that strategic call.

In conclusion, multiple integration platforms are a reality that need not sink an integration strategy – **provided the C4E maintains a strong overarching governance.**

By enforcing common standards, sharing knowledge, and eliminating redundant effort, the C4E ensures that having two or three integration tools doesn't mean having two or three divergent integration **practices**. There remains one integration culture and **one set of goals** across the enterprise, with the C4E as the unifying force.

08

Conclusion:

Building a modern integration capability

Transforming a traditional ICC into a modern Integration C4E is not just a technology shift, but a cultural and organizational one. It aligns integration practices with today's needs for speed, scalability and flexibility, while still safeguarding the enterprise's need for security and reliability.

By implementing the approaches in this playbook, enterprises can expect to see **shorter delivery cycles, higher reuse of assets, better adherence to standards and improved quality** in their integration landscape. When teams can move faster and avoid unnecessary back-and-forth, the whole business benefits with fewer delays, fewer surprises and more value delivered.

A few **key takeaways** from this playbook for business and IT leaders:

Empower teams, but equip them

Enable your distributed product and IT teams to do integration work themselves by giving them the right platform, training and support. When teams have self-service tools and clear guidelines, they can deliver integrations much faster than a central queue ever could. However, don't just turn them loose – invest in the C4E function to provide the necessary guardrails and help.

Treat integration as a product and a capability

Just as you'd invest in improving a customer-facing product, invest in your integration platforms and APIs. Manage their lifecycle, measure their usage, and market their availability internally. This drives reuse (why build something twice?) and increases the ROI of every integration developed. Over time, you build a **library of integration assets** that make future projects easier – a competitive advantage in responding to change.

Diverse operating models can coexist

It's not one-size-fits-all. You might still run a small central integration team for the most critical systems (for risk management), while federating most work, and pushing self-service for standard SaaS integrations. That's fine. The C4E can support multiple modes simultaneously, acting as a flexible framework. The ultimate direction is toward more federation and self-service as maturity grows, but you can pace it according to your organization's readiness.

Governance is an enabler, not a roadblock

Modern integration governance is lightweight, automated, and principles-based. It's about setting teams up for success – **"secure by design, compliant by default"** – rather than catching mistakes at the end. When done right, governance actually speeds delivery (no last-minute security rebuilds) and ensures reliability. So, prioritize establishing those standards and automations early; it will pay dividends as you scale out enablement.

Mind the microservices (and the tools)

Embrace microservices and cloud integrations where they make sense, but do so knowingly. The C4E should guide architecture so you don't end up with more services than you can handle. Similarly, rationalize your toolset: use powerful iPaaS platforms to ease the operational pain of many integrations and converge on consistent ways of working even if multiple platforms exist. The focus should be on **reducing complexity** for teams, not adding to it.

For organizations that get this right, the reward is an integration capability that truly accelerates digital transformation. New customer experiences can be launched faster because back-end integrations are readily available. Mergers and acquisitions are integrated more smoothly. Internal innovation increases as teams can connect systems and data on their own to test ideas (within safe guardrails). Essentially, the enterprise becomes more **nimble and connected,** turning integration – often seen historically as a slow, backend concern – into a strategic asset.

Adopting the C4E model requires leadership support. It may involve reshaping team structures, investing in new platforms and re-training staff. It's important to communicate the vision: a shift from a "control tower" to a "force multiplier" approach. Early wins can be demonstrated by pilot projects where a federated team delivers an API in weeks rather than months, or where reuse of a component saved significant effort. Celebrate those and build momentum.

The Integration C4E playbook provided here is a guide; each organization should tailor it. But the core principle holds universally: i**ntegration should be enabled as a distributed capability, not bottled up.** By focusing on enablement, product thinking, multiple operating models, strong governance, prudent microservice use, iPaaS leverage and cross-platform consistency, any large enterprise can modernize their integration competency into something far more agile and impactful than the ICCs of old.

In closing, the journey from ICC to C4E is about empowering people as much as it is about technology. When your developers and analysts are enabled to connect systems and create APIs quickly (and correctly), they innovate. When they are supported by a central team that provides a great platform and clear standards, they excel. **Speed and safety, innovation and governance, can indeed coexist** – and the Integration C4E is the organizational construct to achieve it. With this playbook, you can begin that transformation and position your enterprise integration capability for the demands of the digital age.

frends

Thank you

for more, visit **frends.com**

Frends iPaaS
Where data flows, business grows.